2007

Recognizing Teamwork Activity In Observations Of Embodied Agents

Linus Jan Luotsinen
University of Central Florida

RECOGNIZING TEAMWORK ACTIVITY IN OBSERVATIONS OF EMBODIED AGENTS

by

LINUS J. LUOTSINEN
M.Sc. University of Central Florida, 2004
B.Sc. University of Dalarna, 2002

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2007

Major Professor: Ladislau L. Bölöni

ABSTRACT

This thesis presents contributions to the theory and practice of team activity recognition. A particular focus of our work was to improve our ability to collect and label representative samples, thus making the team activity recognition more efficient. A second focus of our work is improving the robustness of the recognition process in the presence of noisy and distorted data.

The main contributions of this thesis are as follows: We developed a software tool, the Teamwork Scenario Editor (TSE), for the acquisition, segmentation and labeling of teamwork data. Using the TSE we acquired a corpus of labeled team actions both from synthetic and real world sources. We developed an approach through which representations of idealized team actions can be acquired in form of Hidden Markov Models which are trained using a small set of representative examples segmented and labeled with the TSE. We developed set of team-oriented feature functions, which extract discrete features from the high-dimensional continuous data. The features were chosen such that they mimic the features used by humans when recognizing teamwork actions. We developed a technique to recognize the likely roles played by agents in teams even before the team action was recognized. Through experimental studies we show that the feature functions and role recognition module significantly increase the recognition accuracy, while allowing arbitrary shuffled inputs and noisy data.

To my family for their support, kindness and love.

ACKNOWLEDGMENTS

Thanks to: my advisor, Ladislau Bölöni, for his time, patience, motivation, and inspiration throughout the years; my dissertation committee, Avelino J. Gonzalez, Kenneth O. Stanley and Liqiang Ni for insightful questions and comments; Hans Fernlund, Daniel Barber and Charles Andrew Houchin for providing datasets to work with.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Humans have the ability to observe, recognize, analyze and occasionally adopt or adapt collaborative behavior. Successful team models, such as the Macedonian phalanx, Cromwell's new model army or the 4-2-4 formation of the 1958 world champion Brazilian soccer team represented a formidable advantage against unprepared opponents. Adversaries countered by studying these models, and either imitated or devised countermeasures against them. In our days, the organizational structures of terrorists groups pose significant challenges to homeland security; we are fighting an enemy which is organized along different patterns than we expected. Our ability to recognize teamwork is not restricted to humans: we can identify collaboration in wolf packs, flocks of birds or airplane formations.

Recognizing teamwork in embodied agents has important applications in areas such as surveillance, training, automated annotation and commentary, as well as improving the ability of robotic agents to participate in human teams. Most of the time, the teamwork of embodied agents have a natural expression in coordinated movement. Naturally, teamwork can have many other aspects, in addition to or replacing coordinated movement; examples are voice commands and wireless communication. However, the focus of this dissertation, is on recognizing team actions which are expressed in a coordinated movement pattern of agents with agents playing specific roles in the team. Examples of teams displaying such patterns are sport teams such as football, basketball or

soccer, dance groups, pickpockets in action, groups of animals such as wolf-packs, bodyguards defending a celebrity, military units in the battlefield or a MOUT environment, tank units, air squads, terrorists in a busy marketplace and many others.

In recent years, significant research effort has been directed towards recognizing team actions, and a general consensus has emerged. It was found that although some team actions can be recognized based on static configuration, for more complex team actions we need to consider the temporal evolution of the agents. Hidden Markov Models (HMMs) are one of the proven approaches to model temporal evolution of phenomena; indeed, a majority of team action efforts used HMMs or closely related models. The HMM is a representation of the *idealized team action* (ITA), which can be either knowledge engineered (hand crafted) or acquired through learning. These efforts had shown good recognition accuracy for datasets ranging from simulations to real world data recorded in controlled settings (such as military exercises). However, the existing systems are too brittle for most real-world deployments; frequently occurring perturbations can make the system unable to recognize the team actions it trained for. One additional challenge of the teamwork activity recognition is that for most applications, the recognized action needs to be matched with the human perception of the same action. As embodied team action is a domain where acquiring large corpuses of annotated data is extraordinarily difficult, the representation of the human perception cannot be introduced by learning for large number of human classifications; rather it needs to be engineered in the way in which the features of the classification are extracted.

This dissertation describes our ongoing efforts to improve the performance of teamwork activity recognizer systems.

1.1 Problem Statement

The problem addressed in this research is how to accurately and robustly recognize teamwork activities from spatio-temporal sequences of observations. We limit our recognition system to embodied agents that exists in the physical world or in a simulation of it. Note that, acts of message passing, radio communication and speech can be used to recognize and monitor teamwork [GK04]. In this work, however, the focus is on recognizing teamwork in sequences consisting of features derived from spatio-temporal position traces such as velocity, acceleration, trajectory curvature and so on that depict the physical motion of the embodied agents in the team.

Most of the literature in this domain focus on representing teamwork in models that are generated manually using *a-priori* domain knowledge. In this dissertation, however, focus is on reducing the knowledge engineering effort by automatically learning teamwork and role models from representative observations.

The activity recognition system presented in this dissertation is closely related to the general problem of key-hole plan recognition. In plan recognition, high-level plans or recipes are recognized using sequences of actions in the symbolic domain. Activity recognition on the other hand recognize activities/behaviors using low-level observations. Furthermore, activity recognition is

generally applied to shorter sequences of observations and is therefore less concerned about the high-level plan that the agents are following.

1.2 Challenges

Let us now see some of the challenges faced by a teamwork activity recognition system, which is assumed to have a representation of the idealized team action (ITA):

(a) Observation noise.

(b) Alignment problems: translation and rotation of the team actions compared to the ITA.

(c) Scaling problems: the movement of the team members happen at a different physical scale compared to the ITA.

(d) Temporal scaling: the movement happens slower or faster than in the ITA.

(e) Terrain distortion: the movement patterns are distorted as an adaptation to the particularities of the terrain, such as obstacles or roads.

(f) Movement variants: the team action has several alternative ways of execution, which map to the same human label.

(g) Role count variants: teams with different number of agents perform the action with the same label (for instance n bodyguards surround a person).

(h) Uncertainty regarding the role of the agents in the team action. An important sub-case is the agents which appear in the scene but do not participate in the team action (agents in the bystander role).

(i) Agents changing their roles during the team action.

Different recognizer systems have already addressed at least a subset of these problems. The problem of observation noise (a) can be addressed by using Kalman or particle filters on the data stream. The alignment problems (b) can be handled by the normalization of the observations. This is usually done by transforming the agents into a coordinate system aligned with the centroid of the team. This, however, can be problematic if the team is not known in advance, as bystander agents might shift the position of the centroid. Furthermore, for many team actions, the physical centroid might not be the logical center of alignment of the team. For a team of bodyguards guarding a celebrity, the location and the movement of the celebrity is the logical center and direction of the team. Scaling problems (c) can be handled either through normalization or by training several HMMs for various sizes. To some level, the problem of temporal scaling (d) is already addressed by the HMM. However, certain movement patterns such as "alternative advance of the two sub-teams for arbitrary, but equal time" can not be encoded in the HMM, because it would require historical information, and the Markov property assumes that the evolution of the system does not depend on its history.

One example of movement variant (f) is the case when a group of agents perform a U-turn by turning to the left or to the right while maintaining formation. While this maneuver maps to

the same label, it requires very different movement patterns and role assignments. A simple way to handle such variants is considering them different actions internally and only labeling them identically at the user interface level. In other cases, however, the differences are more subtle. One problem, naturally, is that these variants are labeled identically in training set. Learning techniques based on unimodal distributions might not be able to accommodate these variants.

1.3 Contributions

The contributions presented in this dissertation are enumerated below.

- We developed a software tool, the Teamwork Scenario Editor (TSE), for the acquisition, segmentation and labeling of teamwork data.
- Using the TSE we acquired a corpus of labeled team actions both from synthetic and real world sources.
- We developed a set of team-oriented feature functions, which extract discrete features from the high-dimensional continuous data. The features were chosen such that they mimic the features used by humans when recognizing teamwork actions. The specific set of features address challenges (a), (c), (e) and, to some level (g).
- We developed an approach through which representations of idealized team actions can be acquired in form of Hidden Markov Models (HMMs) which are trained using a small set of representative examples segmented and labeled with the TSE.

- We developed a technique to recognize the likely roles played by agents in teams even before the team action was recognized. We prefix the actual team action recognition with a role recognition module, which allows us to present the recognizer with arbitrarily shuffled input, and still obtain high recognition rates. This component directly addresses challenge (h), and represent a necessary precondition to address challenges (g) and (i), which, however, are outside the scope of this thesis.

- We have tuned the performance of the teamwork recognizer through the choice of probability density functions.

1.4 Applications

Applications that might benefit from our teamwork activity recognition system are listed below.

- *Training:* A team can be trained by closely mimicking the actions taken by an expert team. In such a training scenario, the teamwork recognition system can be employed to identify when and where the learner behavior is deviating from the expert team's behavior. The discrepancies can be used to evaluate the performance of the learning team and to provide important feedback to the team so that weaknesses can be addressed.

- *Surveillance:* Recognizing enemy activity in battlefields, airport security, border control, casino surveillance, and traffic monitoring are some of the application domains where teamwork recognition can play an important role in automated surveillance. The overwhelming

amount of continuous information flow in larger surveillance systems is time consuming and costly to process. The advantage of teamwork activity recognition, besides automating the recognition process, is that complicated activities where, for instance, the correlated movement of people in an airport can be used to recognize possible terrorist activity.

- *Smarter Agents:* Recognizing teamwork has several applications in the intelligent agent domain. Opponent modeling is important in, for instance, coaching agents that need to recognize the actions performed in adversary teams in order to select good countermeasure strategies. Another interesting application is when smart agents or robots are collaborating with humans. Here, the robots have to identify the joint actions performed by the human team so that they can automatically adapt and allocate tasks accordingly.

- *Automated Annotation:* Teamwork activity recognition can be used in many applications where large databases need to be annotated or tagged. Annotating databases collected from military exercises can significantly reduce the amount of time spent in After Action Reviews. In entertainment, automated annotation can be used as a service in Digital Video Recorders which would allow the viewer to "catch up" with live broadcasts without missing highlighted events, such as touchdown plays in football or overtaking in Formula-1 racing, in the recorded video sequence.

1.5 Outline

The remainder of this thesis is organized as follows.

In Chapter 2 we conduct a comprehensive literature review of related research.

Chapter 3 focus on the knowledge engineering and data acquisition aspects of training and evaluating teamwork activity recognition systems. The chapter present several datasets that are used to evaluate the methods and algorithms described in this dissertation. Furthermore, we introduce a knowledge engineering tool designed for visualization, identification, editing and extraction of teamwork activities.

In Chapter 4 a teamwork activity recognition system is developed and evaluated on a real-world dataset. During development of this system several shortcomings and challenges emerged that are systematically addressed in the remaining chapters.

Chapter 5 describes an approach for discrete feature extraction from the original data. The discrete features were carefully selected to provide a good match with the ways in which a human observer would understand and analyze the scene; furthermore, we are introducing and emphasizing features which refer to teams as a whole, rather than individual agents.

Chapter 6 describes a role recognition system, which allows the workflow to recognize the likely role of the agent which it would play in a certain team action.

Chapter 7 describes a study where the goal is to tune the performance of the teamwork recognizer by the choice of probability density functions.

Finally, we conclude and present future works in Chapter 8.

CHAPTER 2: BACKGROUND

This chapter discusses related work on teamwork activity recognition, multi-agent plan recognition and role recognition.

2.1 Teamwork and Multi-Agent Activity Recognition

Many teamwork activity recognition systems were developed for domain specific applications such as play recognition in football [IB01, LMZ05, LS01], automated commentary in soccer [HV99, VAH99, NTM04, VAC06], opponent modeling [KKS06, RVK02] and military [SS06a, SS06b]. These systems were characterized by their ability to recognize teamwork activity in short observation sequences, where the input observations consisted of low-level features such as agent position traces, velocity, acceleration and so on. In the following we review work where Bayesian networks and Hidden Markov Models were used to model teamwork action. We also present several related teamwork/multi-agent recognition systems which were used in domain specific applications.

2.1.1 Recognizing Team Actions with Bayesian Networks

Intille et. al. [Int99, IB01] developed and evaluated a system for recognizing multi-agent activities (plays) in American Football. Temporal logic and high-level descriptors were used to encode

football plays in Bayesian networks and promising recognition results were shown on a real-world dataset acquired from 10 different football plays. The extensive knowledge engineering effort that was required to represent the plays limited the practical use of this recognition system.

Sukthankar et. al. [SS06a] employed Hidden Markov Models to recognize spatio-temporal teamwork activities from simulated teams in the Military Operations in Urban Terrain (MOUT) domain. The recognizer, which consisted of one separate HMM for each teamwork activity, was manually configured through knowledge engineering techniques. The observation vector, which was fed as input to the recognizer, consisted of the normalized position and velocity measurements of each agent in the team. The experimental results showed that the HMM paradigm was able to accurately discriminate between three behavioral patterns executed by teams of two agents. In follow-up work Sukthankar et. al. [SS06b] developed the STABR algorithm for simultaneous team assignment and behavior recognition. The STABR algorithm used template matching to identify a set of candidate team assignments from the spatial relationships of independent agents. Candidates that did not follow any of the pre-defined, parametric, behavior models over a period of time were filtered out. The STABR algorithm was evaluated, with good performance, for teamwork behavior recognition in a simulated military domain.

Liu et. al. developed the Observation Decomposed Hidden Markov Model (ODHMM) for multi-agent activity recognition [LC03]. In ODHMM the observations were separated into sub-observations which represented the individual features associated with each agent. Hence, instead of a single high-dimensional observation vector, a set of low-dimensional observation vectors (one

for each agent in the team) were used as input to the recognition system. Experimental results showed that the ODHMM outperformed the standard HMM on an artificially generated dataset.

In a recent study by Vail et. al. [VVL07] Conditional Random Fields (CRF), which are related to HMMs, were employed for activity recognition. Results showed that CRFs can outperform HMMs for activity recognition in a simple test scenario. One drawback of the CRF approach was that it required significantly more training time as compared to the HMM approach.

2.1.2 Automated Commentary

Many automated annotation systems were developed in the context of the RoboCup competition. Han and Veloso [HV99] employed a probabilistic method to recognize behavior using recordings from RoboCup games. The recognizer was used as a soccer commentary system and it improved the behavior of the robotic soccer players (by making them able to adapt and make decisions based the behavior of the opponents).

In [VAH99] a commentary system, named Rocco, was developed for the RoboCup soccer competition. Rocco used an elementary set of predicates to describe the game state at a single point in time. An ensemble of 15 event templates were generated by combining the predicates to broadly cover the events that can occur in a soccer game. Automatic commentary (in text and speech form) were generated using template matching methods which identified the events as they were played in the soccer game.

In similar work, Nair et. al. [NTM04] developed ISAAC, an automated team analysis tool for RoboCup competitions. ISAAC was also evaluated using a simulated emergency evacuation scenario.

In [VAC06] a team of humanoid game commentators were developed to annotate the events in a robotic soccer game. The team consisted of two robots that coordinated their announcements based on input observations from vision processing and from wireless communication.

2.1.3 Computer Vision and Content-Based Retrieval

Computer vision research have for some time observed a gap between low-level video processing, such as segmentation, geometry detection and object tracking, and high-level detection of events in video sequences. In an attempt to bridge the gap Hakeem et. al. [HSS04] developed an event grammar that was able to express multi-agent temporal events with hierarchical dependencies. Predefined event templates, based on spatial coordinates, were used with a similarity measure (the Jaccard coefficient) to detect matching events in video sequences. A follow-up work [HS05] employed a combination of probabilistic and graph based methods to detect events in video.

Content based retrieval systems faced many of the challenges discussed in teamwork activity recognition. In fact, teamwork activity recognition can be directly applied in many content based retrieval systems. For instance, Liu et. al. [LMZ05] developed a content based retrieval system for American football. The system detected football plays in recorded sequences of video which allowed the viewers to efficiently watch only the highlights of the game. The implementation em-

ployed computer vision techniques for motion tracking and scene geometry extraction. A cascade of Ada-Boost classifiers were trained to ultimately recognize the plays.

In earlier content based retrieval systems the Hidden Markov Model was applied to extract highlights from video recordings of baseball and sumo wrestling sporting events [LS01].

Petkovic et. al. [PMJ02] described a system performing highlight extraction and content based retrieval using TV broadcasts of Formula-1 races. Their implementation employed a Dynamic Bayesian Network classifier. A multimedia database was created from three different information sources: speech, video and superimposed text. Queries for video sequences, such as "Retrieve all sequences where Räikkönen is in first position and Alonso is in second position", were executed by applying the classifier to the database. Results showed that 80% of the interesting race events (highlights) were detected using the speech and video cues.

2.1.4 Coaching and Opponent Modeling

Riley et. al. [RVK02] developed a coaching agent for robotic soccer. The coaching agent provided advice to the team by analyzing the opponent from recorded log-files of previously played games. While playing against an opponent team, the coach was able to recognize formations and roles (defender, midfielder, attacker) which were used by the coach to either advice its team to imitate the opponent or to direct its defenders to mark specific attackers. The coaching agent also learned a set of rules which defined the passing game of the opponent. These rules were used to predict where

the ball holder was most likely to pass the ball. Experimental results showed that the coaching agent improved team performance.

In [KKS06] Kuhlmann et. al. developed another soccer coach which modeled the opponent from previously recorded games in order to identify and exploit weaknesses in the adversary team. Their implementation also identified the formations of opponents teams. In addition play-by-play statistics such as shots on goal, passes and directed kicks were employed by the coach to advice its team.

2.1.5 Teamwork Recognition by Overhearing

Unlike the previously described approaches, which use low-level observations of physical motion, the overhearing approach recognizes teamwork by listening to the (social) conversation between agents in the team. Overhearing is employed in teamwork monitoring applications where the goal is to determine whether the team is progressing towards its joint goal or not [GK04]. Overhearing systems were also developed to perform complex tasks such as teamwork plan recognition [KPT02], building and maintaining group formations [LT03] and role recognition [RB04].

2.2 Multi-Agent Plan Recognition

In multi-agent plan recognition the task is to analyze sequences of activity to infer high-level plans or recipes. The multi-agent plan recognition systems can be either intentional or key-hole [Kau90].

In the intentional case the agents are deliberatively showing their intentions to the observer whereas in the key-hole case the agents are unaware of the observer. Key-hole multi-agent plan recognition is related to teamwork activity recognition. However, instead of using low-level input over short periods of time, the plan recognizer generally infer plans using high-level symbolic input extracted over longer periods of time. In this section we review work from the multi-agent (key-hole) plan recognition domain.

In [KPT02] Kaminka et. al. developed a novel method to monitor teamwork by inferring the team's state using a probabilistic multi-agent plan recognition system. Plans were recognized by observing to the social communication patterns exhibited by the agents. The plan recognition module was sufficiently fast to allow for real-time execution.

In [Suk07] a method was developed to recognize teamwork plans in dynamic teams by maintaining multiple hypotheses of team behaviors. The plan recognition system used node traces from pre-defined plan-trees as input to recognize the high-level goals of the team. The system was evaluated in a large scenario consisting of 100 agents and 40 simultaneously executing plans.

Tambe [Tam96] developed the RESC_{team} approach (an evolution of RESC [TR95] used for single agent tracking) specifically designed for real-time teamwork tracking. In RESC_{team} higher-level goals and joint intentions of teams were tracked using team models (which consisted of team states and team operators). The system was evaluated in synthetic environments involving helicopter simulation and fighter combat.

2.3 Role Recognition

Role recognition can provide insightful cues about teamwork activities which in turn can be used by the teamwork activity recognizer to improve robustness and recognition accuracy (See Chapter 6). In the following we review work where role recognition methods were employed to identify team-organization.

In multi-agent systems such as STEAM [Tam97] and Machinetta [TZ98, SOM05] a significant manual knowledge engineering effort was required to create team-plan hierarchies and team-organization hierarchies to model team behavior. To reduce the knowledge engineering effort, Prasad et. al. [PLL96] developed the L-TEAM framework which used a machine learning approach to infer team-organization hierarchies from observations. The team-organization knowledge allowed agents in the framework to automatically negotiate which role to play given the task.

In [RB04], Rossi et. al. proposed an approach for recognizing social roles of agents in a team using rule based systems. Social roles are assigned to agents by an overhearing agent that senses the communication between all agents in the team. The overhearing system was successfully validated against artificially generated organizations that followed well known protocols.

In [GBC02], Guessoum et. al. develop a fault tolerant multi-agent framework where the most critical agents were recognized and replicated. The *criticality* was determined by recognizing activities and roles of each agent in the multi-agent system. Similarly to overhearing, roles were

assigned to agents by feeding a recognizer with sequences of communication and social interaction between agents.

CHAPTER 3: TEAMWORK DATA ANALYSIS AND ACQUISITION

In this chapter we describe the datasets which were used to train and validate the teamwork activity recognizer presented in this thesis. In Section 3.1 we describe the general characteristics of teamwork patterns. In Section 3.2 the domain specific movement patterns of military teams are described. In Section 3.3 we introduce a knowledge engineering tool that was developed specifically to visualize, identify and extract training and validation datasets from the recorded datasets. The general data format that was used to represent the actions of embodied agent is described in Section 3.4. Finally, in Section 3.5 we present two datasets that were collected from two separate real-world military exercises. The first dataset was recorded in a large-scale warfare exercise consisting of hundreds of soldiers and tanks over a three-day period. The second dataset was generated at the Ft. Polk and Ft. Benning training centers in a small-scale urban exercise. Furthermore, we describe a simulation environment which can be used to generate artificial datasets from agents competing in a foraging game.

3.1 Teamwork Patterns

The teamwork activity recognition system described in this thesis annotates observations by recognizing team actions based on a library of patterns. In many application domains these patterns are *prescribed*. That is, external rules force the agents to act in predetermined ways. This is the

case for many sports such as American football, soccer or basketball, for dance groups, for the movement of firefighters in a building, and for the movement of rescue teams. Military tactics in various environments (battlefield, urban environment, naval or air operations) also described specific patterns. In this case, the knowledge engineering work of extracting teamwork patterns is made comparatively easy by the fact that the patterns are documented by training manuals, military doctrine or other similar documents.

In addition to prescribed patterns, many application domains show *emergent* patterns which are the result of the physical or psychological interaction between the participants. Examples are the lines which form when people move on narrow corridors, the specific formations emerging in crowded places with multiple interest points such as markets or museums or the flying patterns of cranes. In these cases, no prescriptive description of the movement patterns is available, so the knowledge engineer needs to work from the observations.

One difficulty in acquiring the teamwork patterns is that even when descriptions of the team patterns exist, their format is usually natural language and diagrams intended for human reading, which are not suitable for immediate application by a recognition engine. One approach is to translate the natural language description (and the associated diagrams) and encode them into the format of our recognizer (which is the approach taken in [SS06b, SS06a]). Alternatively, we can use the specifications to identify representative occurrences of the specific teamwork patterns in our observations, and use these sequences to train our recognizer, which is the approach taken in this thesis.

3.2 Military Movement Tactics

In the following, we succinctly summarize the teamwork patterns in our application domain, military movement tactics for battlefield and urban operations. Although the characteristics and features of units such as tanks, infantry carrier vehicles (ICV), mobile gun system (MGS) vehicles and infantry are under constant development, the fundamental movement tactics remain unchanged.

3.2.1 Movement Techniques

A military team uses different movement techniques based on the battlefield context. The team leader selects the appropriate tactics by estimating the possibility of enemy contact, the expected type of contact, the availability of an overwatch unit, the characteristics of the terrain, and the required speed and security. The main movement tactics are: 1) Traveling, 2) Traveling overwatch, and 3) Bounding overwatch (with either alternating or successive bounds).

Traveling

Traveling is used when contact or conflict with enemy is not expected and high movement speed is desired. It is characterized by concurrent movement of all units in the team.

Traveling Overwatch

Traveling overwatch is used when it is possible to engage in contact with enemy. It maintains a balance of speed and security. It is characterized by continuous movement of a lead unit and alternating advancements of trailing overwatch units. The trailing units may come to complete stops to provide suppressive fire if enemy is detected.

Bounding Overwatch

Bounding overwatch is used when enemy contact is likely. It favors security over speed and is characterized by an overwatch unit that is able to suppress enemy fire at all time. There are two types of bounding overwatch. In the *alternating bounds* approach, the bounding unit moves forward and is protected by an halted overwatch unit. When the bounding unit reaches the overwatch unit's protection range the overwatch unit becomes the bounding unit and the bounding unit becomes the overwatch unit. This cycle continues until the destination is reached. In the *successive bounds* model the overwatch unit and the bounding units retain their roles throughout the movement.

3.2.2 Movement Formations

Movement tactics are used in combination with movement formations to facilitate various tactical needs such as increased flank protection or heavy fire support. The formations differ in security,

control, speed and fire power direction. The formations are chosen based on type of mission, enemy positions, terrain, weather conditions, speed and flexibility.

Column Formation

Column formation is based on a leader followed by the rear units, and is used when fire power is needed in all directions (flanks, front and rear). The column formation is efficient for fast traveling while still allowing ease of formation change. For instance a team in column formation can easily form an echelon formation, hence reinforcing flank and front fire power if needed. The formation is easy controlled by a commander since rear units are following a leader unit. Figure 3.1 illustrates the column formation.

Figure 3.1: Column formation.

Line Formation

Line formation is used when heavy fire power is needed in the front. The downside of this formation is that fire power in the rear and flanks are weak. The formation is difficult to control because the team leader needs to synchronize the speed of the team members to maintain the line. In addition, transforming to other formations from the line formation is cumbersome. Figure 3.2 illustrates the line formation.

Figure 3.2: Line formation.

Wedge Formation

Wedge formation is used when fire power is needed in front and on the flanks. The wedge formation is relatively easy to control, and it can easily transform into a column or echelon formation. Figure 3.3 illustrates the wedge formation.

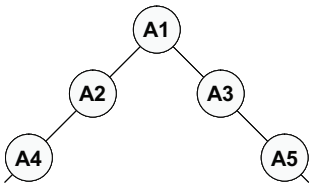


Figure 3.3: Wedge formation.

Vee Formation

Vee formation has two groups of units in the front, one unit group in the center and one unit group in the back. The vee formation provides good fire power in the front. The center and back units either perform overwatch or trail the front units. This formation is difficult to control and slow in speed. The main benefit of using this formation is that if attacked from any flank it is possible to counter attack with two groups and maneuver at least one group. Figure 3.4 illustrates the vee formation.

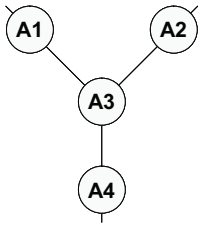


Figure 3.4: Vee formation.

Echelon Formation

Echelon formation is used if enemy contact is expected in the front or on one of the echeloned flanks. This formation is used when one of the flanks is secured by obstacles or other friendly units. The formation is difficult to control and therefore also slow. Figure 3.5 illustrates the echelon formation.

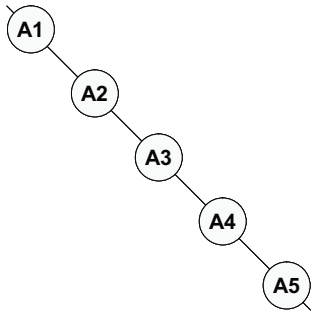


Figure 3.5: Echelon formation.

3.3 Teamwork Scenario Editor

Recording the teamwork of embodied agents results in relatively large data collections. Identifying and extracting representative examples from such large database can be a tedious and time consuming knowledge engineering task. For instance, one has to identify which team members contribute to the team behavior as well as where the representative example of interest starts and ends in time. In addition, the same teamwork behaviors can be performed by different team constellations further complicating the identification process. We have developed an interactive application for the editing and manipulation of recorded observations which simplifies the identification and extraction of representative examples used for training of the teamwork activity recognition system. The workflow of the application is inspired from video editing applications, but instead of a single stream of video information, it allows the editing of the observation streams coming from a large number of agents. As the geographical area covered by the observations can be very large, the applications allows us to zoom in to a specific area, select the agents of interest and replay specific

actions at various speeds or step by step. The application allow the tracking of the trajectories of the selected agents and export the movement traces. Using the application we can find and isolate the representative examples, which can be exported to the teamwork activity recognition system for direct input to the training algorithms that learn models of teamwork activity.

The teamwork scenario editor is used to simplify editing and visualization of data stored using the *Teamwork Data Format* (see Section 3.4). Figure 3.6 is a screenshot of the teamwork scenario editor visualizing tanks in a real-world warfare exercise dataset.

The application is able to perform bookmarking and sequence extraction to generate train and test datasets for the teamwork activity recognition system. The extracted sequences can also be warped or transformed by scaling, rotation and/or distortion operations. The transformed sequences can, for instance, be used to validate whether or not the teamwork activity recognition system is invariant to its features.

3.4 Teamwork Data Format

During the course of this work we evaluated our methods using datasets from several external data sources that were represented in different formats. To simplify the knowledge engineering effort we have converted these data sources into the *Teamwork Data Format* which is used in the *Teamwork Scenario Editor* for visualization and extraction of training and validation datasets.

The *Teamwork Data Format* was designed to encapsulate the following characteristics of actions performed by embodied agents from recorded teamwork activities:

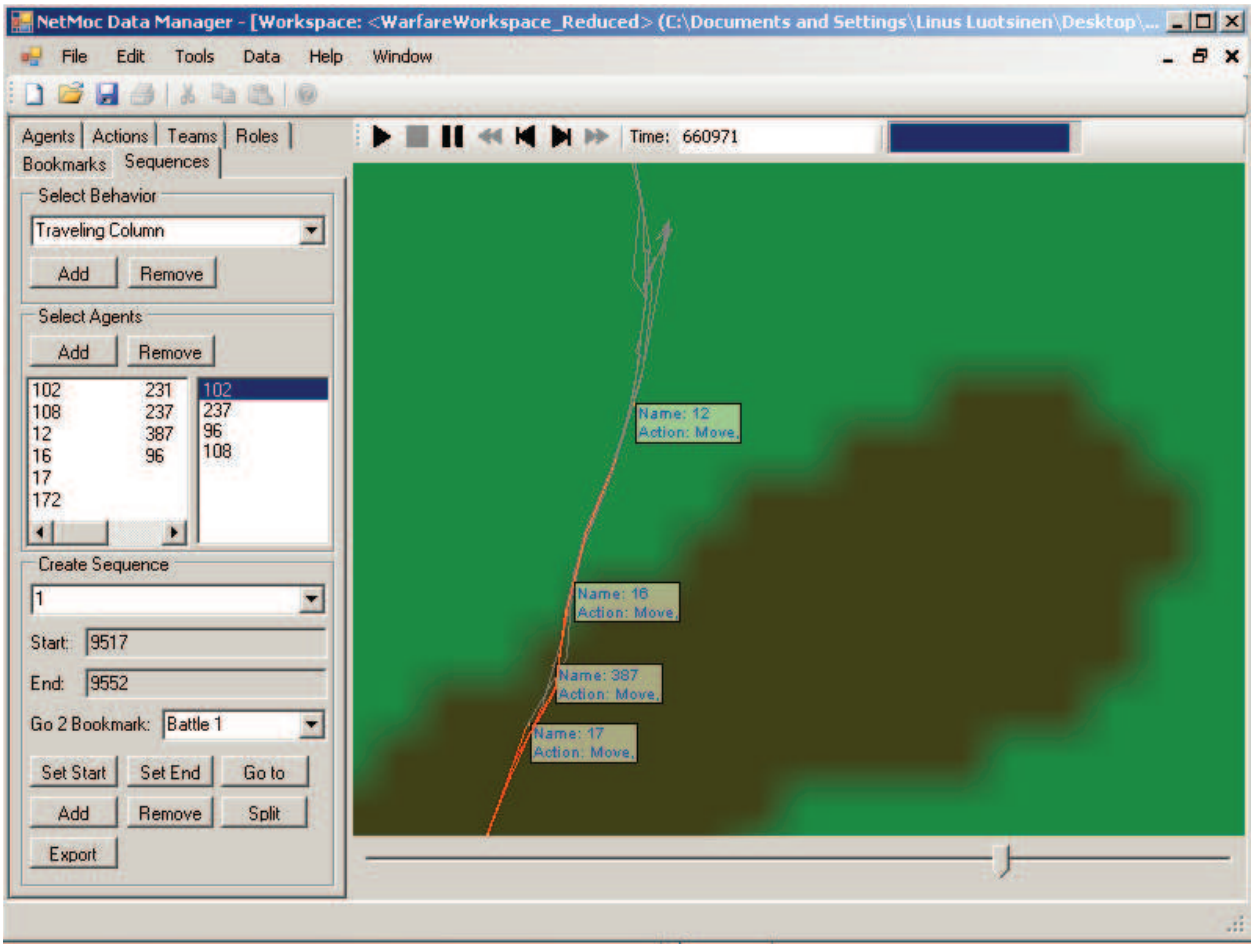


Figure 3.6: Screenshot of the teamwork scenario editor. The main area visualizes the selected geographical area, with the location and motion traces of the agents. The labels indicate short descriptions of the major parameters of the agents. The video editor-type controls are on the toolbar. The left hand panel allows the selection of agents, teams and actions, and contains controls for the marking and saving of selected observation subsets.

- *Location.* Each agent has a location in X, Y and Z[1].

- *Orientation.* The orientation of each agent is encoded by *Yaw*, *Pitch* and *Roll*.

- *Velocity and acceleration.*

- *Team memberships.*

- *Roles.*

In Figure 3.7 we present a relationship diagram that describes the tables and their relationships in the *Teamwork Data Format*.

3.5 Teamwork Datasets

Many of the applications in the teamwork recognition domain are based on activities from the real-world. Hence, it is important that the teamwork recognition system presented in this thesis is evaluated using real-world datasets. From a machine learning perspective real-world datasets are the most challenging datasets to learn, generalize and/or classify for the following reasons. First, real-world datasets have limited amount of examples available for training and validation. Second, the data is usually noisy due to precision errors in reporting devices, sensors and receivers.

[1]Datasets that use GPS coordinates can easily be converted to our format using UTM grid conversion software.

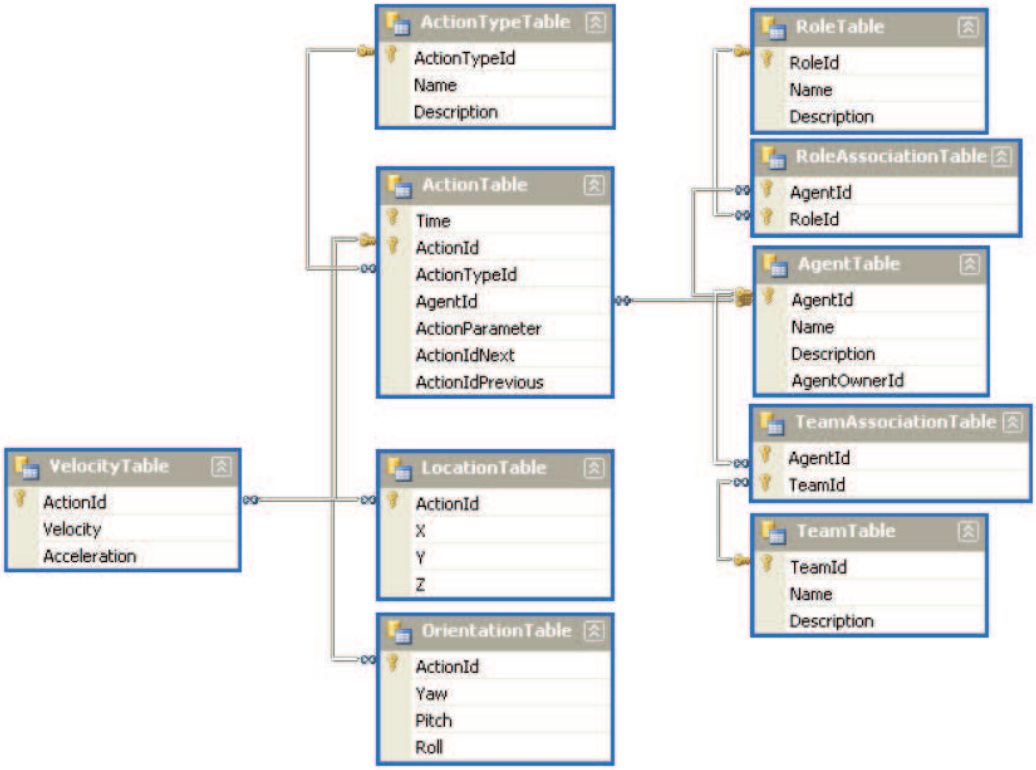


Figure 3.7: Relationship diagram of teamwork data format.

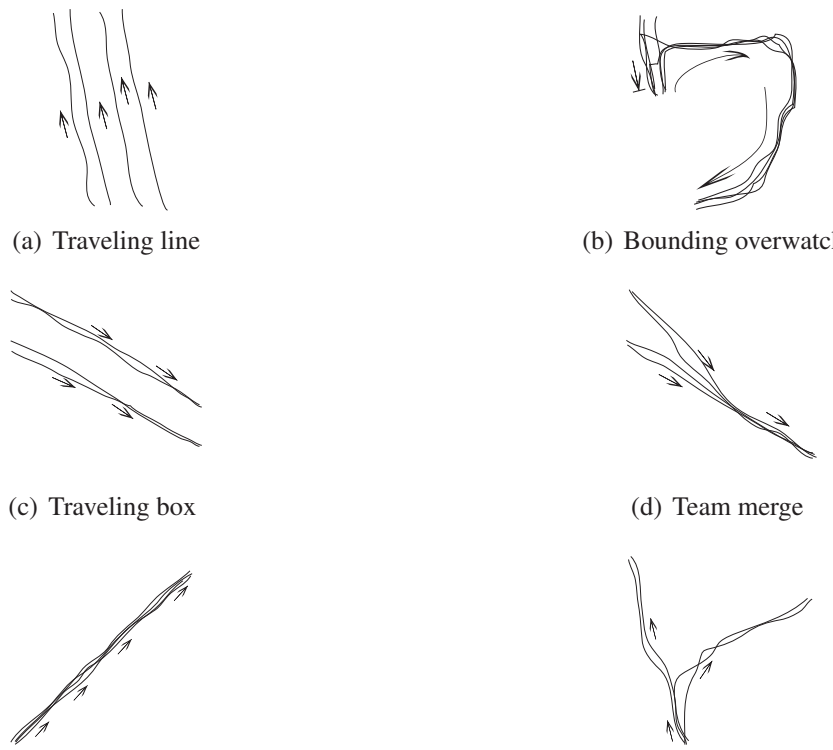
3.5.1 Warfare Exercise

This dataset was acquired from a real-world recording of a large-scale warfare exercise [FGE06]. The data, which was collected over several days, stem from multiple tanks and soldiers acting in a real-world environment. To extract the representative examples necessary to train the teamwork recognition system, we had chosen data from eight tanks forming two competitive platoons. The exercises are set up such that the platoons will make unanticipated contact. The tanks are equipped with global positioning system (GPS) devices, laser range finders (LRF) and laser range detectors

(LRD). The tanks simulate guided missile and large-caliber gun fire attacks using laser technology. The inflicted damage of an attack is determined using prior knowledge about the type of rounds fired as well as the impact point on the target tank. The LRFs are used to lock in on target tanks, while the LRD unit detects the origin of an incoming LRF.

All events during the warfare exercise were recorded in a database. Five event types were used to describe the actions that were performed by the actors in the exercise: The *state* event, which is used to report position, orientation, velocity and status of the actor was issued on a timeout basis for each actor in the exercise; The *fire* event was issued when one actor attacked another actor. The type of firearm and ammunition is recorded in the event; The *hit* event described the impact the attack on the target actor; The *detonation* event was used to describe the location and outcome of a detonation; Finally, the *LRF* event was used to store information about LRF usage in the exercise.

Using the *Teamwork Scenario Editor*, we extracted representative examples for the following teamwork behaviors: 1) Column formation traveling; 2) Line formation traveling; 3) Box (combined line and column) traveling; 4) Team split; 5) Team merge; and 6) Bounding overwatch. The details of these activities were described in Section 3.2. In this dataset, the bounding overwatch movement was executed as follows. After all four tanks come to a stop, the team splits so that two tanks advance while the other two tanks keep their position. When advancing tanks stop, the previously halted tanks start their advance. Sample movement patterns extracted from the dataset are shown as trace plots in Figure 3.8.



(a) Traveling line

(b) Bounding overwatch

(c) Traveling box

(d) Team merge

(e) Traveling column

(f) Team split

Figure 3.8: Movement pattern traces. Arrows illustrate direction of the team.

Figure 3.9 show a snap-shot, using a dataset visualizer, of the warfare exercise. The image show four tank models operating in wedge formation in the environment. Tank models are superimposed on top of aerial photographs.

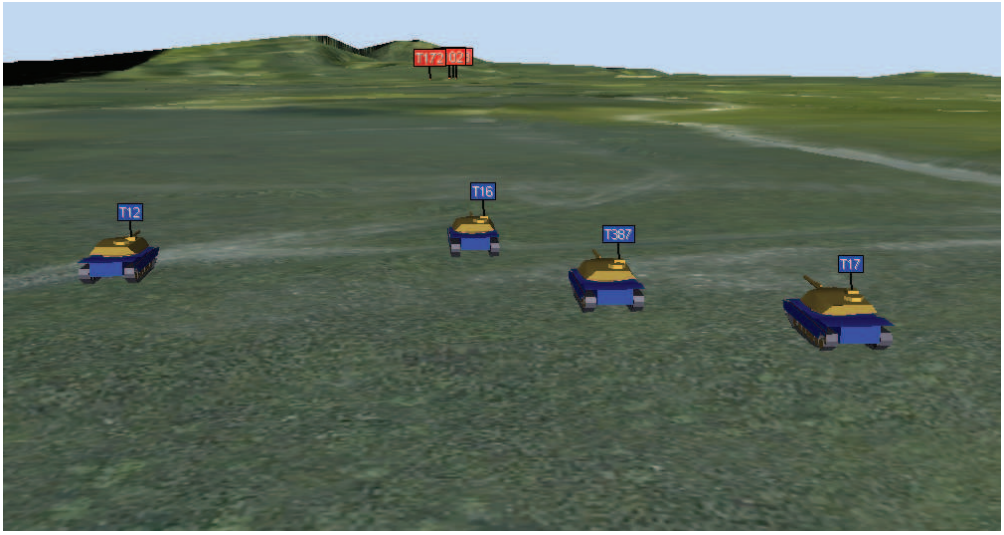


Figure 3.9: Screenshot visualizing a platoon consisting of four tanks in the real-world warfare exercise dataset. The billboards on the horizon mark the locations of the tanks of the enemy platoon.

3.5.2 Military Operations in Urban Terrain Exercise

This dataset was collected from exercises in the Military Operations in Urban Terrain (MOUT) domain. The recordings were generated at the Ft. Polk and Ft. Benning MOUT training centers. In addition to the real-world soldiers a set of artificially generated soldiers were added to the dataset using the OneSAF Testbed Baseline Semi-Automated Forces (OTBSAF) software which

was designed for training, experimentation and analysis of military exercises with interacting real-world and artificial soldiers.

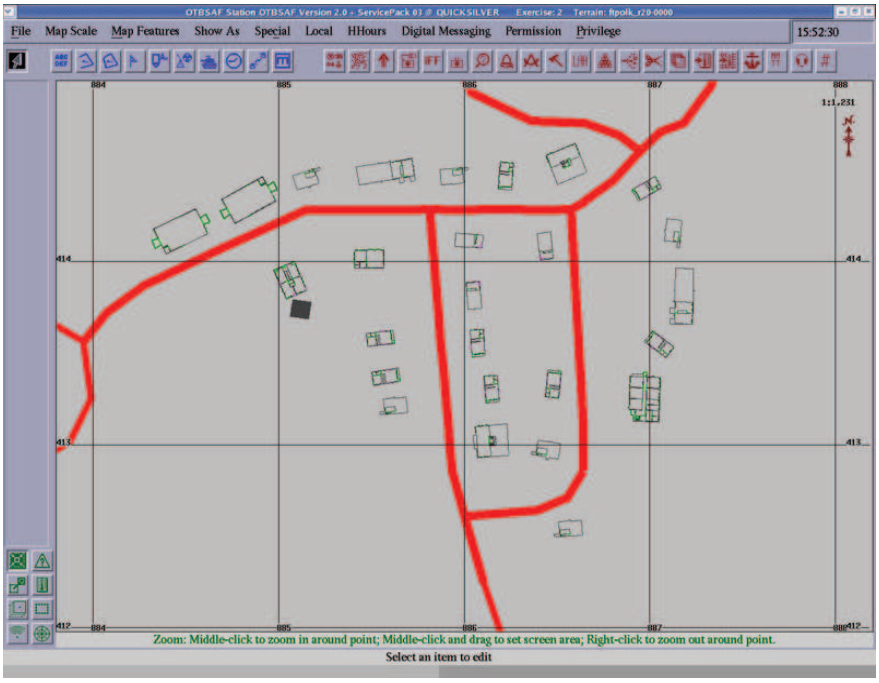
The MOUT exercise in this recording consisted of a hostage scenario where a group of terrorists have captured a group of civilians. The terrorists are hiding in an unknown location in the city. The objective for the soldiers in this exercise is to locate the terrorists, approach, systematically eliminate the threats and finally free the hostage. The exercise was repeated multiple times in clear daylight conditions, when it is raining and at night-time.

Figure 3.10 shows the OTBSAF software loaded with maps of the Ft. Polk and the Ft. Benning training centers.

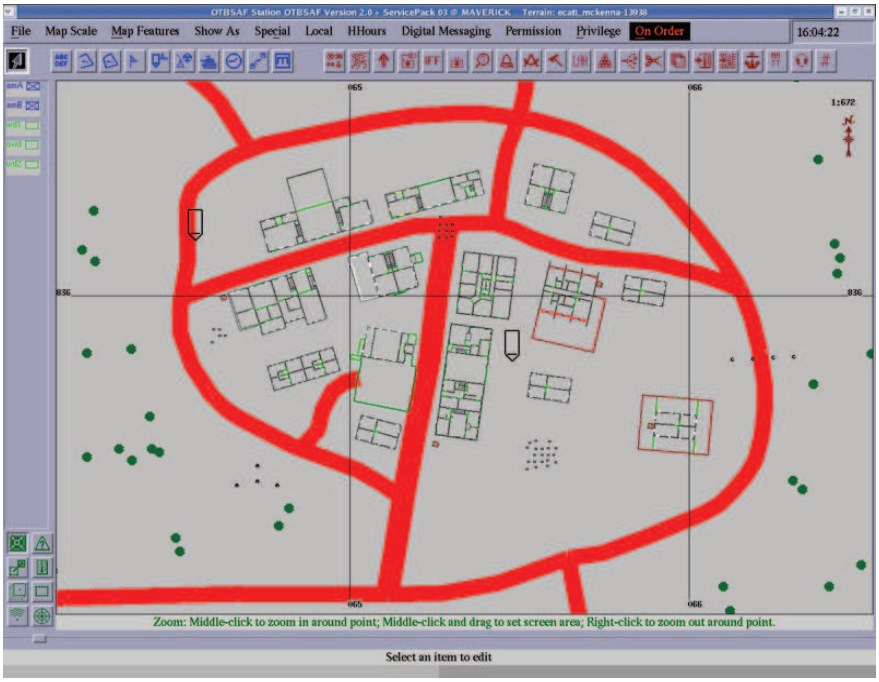
3.5.3 The Feed-Fight-Multiply Game

Artificial datasets can be automatically generated using a simple gaming environment named Feed-Fight-Multiply (FFM). FFM reflects many of the challenges encountered by agents in the real world and provide multiple paths to success which are desirable features of an artificial environment. The FFM environment is implemented in the Java based YAES simulation environment [BT05].

Twelve autonomous agents, implemented in twelve different paradigms of agency, have been evaluated in [BLE07]. Seven out of the twelve paradigm implementations employed teamwork strategies. In choosing the paradigms the authors strived to maintain a balance between high level, logic based approaches and low level, physics oriented models; between imperative programming,



(a) Ft. Polk.



(b) Ft. Benning.

Figure 3.10: Screenshot of OTBSAF showing the Ft. Polk and the Ft. Benning training sites.

Table 3.1: Concise description of the twelve implemented agents

Name	Paradigm	Paradigm cover-age	Team-work	Offline learn-ing	Realtime adapt.
AffectiveAgent	Affective model, anthropomorphic lifecycle	Limited	No	No	Yes
GenProgAgent	Genetic programming	Full	Yes	Yes	No
Reinforcer	Reinforcement learning	Full	Yes	Yes	No
CBRAgent	Case based reasoning	Full	No	Yes	Yes
RuleBasedAgent	Forward reasoning	Full	Yes	No	No
NaïveAgent	Naïve programming (scripting)	Full	Yes	No	No
GamerAgent	Game theory	Limited	Yes	No	No
CrowdAgent	Particle based crowd modeling	Limited	Yes	No	No
NeuralLearner	Backpropagation neural networks	Full	No	Yes	No
SPFAgent	Social potential fields	Limited	Yes	No	No
CxBRAgent	Context based reasoning	Full	No	No	No
KillerAgent	Minimal heuristic	Full	No	No	No

declarative approaches and "learning from basics"; between anthropomorphic or biologically inspired models on one hand and pragmatic, performance oriented approaches on the other. The implemented agents are concisely described in Table 3.1.

The Feed-Fight-Multiply game borrows elements from turn-based multi-player strategy games and artificial life. Agents are moving in a two-dimensional environment having accessible zones and obstacles. The agents can sense the environment within the range of their sensors. Food resources appear at random points in the environment; consuming food increases the energy level of the agents. Agents can attack each other, by destroying an agent, the attacker gains access to its resources. Finally, agents can multiply by (non-sexual) reproduction. Figure 3.11 summarizes the possible actions of the agents in the FFM game.

Instead of keeping a single score, FFM records multiple parameters of the agent behavior. This means that not only there are multiple paths to success, but the final goals of the agents could be different as well. Of course, all agents are required to work towards their survival, but besides that, the criteria for success could be maximum amount of resources gathered, survival on minimum amount of resources, largest number of agents killed, number of individual agents of the same type at the end of the game, or others. Figure 3.12 shows a typical FFM game in progress.

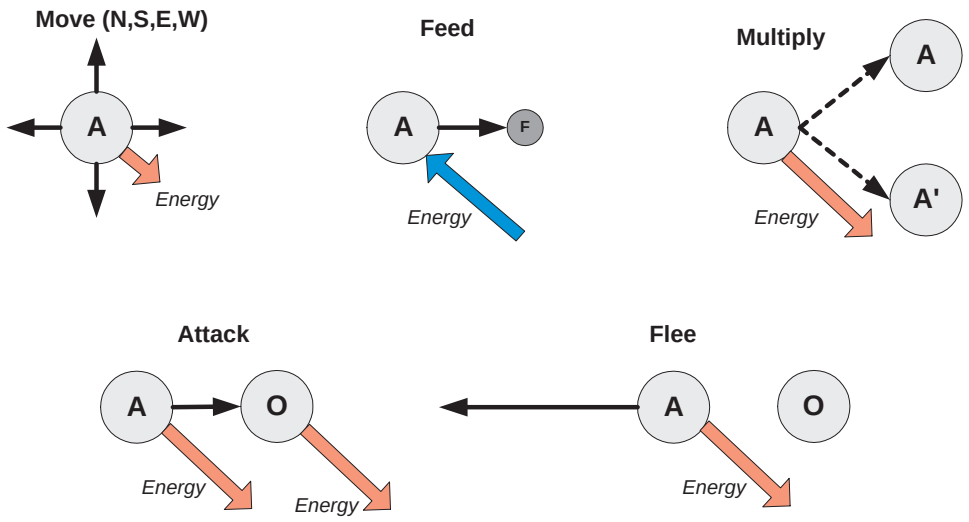


Figure 3.11: The possible actions of the agents in the Feed-Fight-Multiply game.

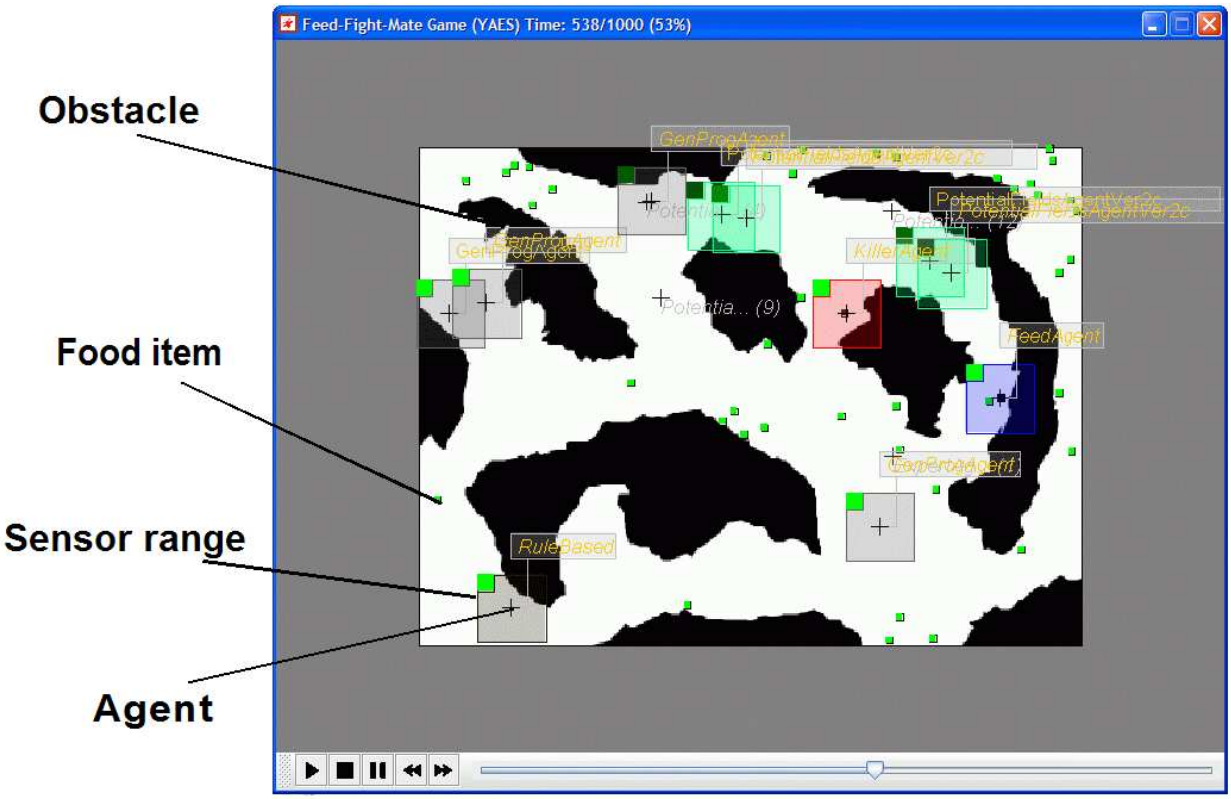


Figure 3.12: Screen-shot of the Feed-Fight-Multiply environment.

CHAPTER 4: TEAMWORK ACTIVITY RECOGNITION

In this chapter we describe an approach to recognize team actions from the observations of the movement of agents in the field. The basic model presented in this chapter will be extended in Chapter 5 with team-oriented feature extraction and in Chapter 6 with a role recognition module.

Our starting point is a recording of the geometrical positions and orientations of the agents in a scene containing obstacles and various types of terrain. Such recordings are regularly done in military exercises where all the participants (soldiers, MGSs, ICVs) carry GPS units. For sport teams, this information can be hand-extracted from videos [IB01]. For many other domains, computer vision techniques can be deployed to extract such scene information.

Our approach is based on recognizing teamwork patterns based on the library of movements, encoded as Hidden Markov Models. The HMM is a representation of the *idealized team action* (ITA), which can be either knowledge engineered (hand crafted) or acquired through learning. Our work builds upon previous work by Sukthankar and Sycara [SS06b, SS06a], and extends it with the ability to dynamically acquire new teamwork patterns based on representative examples. We use a dataset representing a real-world terrestrial warfare exercise (see Section 3.5.1), and consider teamwork patterns commonly specified in military doctrines as described in Section 3.2. The approach can be naturally extended to other domains where teamwork is expressed and can be recognized from the coordinated movement patterns of the embodied agents.

4.1 Pre-processing Observations

An observation sequence of length T is denoted by $\mathbf{V}^T$ and consists of observation vectors, or feature vectors, indexed by time, t, as:

$$\mathbf{V}^T = \{\mathbf{v}_1, \ldots \mathbf{v}_t \ldots, \mathbf{v}_T\}$$

where $\mathbf{v}_t$ represents the visible observation $\mathbf{v}$ at time step t. The features selected for the observation vector $\mathbf{v}_t$ are dependent on the application domain.

In our domain of embodied agents, the sources of the observation data are the GPS readings of the individual agents (*latitude*, *longitude*), which we transform into x and y coordinates according to the UTM (Universal Transverse Mercator) grid. Thus, given a team of four agents at time t we observe 8 features: $\mathbf{v}_t = \{x_{1,t}, y_{1,t}, x_{2,t}, y_{2,t}, x_{3,t}, y_{3,t}, x_{4,t}, y_{4,t}\}$, where x_1 and y_1 depict the x and y coordinate of the first team member. To improve the ability of our recognition system to generalize over observations taken at different locations, scales and orientations, we start our recognition process by normalizing the observations. The normalization process starts by translating the observation sequence such that its centroid, $\mathbf{C}$, coincides with the origin.

$$\mathbf{C} = \begin{bmatrix} c_x \\ \\ c_y \end{bmatrix} = \begin{bmatrix} \frac{\sum_{t=1}^{T} \sum_{n=1}^{N} x_{n,t}}{T \cdot N} \\ \\ \frac{\sum_{t=1}^{T} \sum_{n=1}^{N} y_{n,t}}{T \cdot N} \end{bmatrix}$$

where N is the number of agents in the team.

In the second step we align the observation sequence to an axis by rotating with the value of the observation sequence orientation θ. θ is the angle between the last ($t = T$) observation vector and the first ($t = 1$) observation vector, and it can be calculated with the following formula:

$$\theta = arctan\left(\frac{\sum_{n=1}^{N} y_{n,T} - y_{n,1}}{\sum_{n=1}^{N} x_{n,T} - x_{n,1}}\right) \tag{4.1}$$

The rotation of the observation sequence is performed according to the rotation matrix:

$$\mathbf{R} = \begin{bmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{bmatrix}$$

Finally, we need to consider the problem of recognizing a observation sequence even if the observations are scaled (with a limited range of scaling factors). This is achieved by including team behavior observations at different scales in the behavior modeling (training) process. The scales are carefully chosen based on the application domain so that up-scaling and down-scaling does not disturb team behavior semantics.

4.2 Modeling Idealized Team Actions with Hidden Markov Models

We have used Hidden Markov Models (HMM) to model the spatial and temporal relationships of a specific teamwork pattern, such as traveling overwatch or bounding overwatch. A HMM consists of n hidden states, where the hidden states are denoted by $\boldsymbol{\omega} = \{\omega_1, \ldots, \omega_n\}$. Let us denote with

$P(\omega_j(t+1) \mid \omega_i(t))$ the transition probability, that is, the probability that the state of the HMM at time $t+1$ will be ω_j if at time t it was ω_i. The parameters determining the HMM are:

1. Transition probabilities,

   $\mathbf{A} = \{\alpha_{ij}\}$, where $\alpha_{ij} = P(\omega_j(t+1) \mid \omega_i(t))$

2. Emission probabilities,

   $\mathbf{B} = \{\beta_j(\mathbf{v})\}$, where $\beta_j(\mathbf{v}) = P(\mathbf{v} \mid \omega_j(t))$

3. Initial probabilities,

   $\boldsymbol{\pi} = \{\pi_i\}$, where $\pi_i = P(\omega_i(t))$

Thus, we can model a team behavior as $\boldsymbol{\theta} = \{\mathbf{A}, \mathbf{B}, \boldsymbol{\pi}\}$. One way of relating $\mathbf{A}$ and $\mathbf{B}$ to a team behavior is to think of it as a model of the team's mental and physical state respectively [HV99]. Figure 4.1 illustrates a HMM with three hidden states, each with a continuous probability density function (PDF) representing emission probabilities ($\beta_j(\mathbf{v})$). The indexes i and j denote the hidden states and $\mathbf{v}$ denotes the observed visible state. We model the emission probabilities using the general multi-variate normal density:

$$\beta_j(\mathbf{v}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} exp\left\{-\frac{1}{2}(\mathbf{v}-\boldsymbol{\mu})^t\boldsymbol{\Sigma}^{-1}(\mathbf{v}-\boldsymbol{\mu})\right\} \tag{4.2}$$

where $\boldsymbol{\mu}$ is the mean vector, $\boldsymbol{\Sigma}$ is the covariance matrix and d is the dimensionality of $\mathbf{v}$. $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are estimated using maximum-likelihood methods on training data.

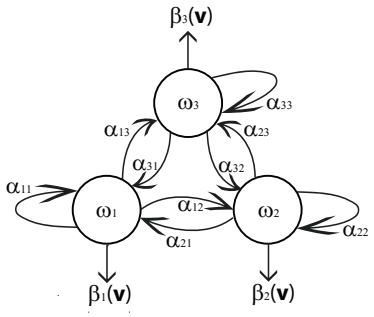


Figure 4.1: A HMM model with three hidden states, α_{ij} transition probabilities and $\beta_j(\mathbf{v})$ emission probabilities.

The model of a team behavior $\boldsymbol{\theta} = \{\mathbf{A}, \mathbf{B}, \boldsymbol{\pi}\}$ can be created through knowledge engineering, translation from other representations or learning.

Knowledge engineering is the manual creation of the model based on interviews with experts, natural language descriptions or direct observation of recordings. This is the approach taken in [SS06b, SS06a]. Unfortunately, the hidden nodes of the HMM do not necessarily correspond to states for which we have a good intuition, making the manual assigning of probabilities a difficult task. Therefore, the knowledge engineering of the models is a difficult, time consuming process, requiring significant expertise and several trial and error iterations. At every iteration we need to verify the models' recognition ability against representative test samples.

An alternative approach is to generate the model from existing representations. For instance, if the problem domain contains only autonomous robots with known programs, it is possible to derive the HMM model from the control programs of the robots. Unfortunately, this method is not feasible for human participants or embodied agents for which the control program is not readily available.

In general, the available descriptions of team actions, such as military doctrine or the diagrams of sport team coaches, typically require significant background and common sense knowledge, and thus, their automatic translation to HMM is difficult.

Finally, the third approach, taken by the work presented here, is to learn models from a set of representative examples of teamwork actions. This requires us to first manually annotate the occurrence of specific team actions in our observations[1]. We've employed two separate learning algorithms: Baum-Welch [BPS70]; and Segmental K-Means [JR90]. The two algorithms differ in the choice of optimization criterion employed. Baum-Welch optimizes $\boldsymbol{\theta}$ such that $P(\mathbf{V}^T \mid \boldsymbol{\theta})$, that is the probability that a $\mathbf{V}^T$ was generated by $\boldsymbol{\theta}$, is maximized. Segmental K-Means optimizes $\boldsymbol{\theta}$ such that $P(\mathbf{V}^T, \mathbf{I} \mid \boldsymbol{\theta})$ is maximized, where $\mathbf{V}^T$ is a sequence of observations and $\mathbf{I}$ is the optimum hidden sequence as given by the Viterbi decoding algorithm [For73]. In Appendix A we describe the learning algorithms in detail.

Figure 4.2 shows an overview of the full workflow of teamwork annotation (we will show an extended version of this workflow in Chapter 6). An external database of observations is imported into the teamwork scenario editor where visualization and identification of teamwork activities result in a set of representative examples which will be used as training and validation data. The training and validation data is exported to the teamwork annotation framework where teamwork behavior models are generated through the automatic learning process described in the previous section. The library of teamwork behavior models is then used by the classifier to annotate a real time data stream of observations.

[1]This step is necessary even in the case of knowledge engineered models for verification purposes.

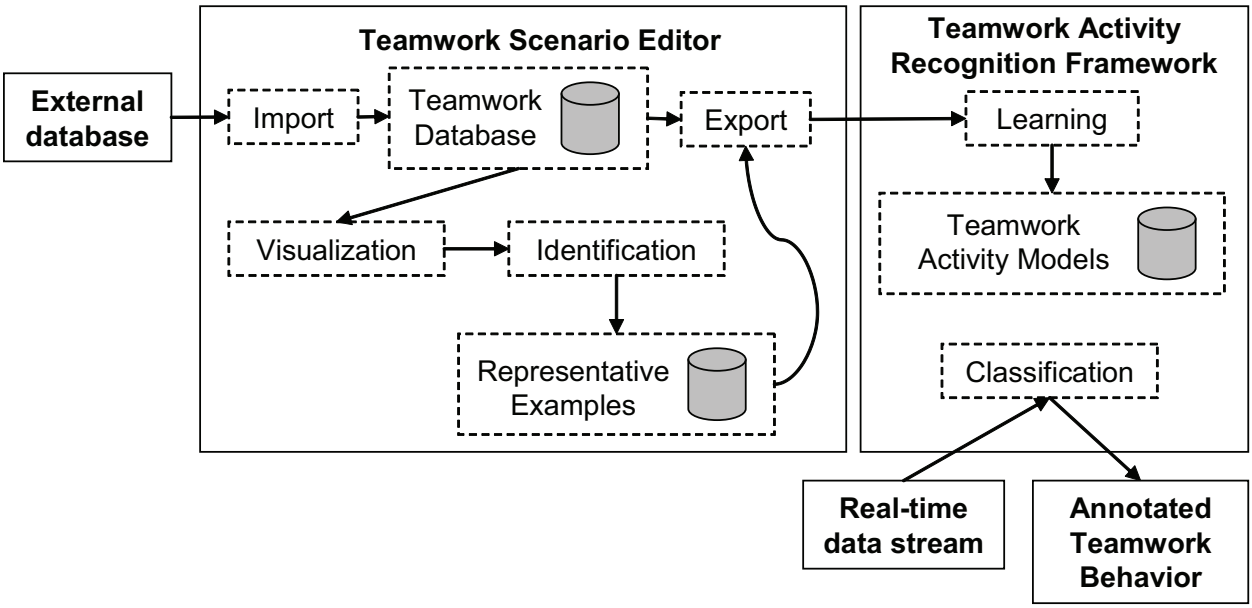


Figure 4.2: An overview of the teamwork annotation workflow. The external observations are edited in the teamwork editor application, resulting in a set of representative examples for training and validation data. The learning process in the teamwork annotation framework uses these examples to generate a library of behavior models. This library is used by the classifier to annotate a real time data stream of observations.

4.3 Behavior Classification

The library of the team behaviors is used to continuously check whether the currently observed behavior of the agents matches one of the patterns. This process, called behavior classification, annotates the observations with the recognized team action. Given a behavior model $\boldsymbol{\theta}$ we calculate $P(\mathbf{V}^T \mid \boldsymbol{\theta})$ recursively using the forward evaluation algorithm.

$$P(\mathbf{V}^T \mid \boldsymbol{\theta}) = \sum_{i=1}^{n} \rho_T(i) \tag{4.3}$$

$$\rho_{t+1}(j) = \begin{cases} \pi_j \beta_j(\mathbf{v}_{t+1}) & \text{if } t = 0 \\ \beta_j(\mathbf{v}_{t+1}) \sum_{i=1}^{n} \rho_t(i) \alpha_{ij} & \text{otherwise} \end{cases}$$

Figure 4.3 illustrates a trellis unfolding [DHS00] of the HMM from Figure 4.1. In the trellis, vertices represent $\rho_{t+1}(j)$ and directed edges show how the summation should be performed if $t \neq 0$. The probability values of the trellis vertices at $t = T$ are accumulated to form the final evaluation probability. The output of the classification is the behavior model with the maximum evaluation probability.

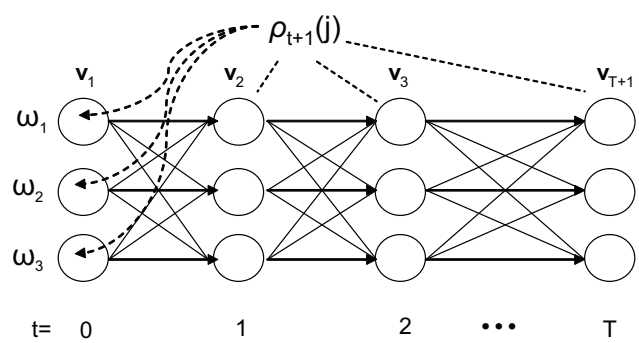


Figure 4.3: HMM trellis unfolding.

4.4 Dataset Acquisition

One of the problems faced by all efforts similar to ours is the scarcity of data which can be used to train and test our observations. Our group was fortunate to have a large, real world dataset covering military operations over three days and an area of over 100 km^2 (see Section 3.5.1). Nevertheless, even with a dataset of this size, there are a relatively small number of examples of each teamwork

action[2]. For a better testing of our algorithms, we used as testing data not only unmodified real world recordings, but we created additional, semi-artificial data generated from the real world data through a series of distortion operations. The elementary operations performed on the real world data $\mathbf{V}^T$ were rotation, scaling and addition of noise. We generate 20 test sequences from every real world recording by applying these operators with random parameters. Table 4.1 summarizes the number of observation sequences and observation vectors for the train and validation data.

Table 4.1: Warfare dataset summary depicting number of observation sequences that were used during training and testing. Also, the total number of observation vectors within the sequences are shown for train and test data.

Behavior	Train seq.	Test seq.	Train obs.	Test obs.
Bounding overwatch	6	40	651	4340
Team split	15	100	453	3020
Traveling line	9	60	345	2300
Traveling box	9	60	273	1820
Team merge	9	60	282	1880
Traveling column	12	80	342	2280

4.5 Teamwork Activity Recognition Accuracy

The accuracy of the teamwork activity recognition is calculated using the cross-validation procedure. The accuracy is represented by the fraction of successful classifications (in a range from 0 to 1).

[2]Note that in every military operation, long periods of relative inactivity (for instance, traveling without enemy contact) are alternating with bursts of action.

Table 4.2 shows the accuracy numbers in the function of the number of hidden states for the two learning algorithms. The same number of hidden states were used for all the samples. The best overall result was obtained by the segmental K-Means algorithm using 4 hidden states, yielding a 0.8244 rate of successful classifications. The Baum-Welch algorithm peaks when 3 hidden states are used for each HMM, with a performance number of 0.7865. The average training time, using a 1.8GHz processor, was 5.1158s for the Baum-Welch algorithm and 0.5596s for the Segmental K-Means. Although we find the K-Means algorithm to be approximately 9 times faster than the Baum-Welch algorithm, the time necessary to prepare the representative training examples is significantly longer than these numbers. Thus, in a practical deployment, a good choice would be to train patterns using both algorithms and choose the ones giving the higher recognition accuracy.

Another statistics of interest is the classification accuracy for every type of teamwork behavior Table 4.3. The teamwork behavior library used was the one generated by the Segmental K-Means learning algorithm with 4 hidden states. We notice that the recognition accuracy is not distributed evenly across the behaviors. The behavior which was recognized most consistently was the traveling column, with a 1.0 recognition rate. Interestingly, the relatively simple behavior of traveling line fared the worst, being recognized only 31 times out of the 60 test sequences, yielding a match rate of 0.52.

We are interested not only in the number of behaviors which were classified incorrectly, but also which behaviors were the misclassified behaviors mistaken for. The confusion matrix in Table 4.4 contains this information, with an extra column added for cases where the annotation frame-

work was unable to match an observation sequence with a behavior model. For instance, we find that errors in the classification of the traveling line behavior model occurred by the behavior being confused with the team split (79%) and traveling box (17%) behaviors. This is not a surprising result since the trace appearance of the traveling box behavior is similar to the traveling line behavior. Also, team split can occur in many ways. One particularly interesting observation sequence, that is included in our training dataset, is when the team splits into two team in the traveling line mode. The similarity of this training example might have led to the high confusion rate in this case.

Table 4.2: The ratio of correctly classified samples for HMM patterns with the number of hidden states ranging from 1 to 5, and trained in using the Baum-Welch and the Segmental K-Means learning algorithms.

Hidden states	Baum-Welch	K-Means
1	0.6787	0.6787
2	0.7294	0.7419
3	0.7865	0.7817
4	0.7588	0.8244
5	0.7188	0.766

Table 4.3: Classification accuracy over all teamwork behaviors. The table shows correct classification rate and misclassification rate.

Behavior	Match rate	Error rate
Bounding overwatch	0.75	0.25
Team split	0.98	0.02
Traveling line	0.52	0.48
Traveling box	0.85	0.15
Team merge	0.85	0.15
Traveling column	1.0	0.0

Table 4.4: Error distribution matrix showing misclassification rates for all behaviors.

Misclassified as →	a	b	c	d	e	f	Unknown
Bounding overwatch (a)	-	10%	0%	0%	0%	0%	90%
Team split (b)	0%	-	0%	100%	0%	0%	0%
Traveling line (c)	0%	79%	-	17%	0%	0%	4%
Traveling box (d)	0%	56%	33%	-	11%	0%	0%
Team merge (e)	0%	100%	0%	0%	-	0%	0%
Traveling column (f)	0%	0%	0%	0%	0%	-	0%

Overall we find the performance results promising. Using very simple feature vectors consisting of basic coordinate (x, y) pairs the framework was able to discriminate 6 types of movement techniques with 82% accuracy.

4.6 Classification Speed

Let us now investigate the speed at which the annotation of the observations can happen. One threshold, extremely important for many practical applications is whether the speed is high enough to allow the processing of a real time data stream. Table 4.5 shows the mean CPU time elapsed during the validation process, using a 1.8GHz processor. For instance, for a classifier using HMMs with 4 hidden states the average CPU time required to make a decision was 9.39ms. The box plots in Figure 4.4, depict the median, minimum, maximum, lower quartile and upper quartile of the CPU time used by the behavior classification algorithm during validation. All graphs show CPU time for models with hidden states ranging [1,5].

The results show that there is a trade-off between classification accuracy and real-time constraints. As the complexity of HMMs increase, the classification algorithm use more CPU time.

Let us now discuss the problem of real time recognition. Naturally, in the case of our validation data, the recognition happens on the order of tens of milliseconds, while the real life maneuvers depicted happen on the scale of tens of seconds. Real time observation annotation is therefore possible if the observations are strictly restricted on the team members, and the team in the observations is of the same structure as the one used in the training of the teamwork patterns. This condition might hold in the observation of a military exercise, but it does not apply to the observation of a real battlefield. If the team action needs to be recognized from an observation stream containing hundreds or thousands of agents, a large number of hypothesis needs to be tested concurrently, which can erode the speed advantage of the classifier.

Table 4.5: Mean CPU time used by the behavior classification algorithm during validation.

Hidden states	Mean time (ms)
1	2.543
2	4.743
3	7.02
4	9.394
5	12.06

4.7 Conclusions

In this chapter we described an approach for detecting and annotating teamwork behavior in observations of embodied agents. The approach relies on a library of behavior patterns encoded as

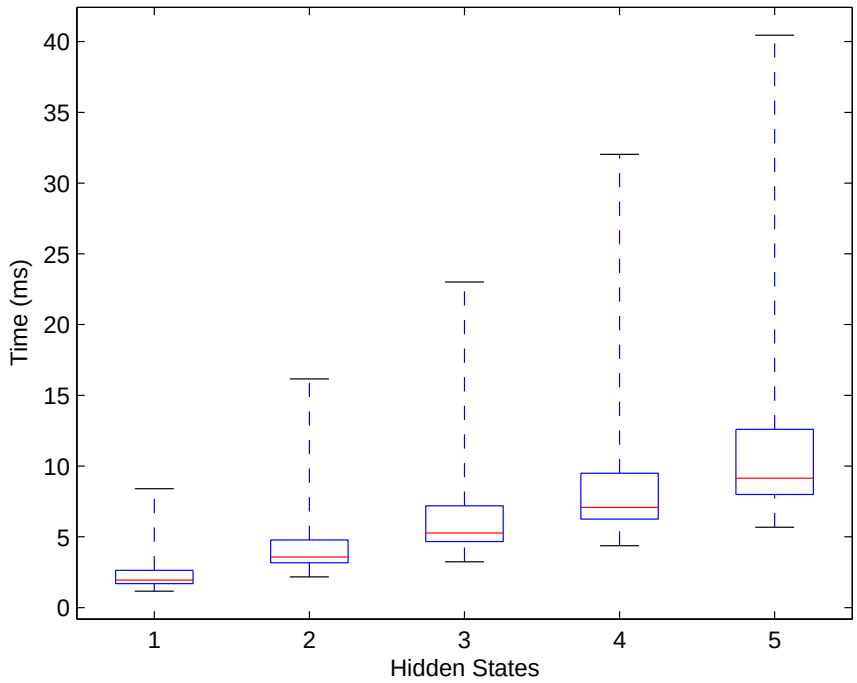


Figure 4.4: Box plots representing the median, minimum, maximum, lower quartile and upper quartile of CPU time used by the behavior classification algorithm.

Hidden Markov Models. The main contribution of this study compared to previous approaches is that we use an approach through which the HMMs are learned from a small number of representative examples extracted from observation data. We deployed two learning algorithms: the Baum-Welch and the Segmental K-Means algorithms. On our dataset we found that both algorithms yielded approximately equivalent recognition accuracy, but the K-Means algorithm used significantly lower CPU time for learning. We tested our approach on a dataset representing a real-world military exercise. Representative examples were selected for the creation of six HMMs corresponding to specific team behaviors of tank platoons documented in the military doctrine. We

find that the framework provides a recognition accuracy of approximately 82% on our dataset and a recognition speed a magnitude faster than the time taken by the tanks to complete the maneuvers.

4.8 Lessons Learned

The accuracy and performance numbers obtained refer to the somewhat idealized world of a military exercise, where embodied agents could be isolated from the other participating agents and embedded GPS devices allowed the acquisition of high quality data. In many practical applications, the annotation of teamwork needs to be performed in the conditions of higher environmental noise, less precise data and the presence of many additional agents. In the following chapters we address and focus on improving the following shortcomings:

- In the recognition system presented above, sequences of team positions are normalized to be rotation and translation invariant. The sequences are, however, not scale invariant. Hence, poor recognition performance is expected when teamwork actions are executed in scales that are different from the ones used in the training process. Hence, there is a need to identify and develop team-oriented feature functions that are invariant to irrelevant transformations of teamwork data. (Chapter 5)

- The environment plays an important role when teams are selecting strategies and activities. For instance, a team of tanks in a warfare situation would naturally adapt its movement patterns according to the terrain and/or nearby obstacles. Additionally, in team sports such

as Soccer the environment consists of a field with object such as goals and offside lines that clearly influence activity selection in a team. If environmental cues are available then methods to incorporate them in the recognition system need to be identified. (Chapter 5)

- As more and more feature functions are used in teamwork activity recognition systems it is important to reduce the degrading effects of the "curse of dimensionality". One approach is to reduce the hypothesis space of high dimensional feature vectors through a symbolic discretization process. That is, instead of using real-valued features, which are infinite, this approach move towards a finite and much smaller discrete hypothesis space. (Chapter 5)

- In all experiments up to this point we have evaluated teamwork activity recognition where all activity models have equal number of hidden states. This approach works well in cases where there are only small deviations in complexity between the teamwork activities. However, in practical applications the recognition system have to adapt the complexity, i.e. number of hidden states, of the HMM to the complexity of the activity. (Chapter 6)

- In practical applications of teamwork recognition it is important to recognize the roles that are played by the agents in the team. For instance, in the warfare patterns presented in Section 3.2 it is desirable to identify the leader and the follower. Additionally, from a HMM learning and classification standpoint, the order in which agent features, e.g. x and y coordinates, are presented to the feature vector are fixed. As a result, to facilitate recognition of activities where the order is unknown a mapping between individual team members and po-

sitions in the feature vector need to be identified as a pre-processing step in the recognition system. (Chapter 6)

- In the recognition system presented above the multi-variate Gaussian PDF was employed to model emission probabilities. However, emission probabilities of HMMs can be modeled using alternative and possibly more accurate probability density functions. (Chapter 7)

CHAPTER 5: TEAM-ORIENTED FEATURE EXTRACTION

It is common practice in pattern recognition to reduce the dimensionality and/or discretize the input data before the recognition process. As there is naturally a loss of information in this process, it needs to be performed keeping in mind the ultimate goals of the recognition process. In our case, the teamwork activities are the data which needs to remain in the dataset, while the agent specific information can be filtered out.

In the team action recognition workflow, the input of the recognition activity is the geometrical location of the agents in two dimensional coordinates, with, potentially, the addition of values such as orientation, or gaze. Basic preprocessing, such as alignment on the centroid does not change the nature of this data. As all agents have the same set of coordinates, the dimensionality of the input increases linearly with the number of agents in the scene. Although for small teams of 4-5 agents the resulting input space is still marginally acceptable for direct input to the HMM, the training process is facing difficulties in extracting relevant information. One problem is that important structural information is lost in the form of the data presented to the HMM. For instance, if we represent the input space with an 8 element vector $\{x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4\}$ (used in Chapter 4) the HMM will not know that x_1 and y_1 are referring to the same agent or that y_1, y_2, y_3 and y_4 are referring to the same direction.

In this chapter we describe a process through which the raw physical measurements with continuous values are replaced with semantically rich features. The main goal of the feature extraction is to focus the HMM on the features used by humans when identifying a team action. Many of these features, such as velocity or curvature are extracted over a sliding window of time, thus containing historical information which is very difficult to encode in a HMM. A discretization process is developed which maps the real-valued features to representative discrete symbols.

5.1 Feature Functions

We classify the extracted features in agent-oriented, environment-oriented and team-oriented feature functions.

- The *agent-oriented feature functions* extract information from individual agents such as acceleration, velocity and curvature.
- *Environment oriented feature functions* extract features from agent or team interactions with objects in the environment, such as line of sight (LOS) and object collision.
- The *team-oriented feature functions* focus on extracting features being expressed relative to the team such as cohesion, team orientation and relative position vectors.

5.2 Agent-Oriented Feature Functions

Agent-oriented feature functions extract information from the movement of a single agent over a sliding time-window. The features, together with the allowed discrete values, are summarized in Table 5.1. As they refer to individual agents, agent oriented features can not efficiently describe the correlations among the team members. They can, however, improve the recognition performance when used in combination with team oriented feature functions. Agent oriented features also play a significant role in recognizing team actions where specific agents play determining roles such as the leader or captain.

Table 5.1: Overview of the agent-oriented feature functions.

Name	Discretization	Description
Velocity	{Idle, Low, High}	The rate of change of position.
Acceleration	{Accelerating, Decelerating, Constant}	The rate of change of velocity.
Orientation	{North, South, West, East, NorthEast, NorthWest, SouthEast, SouthWest}	Eight-way orientation with respect to a global reference system.
Orientation4	{North, South, West, East}	Four-way orientation with respect to global reference system.
Turning	{Steady, Left, Right}	The change in orientation.
Curvature	{Straight, Curved}	The rate at which a curve changes direction.

5.2.1 Velocity

Velocity is a measure of the rate of change of position. The velocity can be used to discriminate teamwork activities where the individual team members move at different velocities. Velocity is discretized using a user defined threshold parameter that determine if an agent's velocity is *Low* or *High*. If the velocity is 0 then the discrete output is *Idle*.

5.2.2 Acceleration

Acceleration is the rate of change of velocity with respect to time. Acceleration is typically used in conjunction with velocity in teamwork recognition for improved recognition accuracy. Discretization is performed by determining whether the acceleration value is positive or negative. Hence, the discrete output of this function is *Accelerating* or *Decelerating*.

5.2.3 Orientation

Orientation is the angle that depicts the direction in which agents are facing. The orientation discretization process assigns the orientation of an agent to one of eight discrete orientation values representing north, south, west, east, north-east, north-west, south-east and south-west.

The *Orientation4* feature function is a simpler version of the *Orientation* feature function where only four cardinal orientations (north, south, west and east) are used in the discretization process.

5.2.4 Turning

The *Turning* feature function extract the rate of change of an agent's orientation. The discrete output of the feature function is determined by a user-defined threshold parameter. The discrete outputs are *Left* or *Right* and if the gradient is below the threshold then the discrete output is *Steady*.

5.2.5 Curvature

Curvature is a measure of the rate at which the trajectory of the agent changes direction. The purpose of the *Curvature* feature function is to determine if an agent's movement trace is curved or straight in the two-dimensional Euclidean space. A simple measure of the curvature can be defined as:

$$\kappa(x) = \frac{|f''(x)|}{(1 + (f'(x))^2)^{3/2}} \tag{5.1}$$

The value $\kappa(x)$ is interpreted as follows: when $\kappa(x)$ is small the movement trace is straight and when $\kappa(x)$ is large the movement trace is curved [Ste99].

The first and second derivative can be calculated directly from the movement trace of each agent (or team centroid) over a sliding window in time using a finite difference approximation (with central difference). In this study we employ Equation 5.2 and Equation 5.3 for this purpose, where s depict the size of the sliding window.

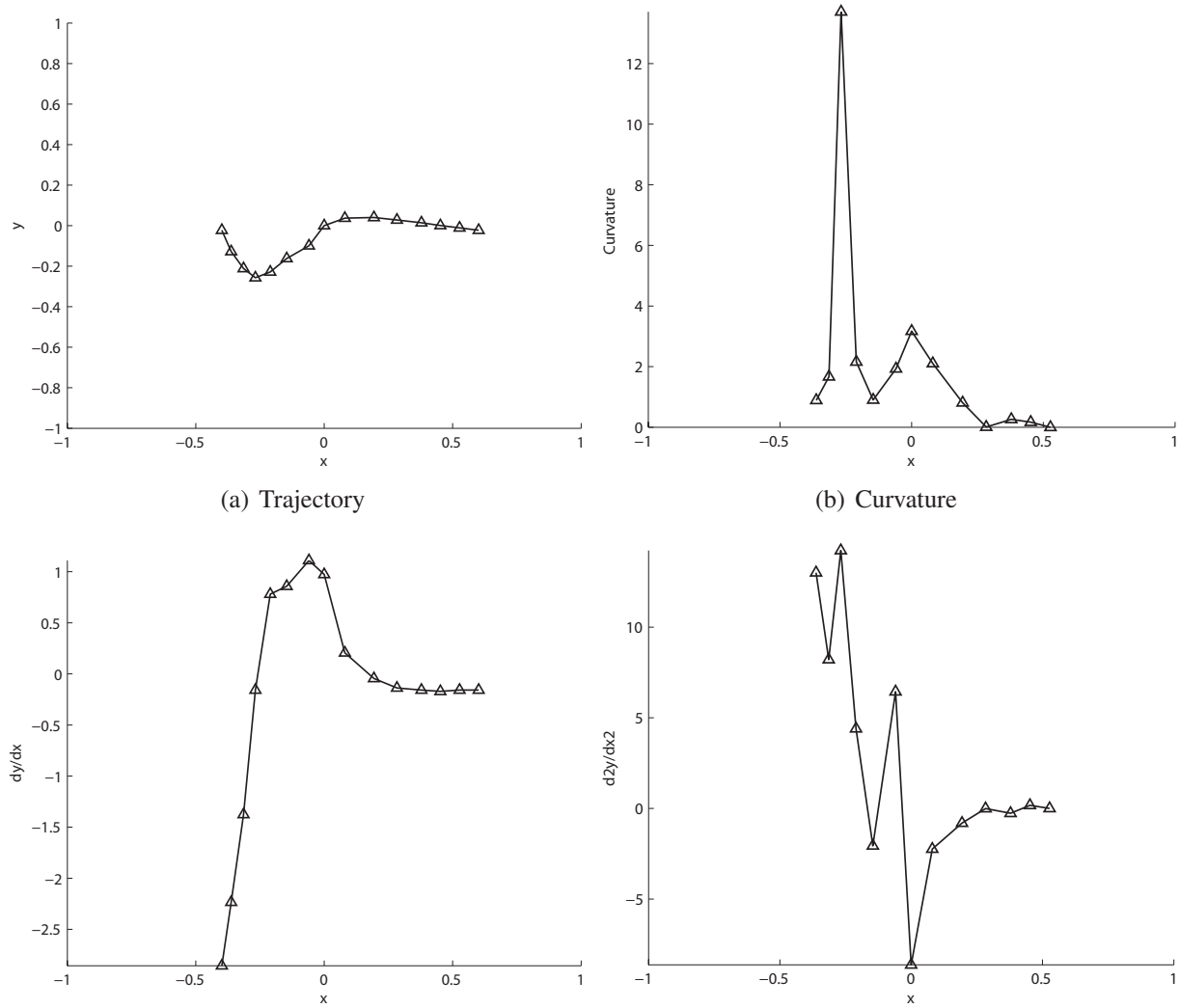
$$f'(x_i) = \frac{f(x_{i+s}) - f(x_{i-s})}{x_{i+s} - x_{i-s}} \tag{5.2}$$

$$f''(x_i) = \frac{\frac{f(x_{i+s})-f(x_i)}{x_{i+s}-x_i} - \frac{f(x_i)-f(x_{i-s})}{x_i-x_{i-s}}}{x_{i+s} - x_{i-s}} \tag{5.3}$$

Figure 5.1 shows the trajectory, estimated curvature, first derivate and second derivative plot for a sequence of agent positions. The curvature was calculated using a sliding window of size 1 ($s = 1$). Note that the agent trajectory is normalized prior to calculating curvature. The normalization is performed similarly to the pre-processing procedure presented in Section 4.1. In addition to translating and rotating the trajectory we also scale the trajectory so that all trajectories can be represented in the same space. The normalization procedure ensures that the curvature feature is invariant to translation, rotation and scale.

5.3 Environment-Oriented Feature Functions

Environment oriented feature functions extract features from interactions of the team members with objects the environment. The objects of the environment can be physical or virtual; both types can be static or dynamic. Terrain features such as buildings or roads are physical and static. The offside line in a soccer game is a virtual and dynamic environmental object that can be inferred from the position of the opponent player who is second nearest to the end of the playing field. The



(a) Trajectory

(b) Curvature

(c) First Derivative

(d) Second Derivative

Figure 5.1: Agent trajectory with estimated curvature and derivatives.

front-line in a war is a similar dynamic and virtual object. Table 5.2 provides a summary of the environment-oriented feature functions[1].

Table 5.2: Overview of the environment-oriented feature functions.

Name	Discretization	Description
Collision	{?Object, None}	Collision with nearest environment object.
LineOfSight	{?Object}	Nearest environmental object in the agent's LOS.
LineOfSightDistance	{Near, Far, None}	Euclidean distance to object in LOS.

5.3.1 Collision

The *Collision* feature function determine if an agent's path collide with or intersect the boundaries of an environmental object. For example, the feature can be used to determine if a football player cross the line of scrimmage during a play.

5.3.2 LineOfSight

The *LineOfSight* feature function is used to determine if an agent's line of sight (LOS) is intersecting the boundaries of any environmental objects. The feature function is parameterized by an offset angle, *Angle*, and a distance metric, *LOS* , that depicts the direction and the maximum length of the

[1]In many applications environmental information is not available to the recognition system. Acquiring this information can be a time consuming and difficult task. For this purpose, we have developed a tool which is integrated in the *Teamwork Scenario Editor* to interactively manage, add and remove physical objects in environments.

LOS respectively. The agent orientation is used as a reference point for the offset angle parameter. Hence, an offset of 0 degrees result in a line of sight that is aligned with the agents original orientation. Figure 5.2 shows agent *A1* in a typical configuration where the LOS is intersected with an obstacle.

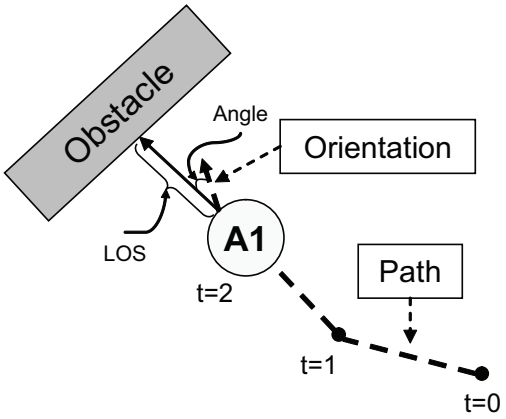


Figure 5.2: Obstacle intersection with line of sight for agent A1.

5.3.3 LineOfSightDistance

The Euclidean distance to an object can be extracted using the *LineOfSightDistance* feature function. The function is parameterized with an offset angle, *Angle*, and the maximum line of sight, *LOS*. The *LineOfSightDistance* feature function output a real-valued distance that need to be discretized using a user defined threshold value. The discrete symbols in this case are *Near*, *Far* and *None* in the case no object are detected in the agent's LOS.

5.4 Team-Oriented Feature Functions

Team-oriented feature functions extract features which are expressed *relative to the team*. In contrast to the previously discussed features, the team-oriented features are designed for the purpose of discriminating teamwork activities. Henceforth, it is expected that the these feature functions will extract the most influential features for teamwork activity recognition systems.

We consider two types of team-oriented features. The first type of feature functions, summarized in Table 5.3, are obtained by replacing the (potential) team with a virtual agent following the team centroid. In addition to these features we designed a series of team-oriented features which are not directly related to agent-oriented feature functions. These features are summarized in Table 5.4.

Table 5.3: Overview of the first type of team-oriented feature functions.

Name	Discretization	Description
TeamVelocity	{Idle, Low, High}	The mean rate of change of team positions.
TeamAcceleration	{Accelerating, Decelerating, Constant}	The mean rate of change of velocity with respect to time of team members.
TeamOrientation	{North, South, West, East, NorthEast, NorthWest, SouthEast, SouthWest}	The orientation of the centroid of the team.
TeamOrientation4	{North, South, West, East}	The orientation of the centroid of the team.
TeamTurning	{Steady, Left, Right}	The mean rate of change of orientation of the team.
TeamCurvature	{Straight, Curved}	The mean rate at which team-member agent's curves changes direction.

Table 5.4: Overview of the second type of team-oriented feature functions.

Name	Discretization	Description
Centroid-Relative Position Vector	$\{A_1, A_2, \ldots, A_n\}$	The relative eight-way position of each team member with respect to team centroid and direction.
Role-Relative Position Vector	$\{A_1, A_2, \ldots, A_{n-1}\}$	The relative eight-way position of agents with respect to a privileged agent's centroid and direction.
Cohesion	{Separated, Merged}	The bonding together of team members.
CohesionGradient	{Separating, Merging, None}	The rate at which the bonding of team members are changing.
CohesionDirection	{Vertical, Horizontal, Upward, Downward}	The principal component direction of the positions of team members.

In the following we discuss the justification and implementation of the second type of feature functions.

5.4.1 Centroid-Relative Position Vector

Centroid-Relative Position Vector (CRPV) is a vector of size n where for every agent we record the semi-quadrant it occupies in a Cartesian coordinate system centered in the centroid, with the *North* direction aligned with the movement of the team centroid over a sliding window in time. Figure 5.3 shows the encoding in the CRPV of a team of three agents.

There are several advantages of the CRPV compared to the absolute positions of the team members. First, the dimensionality of the input data is reduced. Second, the CRPV is invariant to translation, rotation and scale.

Determining the CRPV does not require full role recognition to be performed before its calculations, but it requires us to eliminate the bystander agents (which would shift the position of the centroid). Thus, CRPV can serve as a feature used in role recognition.

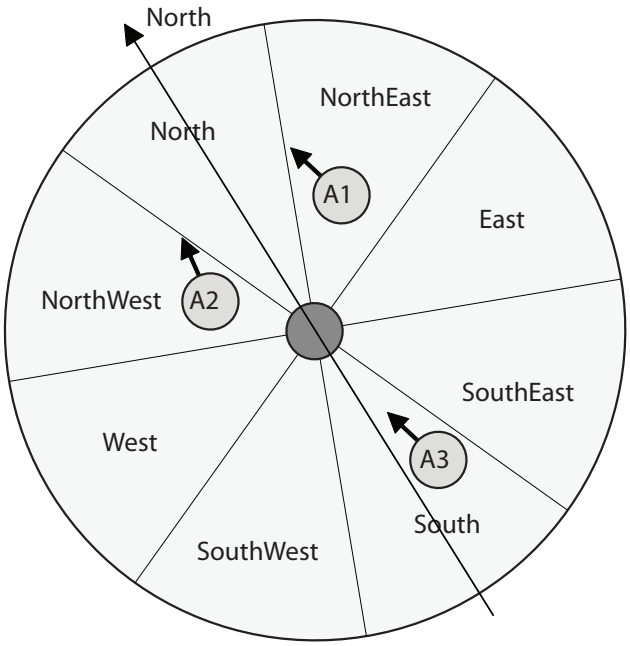


Figure 5.3: The centroid-relative position vector a team of three agents. The team centroid is marked with a dark shaded circle. The position of agents *A1*, *A2* and *A3* are described by the vector {*NorthEast, NorthWest, South*}.

5.4.2 Role-Relative Position Vector

Role-Relative Position Vector (RRPV) is an evolution of the CRPV feature. The RRPV is a vector of size $n-1$, where each element encodes the semi-quadrant occupied by the agent in a coordinate system centered on one agent which has a privileged role. The coordinate system is aligned with the movement of the privileged agent. One of the advantages of such system is that we can, for instance, express the movement of the bodyguards relative to a VIP, or the movement of a group of pickpockets around a victim.

The disadvantage of this encoding is that at least the role of the privileged agent needs to be determined before feature extraction. Finally, the privileged role needs to be clearly marked in the idealized team action (ITA), which requires human intervention in the process.

5.4.3 Cohesion, CohesionGradient and CohesionDirection

Intuitively, the *Cohesion* feature measures how tightly grouped together are the team members, the *CohesionDirection* shows the direction of the greatest cohesion, while *CohesionGradient* measures the variation of the cohesion. For instance, a team which is splitting in two groups is loosing cohesion. To extract a set of values which match well with this intuition, we use a method based on principal component analysis [DHS00]. We first calculate a scatter matrix

$$\mathbf{S} = \frac{\mathbf{1}}{\mathbf{n-1}} \sum_{\mathbf{k=1}}^{\mathbf{n}} (\mathbf{x_k} - \mathbf{m})(\mathbf{x_k} - \mathbf{m})^{\mathbf{t}}$$

where $\mathbf{x_k}$ is the position of agent k in a team consisting of n members and $\mathbf{m}$ is the location of the centroid:

$$\mathbf{x_k} = \begin{pmatrix} x_k \\ y_k \end{pmatrix} \text{ and } \mathbf{m} = \frac{1}{n}\sum_{k=1}^{n} \mathbf{x_k}$$

Next, we analyze the scatter matrix by finding its eigenvalues, $\boldsymbol{\lambda}$, and eigenvectors, $\mathbf{e}$. In our case, since the position of each agent is described in two dimensions, we will find two eigenvalues and two eigenvectors. Geometrically $\mathbf{e}$ and $\boldsymbol{\lambda}$ represent the principal directions of the team member positions and the magnitude of the directions respectively. The *Cohesion* feature is the value of the largest eigenvalue, λ_{max}.

The *CohesionDirection* feature function uses the eigenvector corresponding to λ_{max} to output the principal orientation of the team.

The *CohesionGradient* feature calculates the change of the cohesion value over a sliding time window of the historical values of cohesion.

Figure 5.4 illustrates a scenario where a team of four agents change formation over time. For each time step the figure shows the principal orientation vectors of the team. As discussed above, the larger of these vectors is chosen as the cohesion vector (and it is represented in bold in the figure). Initially, at $Time = 0$, the team members are in near vicinity of each other; λ_{max} is small and the output of the *Cohesion* feature is *Merged*. Next, at $Time = 1$, the team members separate in the x-direction; λ_{max} is increasing and the output of the *CohesionGradient* feature is *Separating*.

Finally, at $Time = 2$, the separation of the team yields a large enough λ_{max} value, such that the output of the *Cohesion* feature is *Separated*.

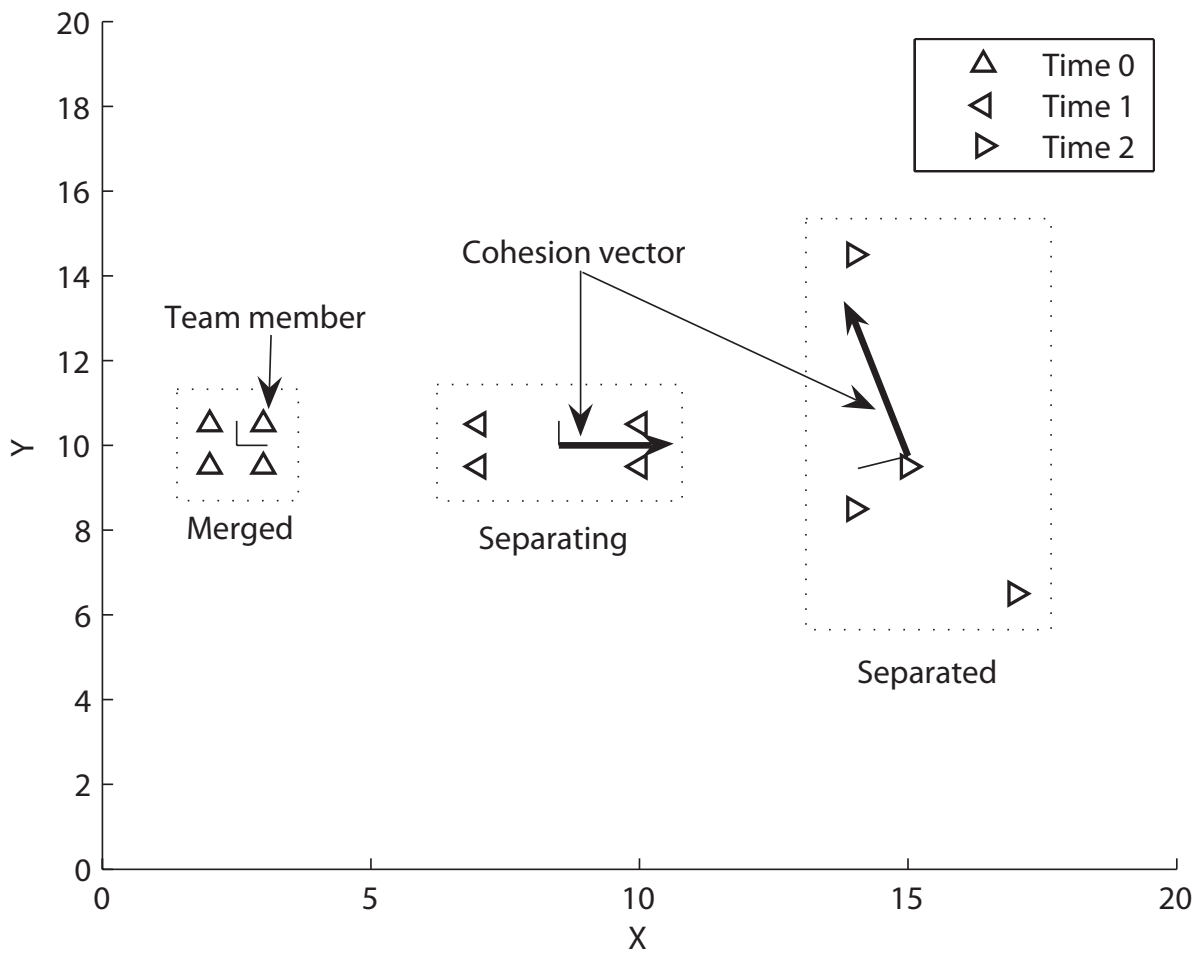


Figure 5.4: Plot of a scenario where a team of four agents change formation over time. The cohesion vector depicts the cohesion and principal orientation of the team.

5.5 Discussion

In this chapter we have developed several feature extraction functions that are specifically designed for teamwork activity recognition. In addition to centroid based features that measures velocity, acceleration, etc., we have developed feature functions to extract relative position of agents in the team and multiple cohesion based features. For instance, the centroid-relative position vector (CRPV) can be used to discriminate different formations in the teams. We have used the team-oriented features in the role-based teamwork recognizer presented in Chapter 6 and in the performance tuning study in Chapter 7.

We have also developed a set of agent-oriented features. These feature functions can be used as complements to the team-oriented features to improve recognition performance. In Chapter 6 the agent-oriented feature functions are used to recognize roles in the team.

CHAPTER 6: ROLE-BASED TEAMWORK ACTIVITY RECOGNITION IN OBSERVATIONS OF EMBODIED AGENT ACTIONS

In this chapter we extend the teamwork activity recognition workflow with a role recognition module which allows us to represent the idealized teamwork activities by roles as opposed to by agents (which was the approach taken in Chapter 4). The role recognition module allows us to present the recognizer with arbitrarily shuffled input, and still obtain high recognition rates.

6.1 Role Recognition

In systems which recognize teamwork from coordinated movement of agents, the input of the recognizer is a fixed arrangement of the recorded features of the agents. A four agent team might be represented by a feature vector $\{x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4\}$ which is either the direct input of the HMM as in Chapter 4, or the input to the feature extraction process, as shown in Chapter 5. However, what does "agent 3" mean? If we are observing a team sport, such as basketball, we can say that "agent 3" corresponds to the player wearing the number 3. However, in many applications such as surveillance, such clear identification might not be possible. Even in the case of team sports, players might switch roles during the game. An HMM which was trained with an input order $\{1, 2, 3, 4\}$ will not recognize the same action if it appears encoded in the order $\{2, 3, 4, 1\}$. A related issue is concerned with the number of agents participating in the team action.

Many essentially identical team actions can be executed with various team sizes. For instance, the defining feature of bounding overwatch is the alternating advancement of two groups of agents. This definition remains valid whether the action is performed in a configuration of 1+2, 2+1, 2+2 or 2+3 agents.

One brute force solution is to train a separate HMM for every possible team and role size combination. Then, at recognition time, we repeat the matching algorithm for every possible permutation of the input data, and pick the one which gives the best match. This is a significant computational penalty, especially in cases where the analyzed observations contain agents which are not part of any team.

In the following, we describe a role recognition module which allows us to recognize what are the (likely) roles played by the agents in a team action before the team action was recognized. Running such a role recognition module before the team recognition allows us to build the idealized team action (ITA) based on roles, rather than agents.

Figure 6.1 illustrate the full workflow of the teamwork activity recognition framework extended with the role recognition module.

6.1.1 Learning Role Models from Observations

A role model allows us to determine the likely roles the agents would play in a team action. To understand how this is possible without the need to recognize the team action as well, let us remember that certain roles are strongly identified with certain feature values.

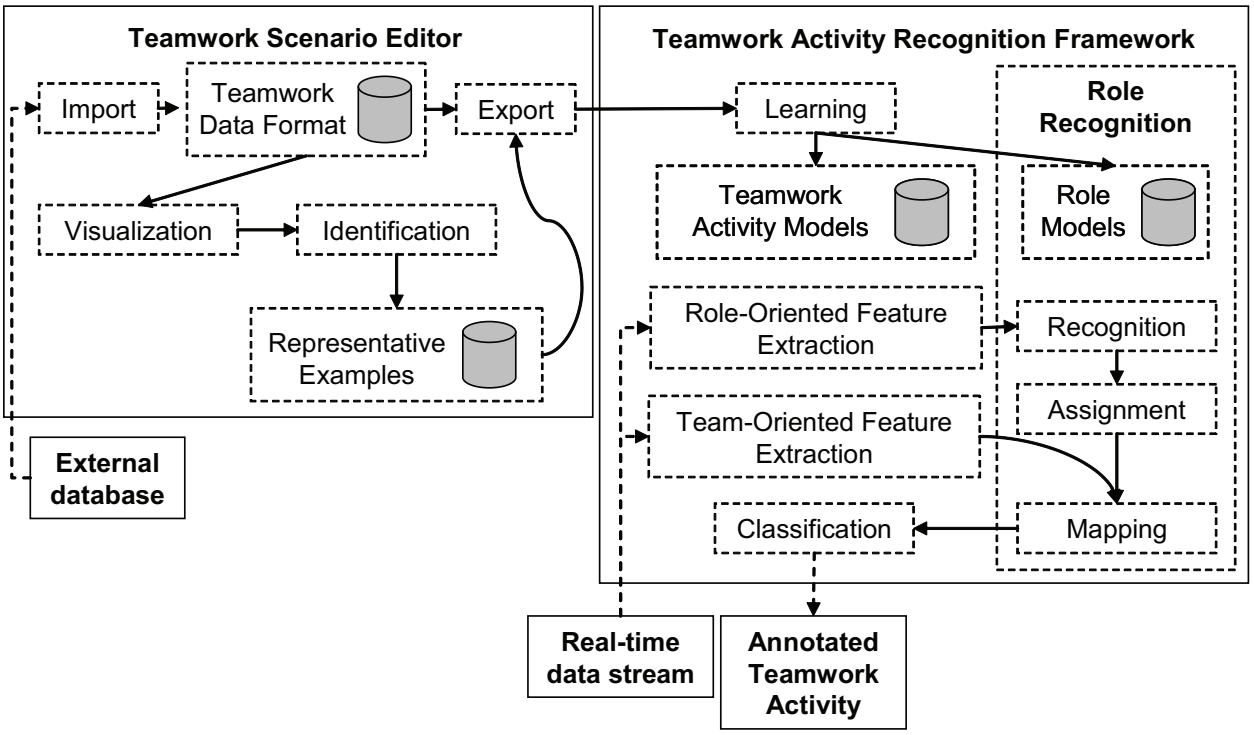


Figure 6.1: The workflow of teamwork activity recognition system extended with role recognition.

For instance, in a "follow the leader" action, the leader is at the forefront of the team. Looking at the formation, we can identify which agent that "can be" the leader, even before we are certain that the team action is, indeed, "follow the leader". The best feature to decide on this is the centroid-relative position vector; the team leaders will need to be in the *North* semi-quadrant.

For identifying the two sub-teams of the bounding overwatch team action, however, the relative positions are useless. The best features for this determination are the velocity vectors: the members of one team are stationary, while the other ones are moving.

We can draw several conclusions from these examples. First, we need a different feature set for role recognition than for team action recognition (although overlaps, of course, exists). As

role recognition is targeted towards individual agents, the agent-oriented feature functions have a significant role. Second, the discriminating feature changes from role to role. Thus, for our role recognition model we chose a decision tree representation, which has the advantage that besides representing the role model, it also allows us to extract and visualize the exact features which were used in the classification.

The decision tree learning algorithm used in our study was the ID3 algorithm followed by a tree pruning procedure to minimize the effects of over-fitting [Qui88].

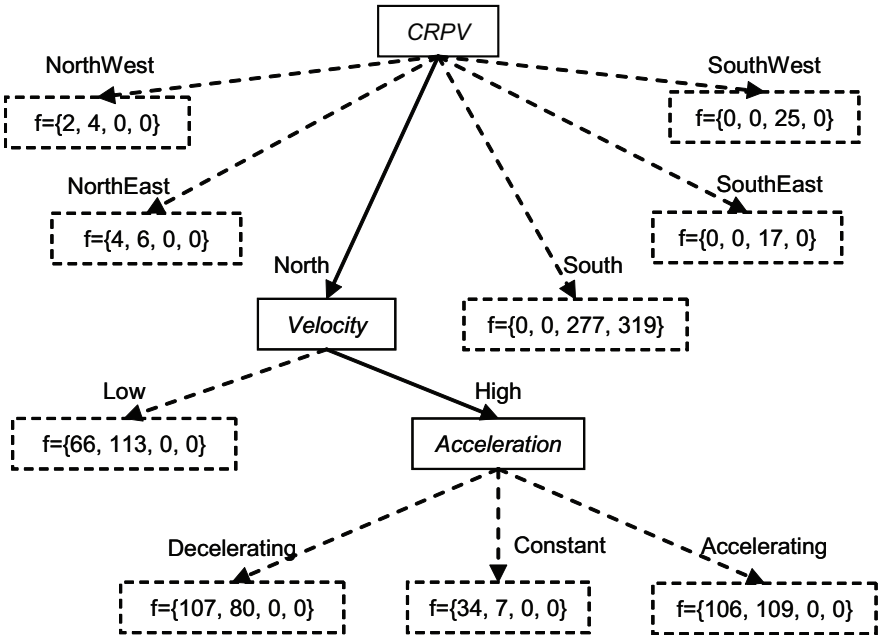


Figure 6.2: Role model represented by a decision tree.

Figure 6.2 illustrate an example of a role model represented by a decision tree. The tree was trained using discretized agent observations for four different roles. The dashed rectangles repre-

sent leaves in the tree, which in turn contains class frequencies, $\mathbf{f} = \{\mathrm{f}_{r_1}, \mathrm{f}_{r_2}, \mathrm{f}_{r_3}, \mathrm{f}_{r_4}\}$, for the roles. The solid rectangles are internal nodes depicting the features required for classification.

6.1.2 Acquiring Role Assignment Probabilities

The decision trees, once trained, can be used to calculate the role assignment probabilities. The first step is to extract observation sequences from the movement traces of each agent, which are used as input to the decision tree classifier. The output of the classifier is a class frequency vector, $\mathbf{f}^t$, for every observation t in the sequence. The probability $Pr(a_i, r_j)$ that agent a_i is playing role r_j is calculated with the following formula:

$$Pr(a_i, r_j) = \frac{\sum_{t=1}^{T} \mathrm{f}^t_{r_j}}{\sum_{k=1}^{K} \sum_{t=1}^{T} \mathrm{f}^t_{r_k}}$$

where T is the length of the observation sequence and K is the number of available roles.

6.1.3 Role Assignment and Mapping

The goal of the role assignment process is to identify the best match of role to agent assignments by searching the $Pr(a_i, r_j)$ values so that the team-oriented feature vectors, which are used by the teamwork recognizer, can be re-mapped accordingly. In this section we describe two distinct strategies for this purpose.

Multiple Role Assignment

The first strategy, described in Algorithm 1, allows agents to play multiple roles. For each role in the *Pr* table the agent with the maximum probability is selected to represent the role.

Algorithm 1 Multiple role assignment algorithm.

Require: Role model, $RoleModel$, and agent observations, $\mathbf{O}_{a_i}$.
Ensure: Mapping, Map, of agents, a_i, to roles, r_j.

```
Pr(a_i, r_j) = Apply(RoleModel to O_{a_i})
for all Roles r_j in Pr(a_i, r_j) do
    i_max = arg max_i Pr(a_i, r_j)
    Assign(r_j → a_{i_max} in Map)
end for
return Map
```

Unique Role Assignment

The second strategy, described in Algorithm 2, limits agents to play only one role. A set of candidate assignments is first identified for all available agents. The candidate set is generated by traversing the roles of each agent and for each agent selecting the role with maximum probability. Next, conflicts are resolved among the candidate assignments, by selecting the agent with maximum probability to represent the role. The agent is then marked as already assigned a role and not available for future allocation. The procedure is repeated until all roles have been assigned to an agent.

Algorithm 2 Unique role assignment algorithm.

Require: Role model, $RoleModel$, and agent observations, $\mathbf{O}_{a_i}$.

Ensure: Unique mapping, Map, of agents, a_i, to roles, r_j.

```
Pr(a_i, r_j) = Apply(RoleModel to O_{a_i})
while Unmapped roles remaining do
    for all Agents a_i in Pr(a_i, r_j) do
        j_max = arg max_j Pr(a_i, r_j), where j ≠ k, ∀r_k in Map
        Assign(r_{j_max} → a_i in HypothesisMap)
    end for
    for all Roles r_{j_max} in HypothesisMap do
        i_max = arg max_i Pr(a_i, r_{j_max}), where i ∈ k, ∀a_k of r_{j_max} in HypothesisMap
        Assign(r_{j_max} → a_{i_max} in Map)
    end for
end while
return Map
```

6.2 Modeling Team Actions with Multiple Hidden Markov Models

In practical applications of teamwork recognition it is likely that the complexities of the activities varies significantly. Hence, identifying the optimal number of hidden states that can encode each activity is important to maintain good recognition accuracy (complicated teamwork patterns requires more hidden states than simple ones).

In this study, we represent a idealized team action with multiple, internal HMMs estimated with different number of hidden states. At the cost of increased processing time for parameter estimation and classification, compared to the approach taken in Chapter 4, this approach is more robust towards variations in activity complexity.

The teamwork activity classification algorithm using multiple internal HMMs can be separated into three phases:

1. The probability $P(\mathbf{V}^T \mid \boldsymbol{\theta})$ of the input sequence is calculated for all internal HMMs in each activity model using the forward evaluation algorithm [DHS00].

2. The internal HMM with maximum $P(\mathbf{V}^T \mid \boldsymbol{\theta})$ is selected to represent each activity model.

3. Finally, the activity model with maximum $P(\mathbf{V}^T \mid \boldsymbol{\theta})$ is chosen as the final classification.

6.3 Experimental Results

6.3.1 Experimental Setup

We performed a series of experiments using a dataset which was acquired from the real world military exercise from Section 3.5.1. The dataset consists of observations of four tanks which performed seven distinct teamwork activities. To achieve a larger corpus, the approximately 40 initial team action observations were multiplied synthetically by applying various distortion operations. The resulting semi-synthetic dataset is summarized in Table 6.1.

The original observations were made using high resolution GPS devices embedded in the tanks, thus for practical purposes, they are free of observation noise. For practical applications, which require observations of opponent agents, such accuracy is not achievable. To simulate observation noise in our dataset, we have added Gaussian noise to the position of each agent by offsetting the coordinates with a randomly generated number following the Gaussian distribution, $N(\mu = 0, \sigma^2 = 1)$, multiplied with a noise magnitude ϵ.

Table 6.1: Dataset summary of teamwork activities, number of sequences and observations that were used during training and validation.

Activity	Sequences	Observations
Traveling column	29	319
Traveling line	30	330
Traveling box	33	363
Bounding overwatch	9	189
Wedge	17	187
Team split	15	192
Team merge	15	177

6.3.2 Creating the Idealized Team Actions

In our workflow the idealized team actions were encoded in the form of multiple internal HMMs as described in Section 6.2, which were trained using representative examples of the team action. These examples were extracted from the observation flow using the teamwork scenario editor and labeled by hand. The positional observations were fed into the feature functions which extracted the team-oriented features.

The HMM was trained using the Baum-Welch expectation maximization algorithm for a predetermined number of states. We assumed that the emission probabilities of the HMM are described by a Gaussian Mixture Model (GMM) multi-modal probability density distribution. The number of hidden states and the number of Gaussian components are parameters of the learning process.

Although one can try to analytically infer appropriate values for these parameters, this would require an in-depth knowledge of the team actions, as well as the number of variants which are

labeled the same by human observers. The automatic tuning of these parameters can therefore save a lot of modeling effort.

We have optimized these parameters by repeating the learning process for every combination of the number of hidden states and Gaussian mixture components in the 1 to 5 range with $\epsilon = 2$. For each case we have tested the recognition accuracy using a 10-fold cross-validation procedure. The resulting accuracy values are shown in Figure 6.3. The optimal parameter configuration was found to be 4 hidden states and 3 Gaussian mixture components. Hence, each teamwork activity is represented by 4 internal HMMs with hidden states ranging from 1 to 4. Figure 6.4 show the distribution of correctly classified sequences among the internal HMMs for each teamwork activity in the optimal parameter configuration.

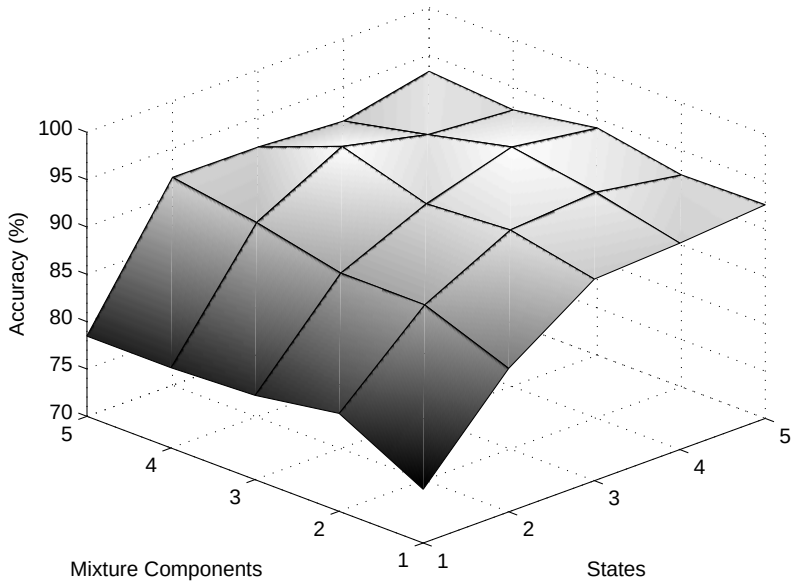


Figure 6.3: Mean accuracy of teamwork activity recognition accuracy. Hidden states and number of Gaussian mixture components range from 1 to 5.

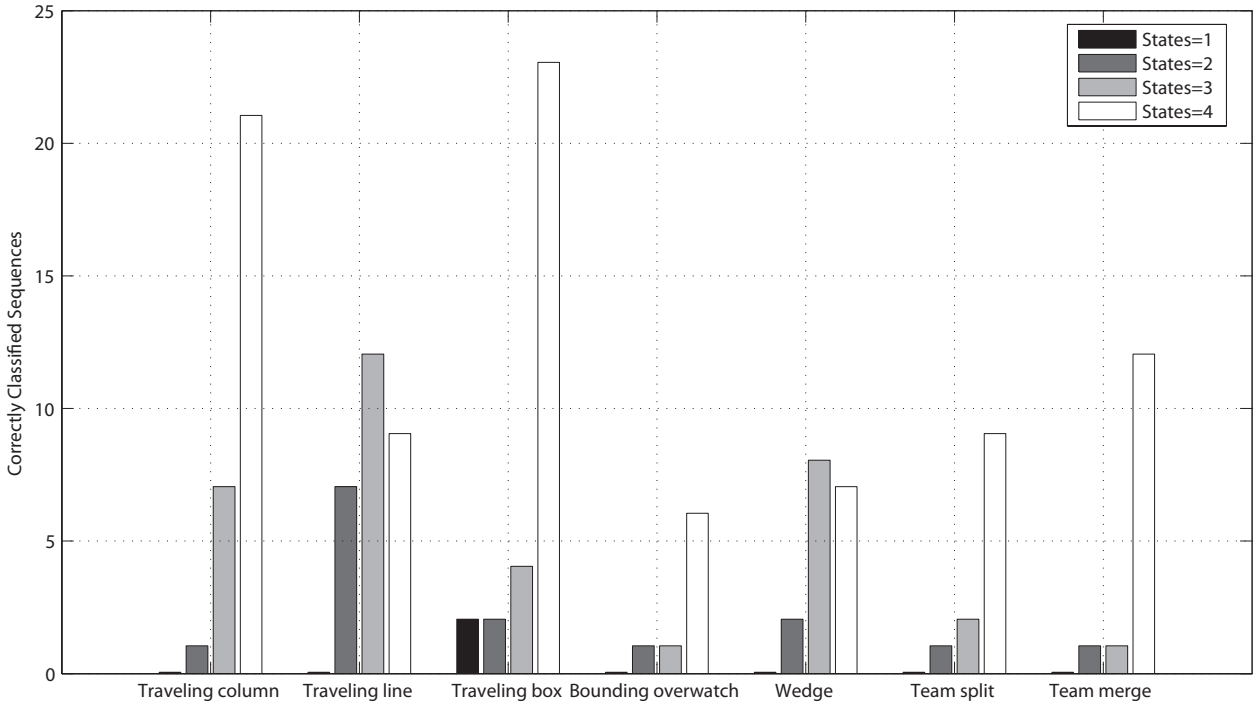


Figure 6.4: Bar-plot showing the distribution of correctly classified sequences among the internal HMMs for each teamwork activity.

6.3.3 Performance Evaluation with Unknown Team-Organization

The experiments in this section measure the performance of teamwork activity recognition in applications where the role assignments are unknown. To simulate unknown role assignments in the dataset, we have randomly shuffled the agent observations.

Figure 6.5 shows the teamwork activity recognition performance, using 10-fold cross-validation. The observation sequences are distorted with Gaussian noise with the multiplier ϵ ranging from 0 to 10. We repeat the experiments with role assignment algorithms 1 and 2. For comparison, we performed a series of experiments under the assumption that the agents are always correctly assigned

to the roles, as well as a series of experiments where the randomly shuffled agent observations were fed into the recognizer without role recognition.

From the experimental results in Figure 6.5 we can conclude that our system is able to recognize and assign roles to agents while still maintaining good teamwork recognition accuracy using both algorithms. *Algorithm 2* slightly outperforms *Algorithm 1* for this problem because the dataset used in this experiment consisted of teamwork activities performed by four agents where each agent always play a unique role in the team.

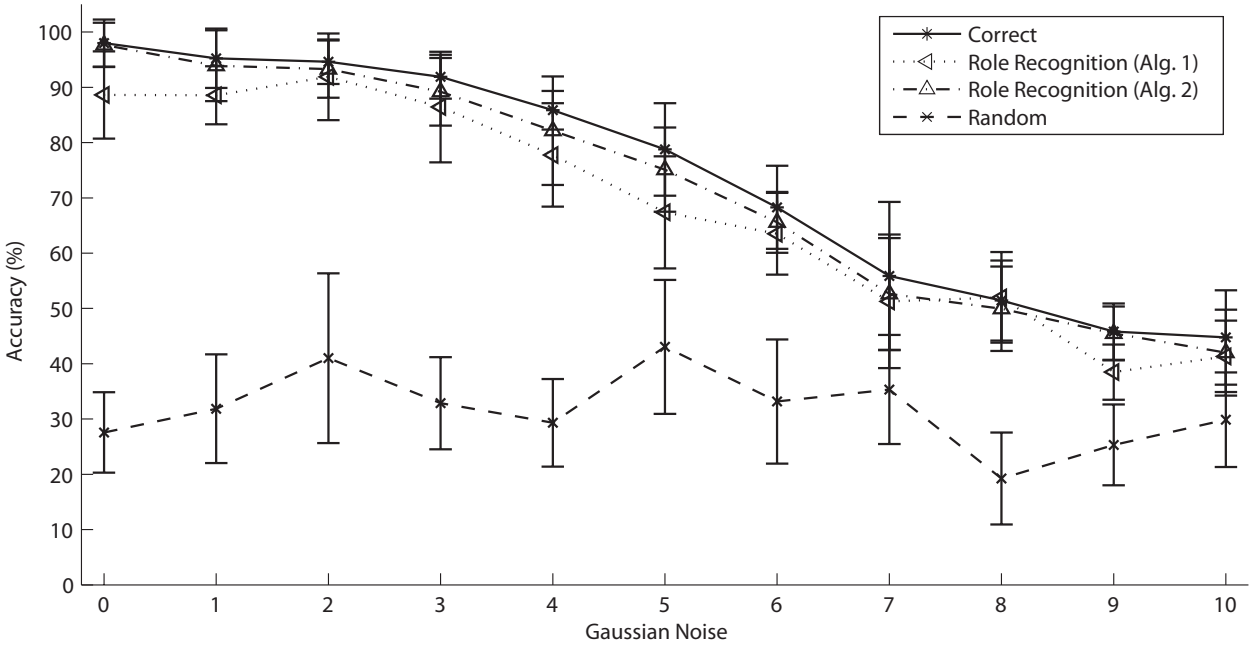


Figure 6.5: Mean recognition accuracy with error bars depicting standard deviation.

Table 6.2 illustrates the confusion matrix for teamwork recognition with *Algorithm 2* when $\epsilon = 2$. In this case the mean accuracy is 92.62% with standard deviation 5.53%.

As a point of reference, in Chapter 4 we achieved recognition rates of approximatively 82% using noise-free data and the agent observations in the correct order. We conclude that with the role recognition module we expect a system which achieves a better performance under less favorable input assumptions: noisy data and input in which the order of the agents was randomly shuffled.

Table 6.2: Confusion matrix depicting teamwork activity recognition performance with unknown team-organization.

			True							Precision
			a	b	c	d	e	f	g	(%)
Predicted	Traveling column	a	29	0	0	0	0	0	0	100.0
	Traveling line	b	0	26	0	0	0	0	0	100.0
	Traveling box	c	0	3	32	2	0	0	0	86.49
	Bounding overwatch	d	0	0	1	7	0	0	0	87.50
	Wedge	e	0	0	0	0	17	0	0	100.0
	Team split	f	0	0	0	0	0	13	2	86.67
	Team merge	g	0	1	0	0	0	2	13	81.25
	Recall (%)		100.0	86.67	96.97	77.78	100.0	86.67	86.67	

6.4 Conclusions

In this chapter we described a role recognition module, which allowed us to recognize the possible roles played by the agents before the team action is recognized. This allowed us to represent the idealized team action in terms of roles (rather than agents), and allowed the recognizer workflow to recognize the action even if the input does not contain information about the roles played by the agents (a situation frequently encountered in applications such as surveillance).

By integrating the role recognition module into the teamwork recognition workflow and by representing the teamwork activities with multiple internal HMMs we have found that, even in the presence of higher levels of noise and where the input is shuffled, the recognition accuracy is improved compared to the results from Chapter 4.

CHAPTER 7: TUNING THE PERFORMANCE OF TEAMWORK ACTIVITY RECOGNITION THROUGH THE CHOICE OF PDF

In the previous chapters we have outlined a complete workflow for team action recognition. Many of the components of this workflow are customizable through a variety of parameters:

- The HMMs can be customized through the number of hidden states, through the probability density function (PDF) choices and through the learning parameters.
- The discrete feature extractions are parameterized through the width of the sliding window, and through the various discretization threshold values.
- The role recognition module is parameterized through the choice of features used in role-recognition, as well as in the parameters of the ID3 learning algorithm (and indeed the choice of the algorithm).

The appropriate choice for all these parameters can have a significant influence on the accuracy and robustness of the recognizer. Many of these choices, however, have only incremental impact. For instance, discretizing the orientation into 4, 8 or 16 directions is simply a matter of the accuracy we want to reach.

We find however, that the choice of the probability density model has a significant impact on the recognition accuracy and it is strongly related to the way in which humans perceive the team actions.

In this chapter we study the ways in which the choice of PDF affects the accuracy and robustness of team activity recognition, and discuss their advantages and disadvantages.

There are several approaches to estimate emission probabilities in the form of discrete or continuous probability density functions:

- Simple HMM estimate emission probabilities using a discrete, or histogram-based, PDF.
- Estimate emission probabilities using the multi-variate Gaussian PDF (the approach taken in Chapter 4).
- Employ a Gaussian Mixture Model (GMM) PDF to estimate emission probabilities (the approach taken in Chapter 6).

In Table 7.1 we quickly summarize the implementation choices and their characteristics.

Table 7.1: A summary of the implementation choices.

Implementation Choice	Multi-Modal	Continuous	Parameters
Simple HMM	Yes	No	Hidden states, Clusters (k)
HMM with Gaussian	No	Yes	Hidden states
HMM with Gaussian Mixture Model	Yes	Yes	Hidden states, Components (C)

7.1 Simple HMM

In the simple HMM the input observations are discrete values and the emission probabilities in each hidden state are represented by a histogram-based probability distribution. However, in teamwork activity recognition and many other application domains the observations are real-valued and multi-dimensional. The simple HMM can be employed in these applications as well. However, a pre-processing step is required to map the real-valued observations into discrete observations.

Vector quantization is a one approach which can be employed to "compress" the real-valued and multi-dimensional observations into discrete values that in turn can be fed as input to the HMM [Gra84]. This method has been applied with success in early speech-recognition software and in image based phrase spotting systems [CWB95].

The simple HMM approach for teamwork activity recognition employs a vector quantization method to discretize the observations acquired from the team-oriented feature functions. This process is visualized in Figure 7.1. The vector quantization approach can be divided into two separate phases:

- *Training:* The first phase of vector quantization consists of the generation of a codebook which can map feature vectors to discrete symbols. The codebook can be manually acquired through knowledge engineering, however, in our approach we automatically generate the codebook from training data using the k-means clustering algorithm [DHS00]. The k-means clustering algorithm is parameterized by the number of cluster centroids, k, which the train-

ing data can belong to. In other words, this parameter determines the number of discrete symbols which can be used in the HMMs.

- *Cluster Assignment:* The second phase in vector quantization consists of assigning discrete symbols to the incoming feature vectors. First, the distances between the input feature vector and each cluster centroid in the codebook are calculated. Then, the symbol (index) of the cluster centroid with the minimum distance is selected to represent the feature vector.

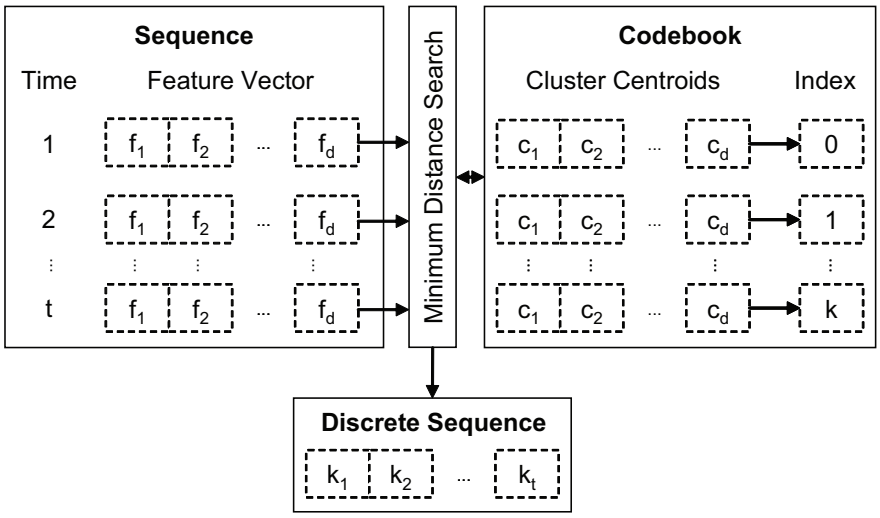


Figure 7.1: The workflow for transforming a sequence of feature vectors using the vector quantization method. Each feature vector is assigned an index that represent the codebook cluster centroid with minimum distance relative the input feature vector. The output is a sequence of codebook indices which can be fed to the teamwork recognizer.

The main advantage of using the simple HMM, besides its simplicity, is that multi-modal emission probabilities can be represented by the histograms. The main drawback is that the histograms can not provide estimates of noisy observations that were unseen in the training process (in these cases the emission probabilities are always zero). To prevent this from happening, a rich set of

training data is required to train the discrete HMM classifier. Furthermore, the number of discrete symbols, or the k-parameter, in the codebook need to be carefully configured to fit the application domain.

7.2 HMM with Multi-variate Gaussian

In this teamwork recognizer the emission probabilities are modeled using the continuous Gaussian PDF. The advantage of using a continuous PDF is that unseen observations can be estimated naturally. The drawback is that only unimodal emission probabilities can be represented using the Gaussian PDF. As a result, teamwork activities that are identical by human perception, but physically different (such as U-turn to the right and left), need to be encoded in separate HMMs but with the same label in the recognizer.

The Gaussian PDF is introduced in Chapter 4. However, for completeness, the Gaussian PDF is presented again in Equation 7.1, where $\boldsymbol{\mu}$ is the mean vector, $\boldsymbol{\Sigma}$ is the covariance matrix and d is the dimensionality of the observation vector $\mathbf{v}$. The parameters, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, are estimated from training data using the maximum likelihood method [DHS00].

$$\beta_j(\mathbf{v}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} exp\left\{-\frac{1}{2}(\mathbf{v}-\boldsymbol{\mu})^t\boldsymbol{\Sigma}^{-1}(\mathbf{v}-\boldsymbol{\mu})\right\} \tag{7.1}$$

Another limitation of the Gaussian PDF is that the covariance matrix, $\boldsymbol{\Sigma}$, must be invertible to calculate probabilities from Equation 7.1. This becomes a problem in applications where training

data is limited and when the observations are high-dimensional. One way to solve this problem is to reduce the expressiveness of the Gaussian PDFs, for instance by representing them with diagonal covariance matrices only, which are guaranteed to be invertible.

7.3 HMM with Gaussian Mixture Model

The Gaussian Mixture Model (GMM) combines the advantages of the simple HMM and the Gaussian PDF approach. The use of a mixture of Gaussian components facilitates modeling of multimodal emission probabilities. Like the Gaussian PDF model, emission probabilities from observations not seen during training can be estimated due to the continuous nature of the GMM.

The GMM, which is described in Equation 7.2, can be defined as the weighted sum over its Gaussian components. The parameters of the GMM are $\mathbf{\Omega} = \{w_1, \mathbf{\Sigma_1}, \boldsymbol{\mu_1}, \ldots, w_C, \mathbf{\Sigma_C}, \boldsymbol{\mu_C}\}$, where C depict the number of mixture components the GMM consists of. Each component is associated with a weight, such that $0 < w_c \leq 1$ and $\sum_{c=1}^{C} w_c = 1$. Estimating $\mathbf{\Omega}$ from observations is performed using the expectation maximization (EM) algorithm [DLR77], which iteratively applies the maximum likelihood method to generate better estimates of the parameters.

$$\beta_j(\mathbf{v}) = \sum_{c=1}^{C} \frac{w_c}{(2\pi)^{d/2}|\mathbf{\Sigma_c}|^{1/2}} exp\left\{-\frac{1}{2}(\mathbf{v} - \boldsymbol{\mu_c})^t \mathbf{\Sigma_c}^{-1}(\mathbf{v} - \boldsymbol{\mu_c})\right\} \tag{7.2}$$

A disadvantage of using the GMM is that the parameter estimation process, which is used to determine $\mathbf{\Omega}$, is computationally expensive compared to the Gaussian PDF approach. Further-

more, the number of mixture components, C, needs to be specified manually for each HMM in the recognizer. In the special case when $C = 1$ the GMM is identical to the Gaussian PDF.

7.4 Experiments

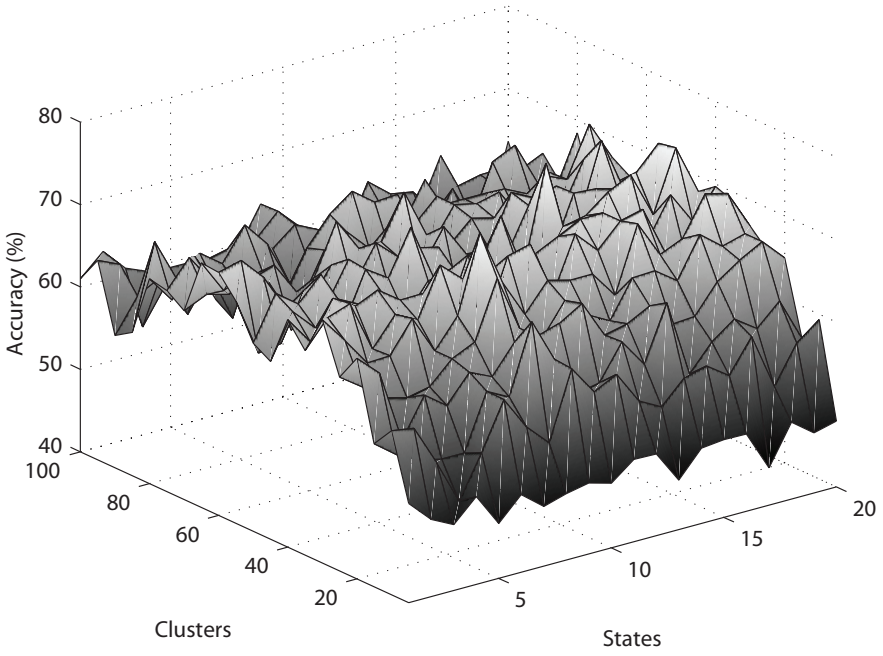
7.4.1 Experimental Setup

The experiments are performed using the same a warfare dataset that was presented in Chapter 6. The dataset, which consists of seven teamwork activities, is summarized in Table 6.1. The team-oriented feature functions were used to extract semantically rich features from the dataset. We have used the 10-fold cross-validation procedure with stratified sampling to generate performance measurements in all of the experiments.

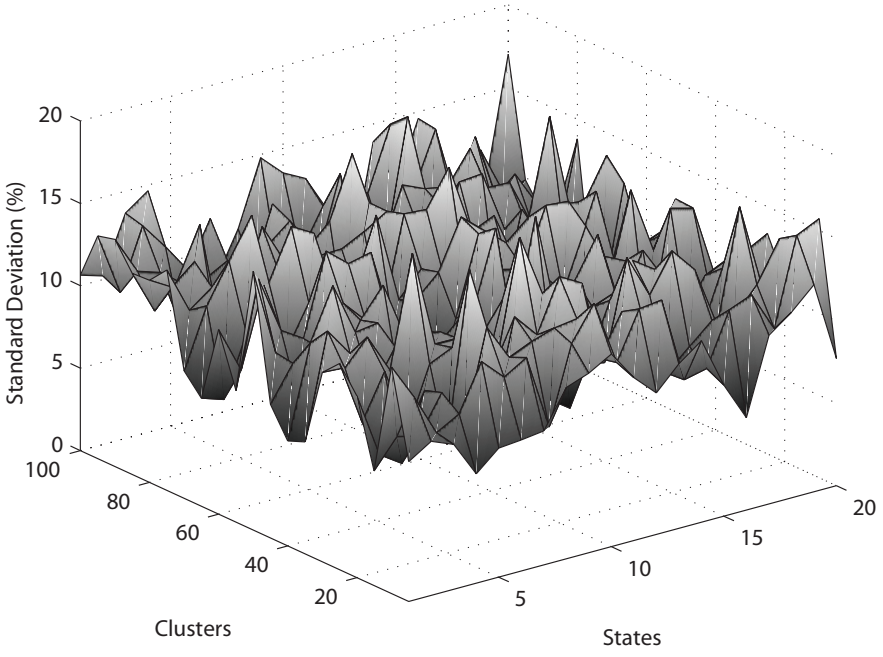
7.4.2 Results: Simple HMM

The simple HMM is parameterized with the number of hidden states and by the number of clusters/symbols, k. To find the optimal parameter configuration we have repetitively applied the 10-fold cross-validation procedure for each parameter configuration, where the number of hidden states is ranging from 2 to 20 and where k ranges from 5 to 100. The mean classification accuracy and standard deviation is shown in Figure 7.2.

The most influential parameter is the number of clusters, k, the vector quantization method is using. When few clusters are used the HMMs do not have enough discrete states to discriminate



(a) Accuracy.



(b) Standard deviation.

Figure 7.2: Mean accuracy and standard deviation of teamwork activity recognition using discrete HMMs acquired from vector quantization.

the teamwork activities. On the other hand, when too many clusters are used, the performance is degraded because the amount of training data is insufficient to accurately encode the teamwork activities in the HMMs.

The overall best performance, 75.05% with standard deviation ±14.89%, was found using 60 clusters and 18 hidden states. The confusion matrix in Table 7.2 shows that the teamwork recognizer with this parameter configuration struggles to classify the *bounding overwatch*, *team split* and *team merge* behaviors. For these behaviors class precision is only 53.33%, 68.75% and 53.33% respectively.

Table 7.2: Confusion matrix depicting teamwork activity recognition performance using discrete HMMs and team-oriented feature functions with 60 clusters and 18 hidden states. The mean accuracy is 75.05% with a standard deviation of ±14.89%.

			True							Precision
			a	b	c	d	e	f	g	(%)
Predicted	Traveling column	a	18	0	1	0	1	0	0	90.00
	Traveling line	b	1	24	1	1	0	2	4	72.73
	Traveling box	c	3	1	28	0	0	0	0	87.50
	Bounding overwatch	d	4	1	0	8	0	0	2	53.33
	Wedge	e	0	2	1	0	14	0	0	82.35
	Team split	f	1	1	1	0	1	11	1	68.75
	Team merge	g	2	1	1	0	1	2	8	53.33
	Recall (%)		62.07	80.00	84.85	88.89	82.35	73.33	53.33	

Occasionally, the discretization process described above filters out potentially good and important information describing the teamwork activities. By carefully selecting the parameters appropriately the negative effects of the discretization process can be minimized. However, the

poor classification accuracy of the *bounding overwatch*, *team split* and *team merge* behaviors is an indication that too much information is filtered out by the discretization process.

7.4.3 Results: HMM with Multi-variate Gaussian

The recognizer using Gaussian PDF is parameterized with the number of hidden states to use in each HMM. In Figure 7.3 we have repetitively applied the 10-fold cross-validation procedure for hidden states ranging from 1 to 5. The figure shows mean classification accuracy and standard deviation for the case where the covariance matrix is general and for the case when the covariance matrix is diagonal.

The best mean accuracy, 91.29% with a standard deviation of ±6.69%, was found with 3 hidden states using the general Gaussian PDF. One problem that we encountered in this case was that the amount of training data was insufficient for the *bounding overwatch*, *team split* and the *team merge* activities for number of hidden states larger than 3. Hence, the sudden drop in recognition accuracy.

In the diagonal case, where the features are assumed to be statistically independent of each other, the recognition accuracy increases with the number of hidden states. The best performance, 89.81% ±5.62%, was found using 5 hidden states.

Table 7.3 shows the confusion matrix for the number of hidden states that resulted in best mean accuracy in the general case. The experiments show that, if the number of hidden states in the HMMs are carefully chosen, classification accuracy is greatly improved compared to the simple HMM approach.

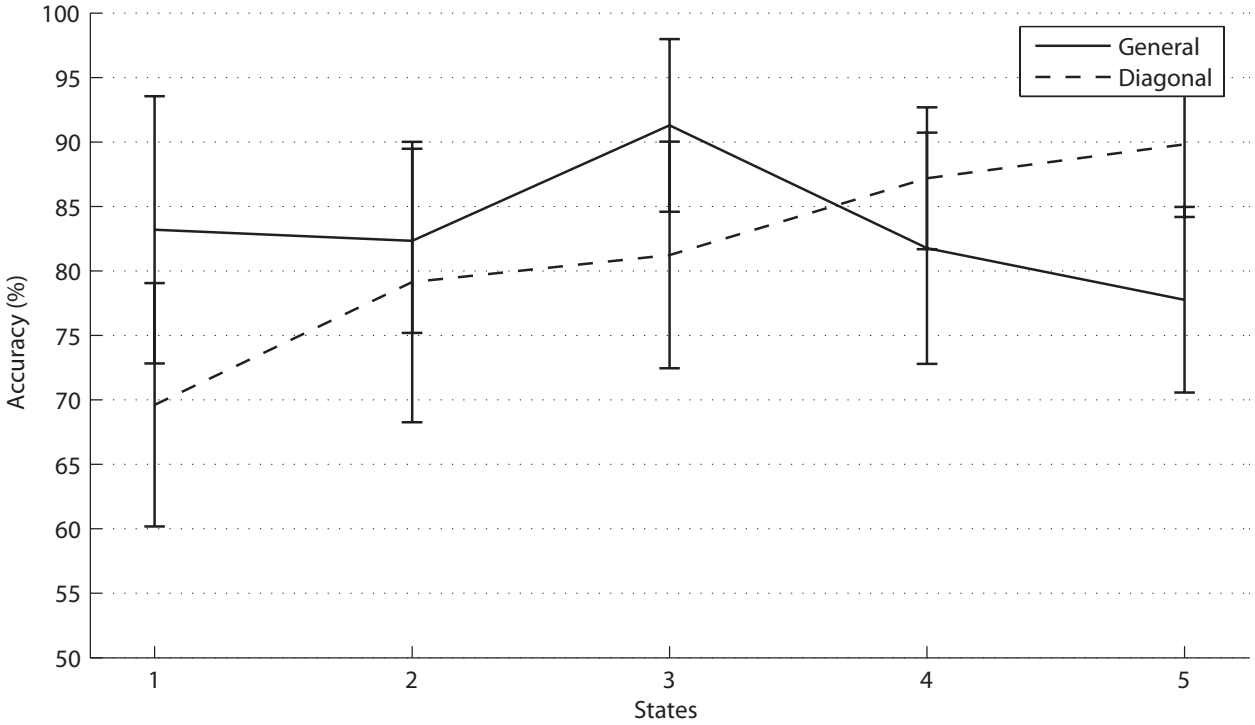


Figure 7.3: Mean accuracy and standard deviation of teamwork activity recognition using multi–variate Gaussian PDF to model emission probabilities in the HMMs.

Table 7.3: Confusion matrix depicting teamwork activity recognition performance using multi-variate Gaussian to model emission probabilities in HMMs with 3 hidden states. The mean accuracy is 91.29% with a standard deviation of ±6.69%.

			True							Precision
			a	b	c	d	e	f	g	(%)
Predicted	Traveling column	a	29	0	0	0	0	0	0	100.0
	Traveling line	b	0	27	2	0	0	0	0	93.10
	Traveling box	c	0	3	30	1	0	0	0	88.24
	Bounding overwatch	d	0	0	0	7	0	0	0	100.0
	Wedge	e	0	0	0	0	17	0	0	100.0
	Team split	f	0	0	0	0	0	14	4	77.78
	Team merge	g	0	0	1	1	0	1	11	78.57
	Recall (%)		100.0	90.00	90.91	77.78	100.0	93.33	73.33	

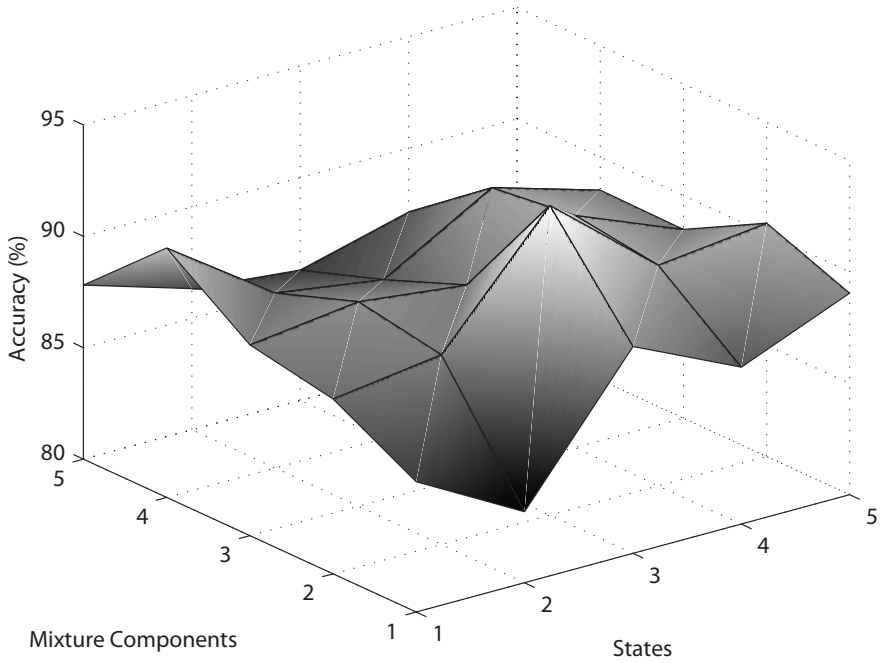
7.4.4 Results: HMM with Gaussian Mixture Model

Teamwork activity recognition using HMMs with the Gaussian mixture model is parameterized by the number of hidden states to use and by the number of components to use in each Gaussian mixture. In the results presented in Figure 7.4 the mixture components and the hidden states range from 1 to 5. The best overall accuracy, 93.90% with a standard deviation of ±4.68%, was found using 2 mixture components and 3 hidden states.

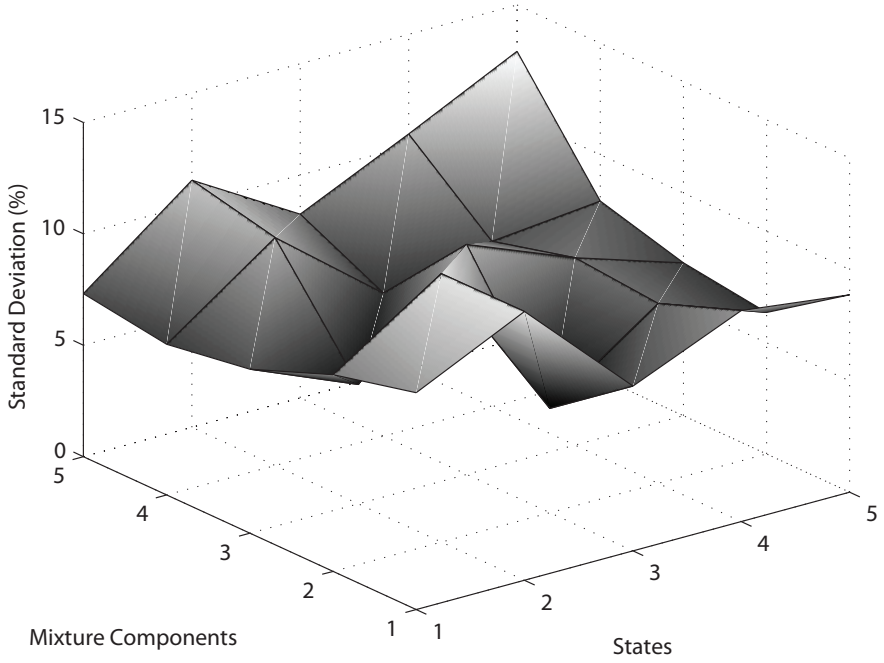
Table 7.4 show the confusion matrix from the best parameter configuration. The overall recognition accuracy using the GMM approach is better than any other recognizer. In Figure 7.5 the HMMs that were generated by the learning process are illustrated for each teamwork activity,

Table 7.4: Confusion matrix depicting teamwork activity recognition performance using the Gaussian mixture model to represent emission probabilities in HMMs with 2 mixture components and 3 hidden states. The mean accuracy is 93.90% with a standard deviation of ±4.68%.

			True							Precision
			a	b	c	d	e	f	g	(%)
Predicted	Traveling column	a	27	0	1	0	0	0	0	96.43
	Traveling line	b	0	29	0	0	0	0	0	100.00
	Traveling box	c	0	0	31	0	0	0	0	100.00
	Bounding overwatch	d	0	0	0	8	0	0	0	100.00
	Wedge	e	0	0	0	0	16	0	0	100.00
	Team split	f	2	0	1	1	1	14	1	70.00
	Team merge	g	0	1	0	0	0	1	14	87.50
	Recall (%)		93.10	96.67	93.94	88.89	94.12	93.33	93.33	

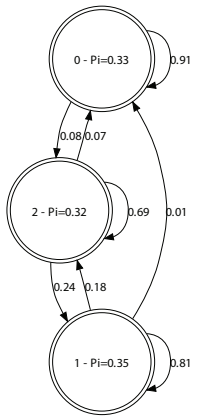


(a) Accuracy.

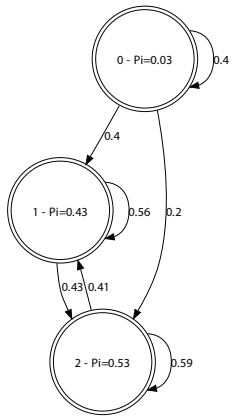


(b) Standard deviation.

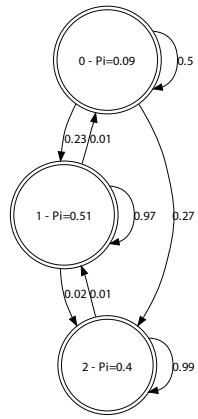
Figure 7.4: Mean accuracy and standard deviation of teamwork activity recognition using mixture of Gaussian PDF to model emission probabilities in the HMMs.



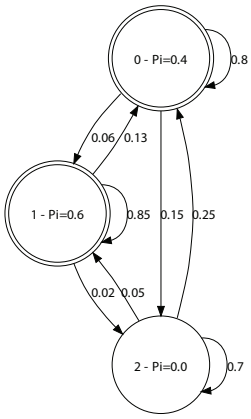
(a) Traveling column.



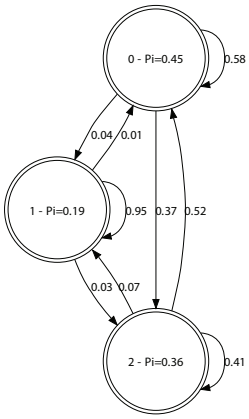
(b) Traveling line.



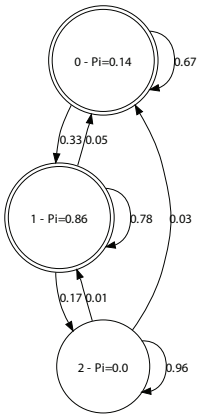
(c) Traveling box.



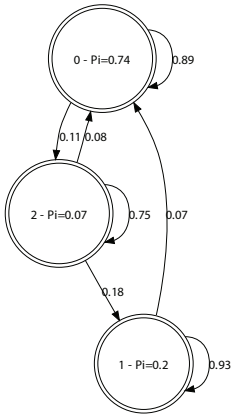
(d) Bounding overwatch.



(e) Wedge.



(f) Team split.



(g) Team merge.

Figure 7.5: The set of HMMs used in the GMM-based teamwork recognizer. Each HMM consists of 3 hidden states and the GMMs were generated using 2 mixture components. The initial probabilities, π, and the transition probabilities, α, are shown in the figures.

7.5 Conclusions

In this chapter we evaluated three implementation choices in HMM-based teamwork activity recognizers where the emission probabilities are modeled using histograms, Gaussian PDFs and Gaussian mixture models. Table 7.5 summarizes the performance of the implementation choices for all parameter configurations. The teamwork recognizer employing HMMs with the Gaussian mixture model produced the best accuracy. From the standard deviation column in the table it is clear that this system is also the least sensitive to changes in parameter configurations.

Table 7.5: Statistical summary of the implementation choices that have been evaluated. The table show mean, maximum and minimum and the standard deviation of accuracy over all parameter configurations.

Implementation Choice	Mean	Max	Min	Std. Dev.
Simple HMM	62.91%	75.05%	44.52%	5.44%
HMM with Gaussian	83.27%	91.28%	77.76%	4.94%
HMM with Gaussian Mixture Model	88.30%	93.90%	83.19%	2.17%

The simple HMM fared the worst in our evaluation. This implementation choice filtered out too much information in the discretization process, which lead to poor recognition performance.

We found that the Gaussian PDF approach with a general covariance matrix could not be trained for HMMs with more than three states for teamwork activities (*bounding overwatch*, *team split* and *team merge*) with limited amount of training data. In this case the recognition performance dropped significantly. To accommodate cases where more hidden states are required and when training data is limited the expressiveness of the Gaussian PDF can be reduced.

We conclude that the GMM approach, which is both continuous and can represent multi-modal distributions, is better than any of the other implementation choices. However, the GMM also have disadvantages:

- The complexity of estimating the GMM is higher compared to the Gaussian PDF.
- The number of Gaussian mixture components C must be carefully configured.

CHAPTER 8: CONCLUSIONS

In this thesis we presented contributions to the theory and practice of teamwork action recognition from the observation of embodied agents.

The first practical challenge of this work is the acquisition of a body of data. The Teamwork Scenario Editor (TSE) was implemented for visualization, identification and extraction of representative teamwork patterns. The tool significantly reduced the amount of knowledge engineering required to extract training (and validation) datasets for the teamwork recognizer.

Next, we described a baseline approach for recognizing teamwork activity in spatio-temporal position traces of embodied agents, based on a library of teamwork patterns encoded as Hidden Markov Models. While previous approaches relied on the acquisition of teamwork patterns by knowledge engineering, we developed an approach that acquires the patterns by learning from a small set of representative examples (segmented and labeled with the TSE). The resulting recognition system was found to match the performance of previous knowledge engineered systems with a recognition rate of around 82%, but it was found to be sensitive to noise and distortion caused by terrain variations.

To improve the accuracy of our teamwork recognizer we developed a set of feature functions to extract semantically rich features from the spatio-temporal position traces. The three classes of feature functions considered were (a) agent-oriented feature functions which discretize the position

and movement patterns of individual agents, (b) environment-oriented feature functions which characterize the interaction between the agent and its environment and (c) team-oriented feature functions which characterize the team as a whole. The feature functions were carefully selected to match the ways in which a human observer would understand and analyze the scene.

We improved the robustness of the teamwork recognizer by integrating a novel role recognition module into the recognition workflow. The role-based recognition approach addressed the challenge where the input to the recognizer is a fixed arrangement of the recorded features. For instance, an HMM which was trained with an input order {1, 2, 3, 4} can not recognize the same action if it appears encoded in the order {2, 3, 4, 1}. Experimental results showed that accuracy is improved, even in the presence of higher levels of noise and where the input is shuffled, when our role-based recognizer is used in combination with the team-oriented feature functions.

Many components of the teamwork recognizer are customizable though a variety of parameters. We studied the ways in which the choice of emission probability representations affects the accuracy and robustness of the teamwork recognizer. The results showed that, compared to the histogram and Gaussian PDF approaches, the Gaussian Mixture Model (GMM) yielded the best recognition accuracy.

8.1 Future Work

Significant future work is required in this domain. For instance, recognizing actions of larger teams of hundreds of agents is not feasible using the feature function set provided in this thesis as the dimensionality of the feature vector will grow out of control.

One fundamental issue in teamwork activity recognition, that was left out for future work in this thesis, is that of team-membership assignment. That is, prior to recognizing the teamwork activity of a team, the team itself must be recognized. The complexity of this task varies greatly. For instance, determining team-memberships in team-sports can be performed by distinguishing visual appearance. In surveillance applications, however, the task is much more complex. In this case, the physical actions or spatial relationships of the agents needs to be considered.

Below we present future work where preliminary studies have been performed. We have included these studies as appendices in this thesis.

- It is well known that real-world datasets are noisy and uncertain due to inaccurate sensors, occlusions and so on. In Appendix B the Kalman filter is proposed as a method to reduce noise and to fuse information from multiple sources with varying uncertainty in the context of the RoboCup Rescue simulation domain. In future work, we plan to improve teamwork recognition performance by reducing the amount of noise in our real-world dataset by use of the Kalman filter and/or particle filters.

- We have in Appendix C, [LEW05], developed a layered approach to learn strategies for individual agents in the Feed-Fight-Multiply environment. In our approach the strategies were automatically generated using genetic programming. Future work extends this implementation to learn teamwork strategies from observations. The teamwork activity recognition presented in this thesis will play an important role in calculating the fitness of the generated teamwork strategies.

APPENDIX A: HIDDEN MARKOV MODELS

In this chapter we introduce the algorithms that were used in this thesis to encode and classify teamwork activities using Hidden Markov Models. In Section A.1 the forward and backward evaluation algorithms are introduced. In Section A.2 the Viterbi decoding algorithm is presented and in Section A.3 we present the Baum-Welch and Segmental K-means learning algorithms.

A.1 Evaluation

The evaluation algorithm calculates the probability $P(\mathbf{V}^T \mid \boldsymbol{\theta})$ that an observation sequence $\mathbf{V}^T$ was generated given an HMM defined by $\boldsymbol{\theta} = \{\mathbf{A}, \mathbf{B}, \boldsymbol{\pi}\}$. The probability value can be calculated as follows:

$$P(\mathbf{V}^T \mid \boldsymbol{\theta}) = \sum P(\mathbf{V}^T|\boldsymbol{\omega}^T)P(\boldsymbol{\omega}^T) \text{ for all } \boldsymbol{\omega}^T \tag{A.1}$$

where $\boldsymbol{\omega}^T$ represents a sequence of hidden states with length T. The summation includes all possible combinations of $\boldsymbol{\omega}^T$. Note that, in the first order HMM, the first and second terms can be rewritten:

$$P(\mathbf{V}^T|\boldsymbol{\omega}^T) = \prod_{t=1}^{T} P(v_t|\omega_t) \tag{A.2}$$

$$P(\boldsymbol{\omega}^T) = \prod_{t=1}^{T} P(\omega_t|\omega_{t-1}) \tag{A.3}$$

Hence, Equation A.1 can be rewritten as:

$$P(\mathbf{V}^T \mid \boldsymbol{\theta}) = \sum \left(\prod_{t=1}^{T} P(v_t|\omega_t)P(\omega_t|\omega_{t-1}) \right) \text{ for all } \boldsymbol{\omega}^T \tag{A.4}$$

Because Equation A.4 relies only on observations from t and $t-1$, $P(\mathbf{V}^T \mid \boldsymbol{\theta})$ can be calculated recursively using the forward evaluation algorithm which is presented as pseudo-code in Listing A.1.

Listing A.1: The forward evaluation algorithm

```
//Set initial probabilities at t=0
k = V[t=0] //Initial observation
for(j=0; j<h; j++) //Loop internal states
{
    p[j] = B[j, k] * π[j]
}

for(t=1; t<T; t++) //Loop observation sequence
{
    k = V[t] //Observation
    for(j=0; j<h; j++) //Loop internal states
    {
        b = B[j, k]
        for(i=0; i<h; i++) //Loop internal states
        {
            sum = sum + p[i] * A[i, j]
        }
        pNext[j] = b * sum
    }
    p = pNext
}

//Calculate return probability
for(j=0; j<h; j++) //Loop internal states
{
    P = P + p[j]
}

return P
```

In the first for-loop a vector of forward probabilities **p** are initialized using the first observation vector in the input sequence $\mathbf{V^T}$ and the HMM's $\boldsymbol{\pi}$ and **B** parameters. In the nested for-loop the probabilities are updated at each time step using the HMM's **A** and **B** parameters. The forward algorithm outputs the summed probabilities in the **p** vector.

The backward algorithm calculates **p** in a similar manner, however, this algorithm initializes **p** using the last observation in the input sequence $\mathbf{V^T}$ and recursively updates the **p** in reverse. Listing A.2 shows a pseudo-code implementation of the backward evaluation algorithm.

Listing A.2: The backward evaluation algorithm

```
//Set initial backward probabilities
for(j=0; j<h; j++) //Loop internal states
{
    p[j] = 1
}

for(t=T-2; t>=0; t--) //Loop observation sequence
{
    k = V[t + 1] //Observation
    for(i=0; i<h; i++) //Loop internal states
    {
        for(j=0; j<h; j++) //Loop internal states
        {
            sum = sum + p[j] * A[i, j] * B[j, k]
        }
        pNext[j] = sum
    }
    p = pNext
}

//Calculate return probability
k = V[t=0] //Initial observation
for(j=0; j<h; j++) //Loop internal states
{
```

```
    P = P + p[j] * B[j, k] * π[j]
}

return P
```

A.2 Decoding

Decoding is performed to find the optimum hidden state sequence given an input sequence. The Viterbi algorithm (VA) [For73, Rab90], which is presented as pseudo-code in Listing A.3, can be used for this task. The algorithm can be separated into three steps:

- Initialize ψ and δ at t=0 (the first visible observation).
- Calculate ψ and δ for the remaining observations by finding the *argmax* and *max* values respectively.
- Backtrack to find the optimum hidden sequence.

Listing A.3: The Viterbi decoding algorithm

```
// Initialization
k = V[t=0]
for(i=0; i<h; i++)
{
    δ[t=0,i] = π[i] * B[i, k]
    ψ[t=0,i] = 0
}

// Calculate ψ and δ
for(t=1; t<T; t++) //Loop observation sequence
```

```
{
    for(j=0; j<h; j++)
    {
        δmax = −∞
        for(i=0; i<h; i++)
        {
            δTmp = δ[t−1, i] * A[i, j]
            if(δTmp > δmax)
            {
                δmax = δTmp
                ψmax = i
            }
        }
        δ[t, j] = δmax * B[j, k]
        ψ[t, j] = ψmax
    }
}

//Find ending sequence state
pmax = −∞
for(i=0; i<h; i++)
{
    pTmp = δ[T−1][i]
    if(pTmp > pmax)
    {
        pmax = pTmp
        sequence[T−1] = i
    }
}

//Backtrack to find valid hidden sequence
for(t=T−2; t>=0; t−−) //Loop observation sequence
{
    sequence[t] = ψ[t+1, sequence[t+1]]
}

return sequence
```

A.3 Learning

The learning problem consists of estimating the HMM parameters $\mathbf{A}$, $\mathbf{B}$ and $\boldsymbol{\pi}$. In this section we discuss two well known learning algorithms:

- Baum-Welch (Forward-Backward) algorithm [Rab90, Bil97, DHS00].
- Segmental K-means algorithm [JR90].

A.3.1 Baum-Welch

The *Baum-Welch* learning algorithm adjusts the parameters in $\boldsymbol{\theta}$ such that $P(\mathbf{V}^T|\boldsymbol{\theta})$, which can be calculated using the forward and backward evaluation algorithms, is maximized. To formally describe the Baum-Welch algorithm let us define $\xi_t(i, j)$:

$$\xi_t(i, j) = P(i_t = i, i_{t+1} = j|\mathbf{V}^T, \boldsymbol{\theta}) \tag{A.5}$$

which is the probability of being in state i at time t and making a transition to state j at time $t + 1$ given the model $\boldsymbol{\theta}$ and observation sequence $\mathbf{V}^T$. $\xi_t(i, j)$ can be described using forward and backward evaluation algorithms:

$$\xi_t(i, j) = \frac{\alpha_t(i)a_{ij}b_j(v_{t+1})\beta_{t+1}(j)}{P(\mathbf{V}^T|\boldsymbol{\theta})} = \frac{\alpha_t(i)a_{ij}b_j(v_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(v_{t+1})\beta_{t+1}(j)} \tag{A.6}$$

where $\alpha_t(i)$ is the forward probability that the HMM is in hidden state i at time t. $\beta_{t+1}(j)$ is the backward probability that the HMM is in hidden state j at time $t + 1$. $a_{ij}b_j(v_{t+1})$ is the probability

of transitioning from state i to state j and observing v_{t+1} at state j. Let us now define $\gamma_t(i)$:

$$\gamma_t(i) = P(i_t = i|\mathbf{V}^T, \boldsymbol{\theta}) \tag{A.7}$$

which is the probability of being in state i at time t given the observation sequence $\mathbf{V}^T$ and model $\boldsymbol{\theta}$. $\gamma_t(i)$ relates to $\xi_t(i, j)$ as follows:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i, j) = \text{ expected number of times that state } i \text{ has been visited} \tag{A.8}$$

which can be interpreted as the expected number of times that state i has been visited. Similarly, by summing $\gamma_t(i)$ and $\xi_t(i, j)$ over time the expected number of transitions from state i and the expected number of transitions from state i to state j can be calculated:

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{ expected number of transitions from state } i \tag{A.9}$$

$$\sum_{t=1}^{T-1} \xi_t(i, j) = \text{ expected number of transitions from state } i \text{ to state } j \tag{A.10}$$

Using these formulas an improved model of $\boldsymbol{\theta}$ can be calculated iteratively as follows:

$$\boldsymbol{\pi} = \text{ expected number of times that state } i \text{ has been visited at time } (t = 1) = \gamma_t(i) \tag{A.11}$$

$$\mathbf{A} = \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{A.12}$$

$$\mathbf{B} = \frac{\text{expected number of times in state } j \text{ and observing visible state } v}{\text{expected number of times in state } j} = \frac{\sum_{t=1}^{T} \gamma_t(j), v_t = v}{\sum_{t=1}^{T} \gamma_t(j)} \tag{A.13}$$

A.3.2 Segmental K-means

The *Segmental K-means* learning algorithm maximizes the joint probability, $P(\mathbf{V}^T, \mathbf{I}|\boldsymbol{\theta})$, of the input sequence $\mathbf{V}^T$ and its optimum hidden state sequence $\mathbf{I}$ given the HMM model $\boldsymbol{\theta}$. The optimum hidden state sequence $\mathbf{I}$ can be calculated using the Viterbi decoding algorithm presented in Section A.2. The Segmental K-means algorithm can be separated into the following steps:

1. Apply the K-means clustering algorithm to the observations in the input sequences to generate k cluster centers which depicts the initial representatives of the hidden states $\boldsymbol{\omega}$. Assign a cluster label to all observations in the input sequences.

2. Learn $\boldsymbol{\pi}$ by calculating the number of occurrences that the first observation in each input sequence belong to each cluster.

3. Learn $\mathbf{A}$ by calculating the number of occurrences where a transition is performed from hidden state i to j over all observation sequences.

4. Learn $\mathbf{B}$ by fitting a probability density function to each state (cluster).

5. Optimize the clusters by calculating the optimum hidden sequence **I** for all input sequences given the model parameters $\boldsymbol{\theta}$ generated from steps 2-4. Reassign the observation cluster assignments following **I**.

6. Perform step 2-5 until converged. That is, there are no reassignments made in step 5.

Listing A.4 shows the pseudo-code of the algorithm. Note that *clusters*($\mathbf{V^T}[\mathbf{0}]$) represents a mapping from a visible state, in this case the first visible state in an observational sequence, to a hidden state. Also, *clusters*[*j*] represents all visible observations in cluster (hidden state) *j*.

Listing A.4: The Segmental K-means learning algorithm

```
//Find internal state representations using K-Means algorithm
clusters = KMeans(observations, h)

//Learn π
for(i=0; i<h; i++) //Initialize π[i]
{
   π[i] = 0
}

for each V^T in sequences //Calculate π[i]
{
   ωFirst = clusters(V^T[0])
   π[ωFirst] = π[ωFirst] + 1
}

for(i=0; i<h; i++) //Normalize π[i]
{
   π[i] = π[i] / (Number of visible sequences)
}

//Learn A
for(i=0; i<h; i++) //Initialize A[i, j]
```

```
{
    for(j=0; j<h; j++)
    {
        A[i, j] = 0
    }
}

for each V^T in sequences //Calculate A[i, j]
{
    for(t=1; t<T; t++) //Loop observation sequence
    {
        ωFrom = clusters(V^T[t-1])
        ωTo = clusters(V^T[t])
        A[ωFrom, ωTo] = A[ωFrom, ωTo] + 1
    }
}

for(i=0; i<h; i++) //Normalize A[i, j]
{
    for(j=0; j<h; j++)
    {
        sum = sum + A[i, j]
    }

    for(j=0; j<h; j++)
    {
        A[i, j] = A[i, j] / sum
    }
}

//Learn B
for(j=0; j<h; j++) //Normalize A[i, j]
{
    B[j] = Fit(clusters[j])
}

//Update clusters
for each V^T in sequences
{
    ω = Viterbi(V^T, π, A, B)
    for(t=0; t<T; t++)
```

```
    {
        ωState = clusters(V^T[t])
        if(ωState != ω[t])
        {
            Remove V^T[t] from cluster[ωState]
            Add V^T[t] to cluster[ω[t]]
        }
    }
}
```

APPENDIX B: A ROBUST METHOD FOR ESTIMATING NOISY MEASUREMENTS APPLIED TO DISASTER RESPONSE OPERATIONS

To efficiently coordinate rescue operations, decision makers need accurate, unbiased and up-to-date information. However, the incoming reports in a disaster scenario are of varying quality. The difficulty of making accurate observations, the stress and affective distress of the reporting agents can introduce inaccuracies and bias in the observations. Breakdowns in the communication system can delay the reporting of the observations.

We propose a method to integrate the noisy reports into a robust estimation of measures, such as the health level of a wounded civilian or the level of damage in a building on fire. Our method uses a Kalman filter to adaptively filter out noise and biased observations. We test our method in the context of a disaster scenario simulated in the RoboCup Rescue simulator. We show that the estimation process is robust and results in more accurate information which ultimately leads to improved rescue response.

B.1 Introduction

Natural disasters such as the enormous tsunami that devastated large parts of Malaysia and Thailand or hurricane Katrina that broke the levees protecting the city of New Orleans cause a lot of suffering, material damage and loss of human lives. Fortunately, disaster response operations can greatly reduce the impact of natural disasters. Efficient disaster response requires the coordination of several organizations: police, firefighters, ambulances, civil and religious organizations and individual volunteers [BKT06]. In the near future, disaster response will be partially performed with

teams of rescue robots and robots will autonomously conduct important disaster response tasks such as search-and-rescue.

Efficient coordination (whether centralized or distributed) requires accurate, unbiased and timely information about the state of objectives such as the number and status of victims, the amount of damage and prognosis of the objectives such as buildings and bridges, and the state of roads and communication networks. This global picture needs to be assembled from a series of observations reported by individual agents (such as police, firefighters or medics) or by civilians.

Even with a multitude of observations, forming an accurate, unbiased and up-to-date estimate is a challenge. Disasters are high stress situations, which affect the objectivity of the observers. Different members of the rescue teams have different levels of training and experience, or they may possess measurement and diagnostic tools of varying levels of sophistication. This affects the accuracy of the reports. The emotional state of the reporting subjects might introduce a bias in the observation. Occasionally downright errors of identification, pranks or malicious agents can introduce outlier observations. Finally, reports may be delayed at various levels due to communication failures, delays in processing due to the queuing of incoming reports, the active rescue tasks taking precedence from the communication tasks and so on.

How can we extract the most possible information from this avalanche of observations of various quality? Obviously, simple methods such as retaining the latest reported observation would not work, because that observation might be of low quality. An approach which would keep a

running average of the last several reported observations would ignore the temporal trends; also, outlier observations such as a mislabeled observation can significantly affect the running average.

In this study we propose an approach which allows us to extract the best possible estimate from a series of noisy observations. We use a Kalman filter to perform robust estimation and prediction of values. We test our system in the context of a modified RoboCup Rescue simulator, and show that significant improvements in the estimation can be obtained.

B.2 Related Work

Noisy and incomplete communication has deteriorating effects on the overall performance of teams in Multi-Agent System (MAS) applications. Farinelli and Scerri [FS05] developed a novel method to improve the efficiency in MAS tasks, such as RoboCup Rescue, by reducing the total number of unnecessary actions taken by robot teams. The method is distributed and allows rescue robots to balance the benefits and costs of acting based on potentially noisy and erroneous sensor readings. A Bayesian filter is employed to calculate confidence values of contradictory sensor readings so that decision whether halt or continue to act can be performed.

In earlier work by Farinelli et. al. [FGI03] components for information fusion, coordination and planning are incorporated into a Cognitive Agent Development framework. Evaluation criteria in the context of RoboCup Rescue for performance measures such as efficiency, reliability and robustness are developed.

Several systems to collaboratively track objects have been developed in the context of the RoboCup Soccer competition. Promising work by Dietl et. al. [DGN01] employed Kalman filters to track player positions and soccer ball position by fusing sensor information from multiple robotic players. Similar work was performed by Stroupe et. al. in [SMB01].

The Kalman filter [Kal60] is a general data processing algorithm and it has been employed in numerous other applications. For instance, these filters have been applied in weather forecasting where multiple models are assimilated to a single and more accurate model [Mac03]. Kalman filters are used to remove drift from inaccurate sensors in autopilot implementations of airplanes and boats [KM90]. Similarly, the Kalman filter is used to solve the simultaneous localization and mapping (SLAM) problem [DNC01] as well as in tracking algorithms [Fox96].

Interesting work by Bacastow et. al. [BCJ03] show that genetic algorithms can be employed to generate agents that learn to tolerate noisy communication. Results show that, in a predator-prey environment, collaborating predators trained without communication noise are outperformed by predators trained with noise.

B.3 Mathematical Foundations of Robust Estimation with Kalman Filters

Our estimation technique is based on the Kalman filter. The Kalman filter is a general data processing algorithm frequently used to remove noise in processes described by linear systems or differential equations [Mac03, Fox96, KM90, DNC01]. The Kalman filter is an optimal recursive

data processing algorithm which statistically minimizes the error of all available sensor data given prior knowledge about the sensor devices [Kal60, May79].

Using a Kalman filter, our estimation process is capable of:

- Fusing measurements from multiple distinct sources.
- Managing measurement uncertainties which are the result of sensor characteristics.
- Controlling the degree in which the estimation process adapts to incoming measurements.
- Recursively calculate new estimates to facilitate real-time application.

B.3.1 The Kalman Filter

The Kalman filter is used to estimate the state in processes that are described by the state equation

$$\mathbf{x_t} = \mathbf{A}\mathbf{x_{t-1}} + \mathbf{B}\mathbf{u_t} + \mathbf{w_{t-1}} \tag{B.1}$$

and measurement equation

$$\mathbf{y_t} = \mathbf{C}\mathbf{x_t} + \mathbf{z_t} \tag{B.2}$$

In Equation B.1, the state is represented by the vector $\mathbf{x_t}$. The matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and the input control vector, $\mathbf{u_t}$, describe the dynamics of the process. Noise is added to the state and measurement equations in the form of process noise, $\mathbf{w_{t-1}}$, and measurement noise, $\mathbf{z_t}$. In our estimation

process the noise vectors $\mathbf{w_{t-1}}$ and $\mathbf{z_t}$ are the only parameters that need to be carefully tuned to achieve good estimation performance.

The Kalman filter algorithm, which is used to continuously update the state estimate, is divided into the *prediction phase* and the *correction phase*. The prediction phase is responsible for predicting the state estimate, $\hat{\mathbf{x}}_\mathbf{t}^-$, (Equation B.3) and its associated error covariance, $\mathbf{P}_\mathbf{t}^-$, (Equation B.4).

$$\hat{\mathbf{x}}_\mathbf{t}^- = \mathbf{A}\hat{\mathbf{x}}_\mathbf{t-1} + \mathbf{B}\mathbf{u}_\mathbf{t} \tag{B.3}$$

$$\mathbf{P}_\mathbf{t}^- = \mathbf{A}\mathbf{P}_\mathbf{t-1}\mathbf{A}^\mathbf{T} + \mathbf{W} \tag{B.4}$$

The second phase is responsible for producing new state estimates using results from the previous phase as inputs. Equation B.5 calculates the weights or Kalman gain, $\mathbf{K_t}$, that are to be applied to the new state estimate, $\hat{\mathbf{x}}_\mathbf{t}$, and its error covariance, $\mathbf{P_t}$. State estimate and error covariance are updated according to Equation B.6 and Equation B.7. Here, $\mathbf{W}$ and $\mathbf{Z}$ represent the covariance of the process noise and measurement noise respectively.

$$\mathbf{K_t} = \mathbf{P}_\mathbf{t}^-\mathbf{C}^\mathbf{T}(\mathbf{C}\mathbf{P}_\mathbf{t}^-\mathbf{C}^\mathbf{T} + \mathbf{Z})^{-1} \tag{B.5}$$

$$\hat{\mathbf{x}}_\mathbf{t} = \hat{\mathbf{x}}_\mathbf{t}^- + \mathbf{K_t}(\mathbf{y_t} - \mathbf{C}\hat{\mathbf{x}}_\mathbf{t}^-) \tag{B.6}$$

$$\mathbf{P_t} = (\mathbf{I} - \mathbf{K_t}\mathbf{C})\mathbf{P_t^-} \tag{B.7}$$

Note that the Kalman filter equations are recursive and therefore does not require us to store the history of measurements. Also, if required by the estimation application the Kalman filter equations can be used as a multiple input and multiple output state estimator.

B.3.2 Modeling Uncertainty

A closer investigation of the measurement noise covariance, $\mathbf{Z}$, in the Kalman filter equations above reveals that it directly models the uncertainty of a sensor measurement. For instance, when $\mathbf{Z}$ is large, the Kalman gain is small. This means that the measurement will have less influence on the updated estimate. On the contrary if $\mathbf{Z}$ is small, the gain will be large and the measurement will influence the updated estimation significantly.

Calculating the noise in sensors or other systems that produce noisy information in real-world applications is usually performed through an experimental process. For instance, it is possible to calculate the noise of sensor by acquiring multiple measurements from experiments.

B.3.3 The Adaptiveness of the Estimation

From Equation B.4 we can see that process noise, **W**, is added to the predicted error noise. The process noise is a very important parameter as it determines how adaptive, or responsive, the estimation process will be to new measurements.

One can also think about the process noise as a parameter that determines the amount of estimation history to include in the new estimate. If the process noise is too small the estimation process will be unable to adapt to new sensor measurements. In this case, too much history is included in the estimation process. If on the other hand the process noise is too large the estimation process will quickly adapt to new measurements resulting in estimates that are no better than the noisy measurements. Both cases should be avoided as they result in poor estimates.

B.4 Implementation

B.4.1 The RoboCup Rescue Simulation Environment

The RoboCup Rescue simulation environment [KTN99] simulates urban disaster response scenarios. The environment contains three types of agents: police-force, ambulance teams and fire-brigade. The simulated rescue robots work together in a city devastated by an earthquake where the ultimate objective is to rescue as many injured civilians as possible. The rescue robots need to perform search-and-rescue missions, put out fires in buildings and remove blockages from the streets in the city.

The rescue robots are specialized: only the fire-brigade can put out fires, only the ambulance team can rescue civilians and only the police-force can remove blockages from the streets. All rescue robots can, however, sense the environment to detect fires and injured civilians in need of rescue. The robots are equipped with sensors that can determine health, damage rate and buriedness of an injured civilian. The damage rate is a measure of how much health a civilian is loosing every simulation cycle and thus allows us to make a prognosis about the state of the civilian in the future. Buriedness measures the effort required to rescue the civilian from a burning or collapsed building.

The rescue robots can directly send reports about the status of civilians, fires and blockages to each other if they are of the same type. Reports which need to be communicated with rescue robots of different type can by processed and forwarded by communication centers.

B.4.2 Estimating the Life-Expectancy of Civilians

Measuring the life-expectancy of an injured civilian includes determining the severity of one or more injuries. In the case of a rescue robot the life-expectancy measurement need to be conducted through sensors or some kind of reasoning system. If the injured civilian is conscious, then for instance, a rule-based or expert system can be used to measure the life-expectancy by interviewing the civilian. There is always a measure of uncertainty and potentially bias in the estimate of the severity of the injuries. In general, we can expect that a medic of the ambulance team provides a better estimate than a policeman or a firefighter.

In this section we apply the estimation process to estimate the life-expectancy of injured civilians in the RoboCup Rescue simulation. Such life-expectancy estimations can be used to prioritize the tasks of ambulance teams in the RoboCup Rescue simulation.

Life-expectancy is derived from reports, generated by the rescue robots, which consist of health, damage rate and buriedness measurements of injured civilians. We apply the estimation process to each measurement individually, therefore, the state vector consists of one variable only.

Our health estimation process is designed based on the simple fact that civilian health, H, is related to the damage rate, $\frac{d}{dt}H$, through the following equation:

$$H_t = H_{t-1} + \Delta T \cdot \frac{d}{dt} H_t \tag{B.8}$$

From Equation B.1, matrices **A** and **C** are constants that we set to 1. Matrix **B** is the elapsed time, ΔT, between two measurement occurrences. The input control vector, $\mathbf{u_t}$ is assigned to the current damage rate estimate, $\frac{d}{dt}H_t$.

Similarly, the estimate process is applied to the damage rate and buriedness measurements. Here, however, since we do not have any control input it is set to 0.

B.5 Experimental Results

B.5.1 Experimental Setup

We evaluate the estimation methodology by applying it to civilian injury reports acquired from multiple ambulance, fire-brigade and police-force teams in a typical RoboCup Rescue simulation run. The ambulance communication center receives reports and continuously executes the life-expectancy estimation process as illustrated in Figure B.1.

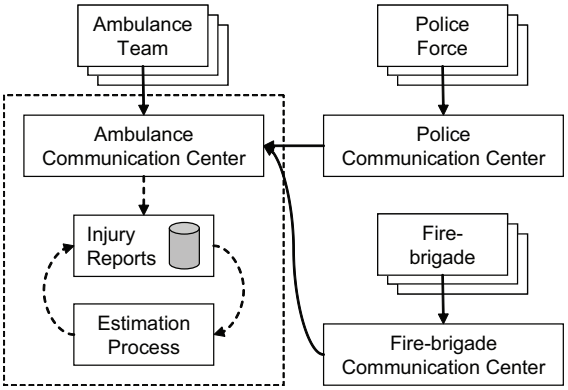


Figure B.1: The communication flow of injury reports for ambulance, fire-brigade and police force. The estimation process is maintained by the ambulance communication center and it continuously estimates civilian life-expectancy based on incoming injury reports.

We employed our *GoldenKnight* rescue team to gather data from the RoboCup Rescue simulation. In our experiments the *GoldenKnight* team consisted of 6 ambulance agents, 10 fire-brigade agents and 8 police-force agents. The simulator was configured to use the standard Kobe map populated with 72 civilians. Simulation time was set to 300 cycles.

The RoboCup Rescue simulator in its default mode assumes that the reports are always correct (although rounded to the nearest 10%). To make the scenario more realistic, we modified the setup such that we add a controlled amount of Gaussian noise to the reports. Our setup also allows us to tailor the noise characteristics depending on the source of the observations.

B.5.2 Estimating the Life-Expectancy of a Single Civilian

We have conducted three separate experiments using reports concerning a single injured civilian. *Experiment 1* uses reports which are using the same amount of noise independently of the observer. *Experiment 2*, assumes that the medics and firefighters can assess the health of the civilian better than the police-force, and thus applies a differentiated amount of noise. The measurement noise parameter of the estimation process is adjusted accordingly. Finally, in *Experiment 3* we apply the estimation process to reports that are free of noise. In all experiments, the ambulance communication center received a total of 44 reports for this civilian.

Experiment 1

In this experiment all the injury reports have a Gaussian noise with a standard deviation of 750 for the health value, 10 for the damage and 10 for buriedness. One way to measure how good our estimation process is to compare it against a naive, history-less estimation strategy, which assumes that the latest reported observation is the current best estimate.

The results of the experiment are shown in Figure B.2. The top three graphs show real value, the naive and the robust estimate for civilian health, damage and buriedness respectively. The bottom three graphs show the absolute error of the naive and robust estimation values. Reports from firefighter, ambulance and police are indicated by diamond, square and triangle markers respectively.

The statistical properties of the estimation error are shown in Table B.1 for the naive estimation and in Table B.2 for the robust estimation. We find a very significant reduction in all the statistical measures of the estimation error.

Experiment 2

One of the benefits of using our estimation process is that reports with different noise or uncertainties can easily be incorporated into the estimation process by adjusting the measurement noise parameter. As a result, the estimation process can be configured to assign less importance to reports we know are uncertain.

In this experiment we assume that the police-force is less qualified than medics and firefighters to evaluate the severity of an injury. We model this by adding a Gaussian noise with a standard deviation of 1500 for the health, 20 for damage and 20 for buriedness for the reports originating from the police, while maintaining the values of 750, 10 and 10 respectively for the reports originating from medics and firefighters. Also, we incorporate this knowledge into the estimation process by adjusting the measurement noise parameter accordingly.

From Figure B.3 we can visually determine that the robust estimation process successfully applies less weight to reports from police force. As an example, the robust estimation error of health is not significantly increased by the first two spikes that originate from the police at time 0-25. On the other hand, the robust estimation error is increased for spikes that are generated by firefighters as observed around time 150-200.

As expected, the statistical summary in Table B.3 and Table B.4 reveals that the absolute error is significantly smaller for the robust estimation.

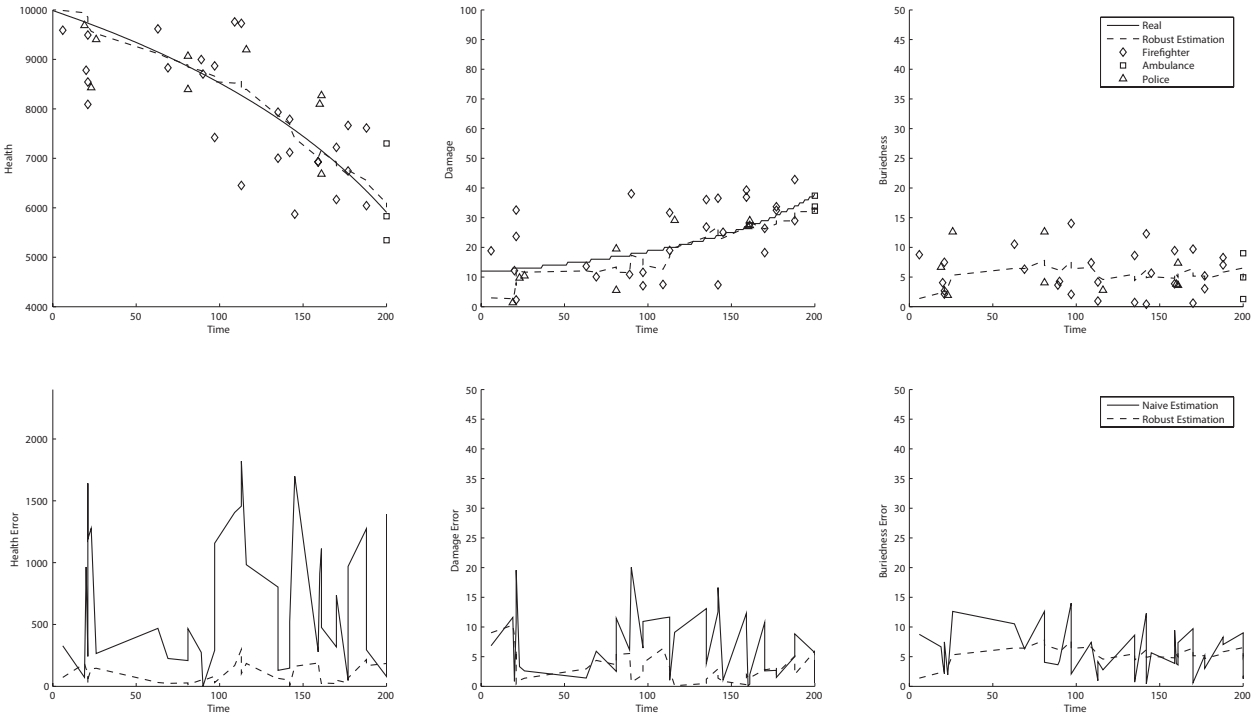


Figure B.2: Measurements on *Experiment 1*. The top graphs show real values and the naive and robust estimations for the health (left), damage (center) and buriedness (right) of the civilian. The bottom three graphs show the absolute error of naive and robust estimation.

Table B.1: Statistical properties of the absolute error for the naive estimation in *Experiment 1*.

Measurement	Mean	Std. dev.	Max	Min	Conf. int. 95%
Health	686.8	534.3	1820.9	0.4	±157.9
Damage	7.0	5.3	20.0	0.04	±1.6
Buriedness	6.0	3.7	14.0	0.4	±1.1

Table B.2: Statistical properties of the absolute error for the robust estimation in *Experiment 1*.

Measurement	Mean	Std. dev.	Max	Min	Conf. Int. 95%
Health	103.4	75.0	298.3	4.7	±22.2
Damage	3.2	2.4	10.3	0.1	±0.7
Buriedness	5.3	1.4	7.8	1.4	±0.4

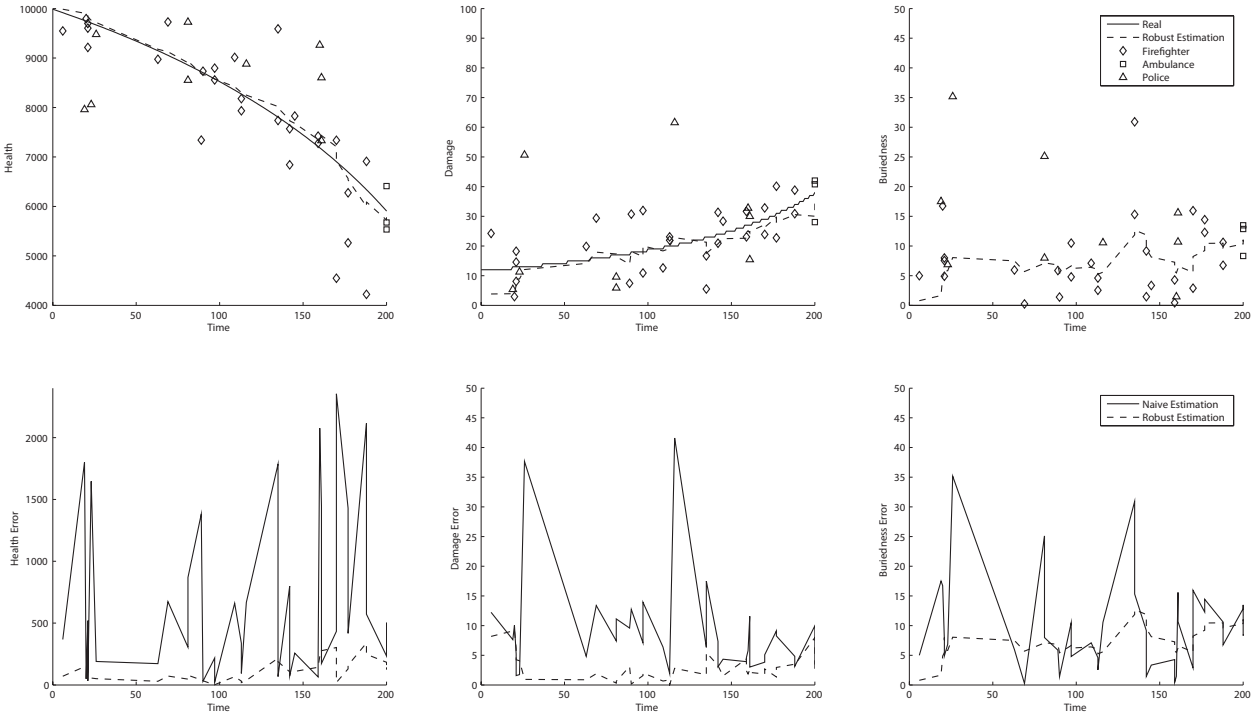


Figure B.3: Measurements on *Experiment 2*. The top graphs show real values and the naive and robust estimations for the health (left), damage (center) and buriedness (right) of the civilian. The bottom three graphs show the absolute error of naive and robust estimation.

Table B.3: Statistical properties of the absolute error for the naive estimation in *Experiment 2*.

Measurement	Mean	Std. dev.	Max	Min	Conf. Int. 95%
Health	639.2	657.2	2354.2	6.3	±194.2
Damage	8.5	8.1	41.6	1.3	±2.4
Buriedness	9.8	7.6	35.2	0.2	±2.2

Table B.4: Statistical properties of the absolute error for the robust estimation in *Experiment 2*.

Measurement	Mean	Std. dev.	Max	Min	Conf. Int. 95%
Health	112.8	87.0	339.0	0.3	±25.7
Damage	3.1	2.4	9.3	0.02	±0.7
Buriedness	7.7	2.7	12.7	0.8	±0.8

Experiment 3

Finally, we apply our estimation process to reports generated without Gaussian noise. This is the original configuration in which RoboCup Rescue competitions have been conducted in the past. Note that, in this simulation, the health and damage can only be sensed by the rescue robots in increments of 1000 and 10 respectively. As a result, there is a round-off error in the reports.

In Figure B.4 we can see that the absolute error using the robust method is smaller than the naive error. Hence, even in the absence of noise the robust estimation process is outperforming the naive approach. Statistical results are presented in Table B.5 and Table B.6.

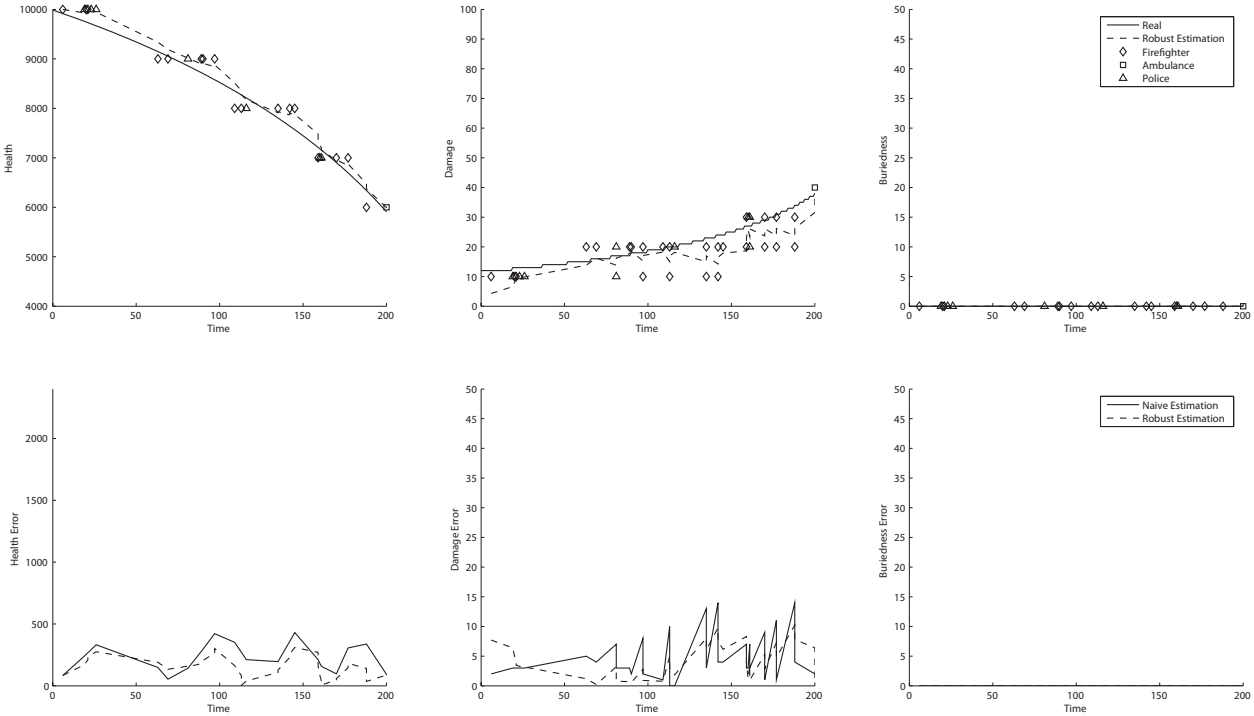


Figure B.4: Measurements on *Experiment 3*. The top graphs show real values and the naive and robust estimations for the health (left), damage (center) and buriedness (right) of the civilian. The bottom three graphs show the absolute error of naive and robust estimation.

Table B.5: Statistical properties of the absolute error for the naive estimation in *Experiment 3*.

Measurement	Mean	Std. dev.	Max	Min	Conf. Int. 95%
Health	228.2	100.5	431.0	54.0	±29.7
Damage	4.1	3.6	14.0	0.0	±1.1
Buriedness	0.0	0.0	0.0	0.0	±0.0

Table B.6: Statistical properties of the absolute error for the robust estimation in *Experiment 3*.

Measurement	Mean	Std. dev.	Max	Min	Conf. Int. 95%
Health	150.1	80.0	310.0	4.1	±23.6
Damage	3.7	2.9	10.2	0.0	±0.9
Buriedness	0.0	0.0	0.0	0.0	±0.0

B.5.3 Estimation Process Applied to All Civilians

In this section we present the absolute error of robust and naive estimation applied to all civilians throughout one simulation. Figure B.5(a) and Figure B.5(b) show box plots of the absolute error where Gaussian noise was added equally, as in *Experiment 1*, and where more Gaussian noise was added to the police-force, as in *Experiment 2*.

It is noticeable that the standard deviation, from the single civilian experiments, as well as the inner quartile range using robust estimation of health, damage and buriedness in Figure B.5 is consistently smaller compared in the to the naive approach. This is an indication that measurements from our estimation process not only are on average closer to the real values but also more robust and reliable.

B.6 Conclusions

In this study we presented an approach to acquire robust estimations of critical information in disaster rescue scenarios by applying a Kalman filter to noisy reports coming from agents in the field. We implemented our approach in the context of a modified RoboCup Rescue simulation environment. A series of experiments validate our approach. We found that our estimate methodology gives significantly better results than a naive, history-less, approach. Future work will involve more fine grain characterization of the information sources, cross-validation using correlated measurements as well as investigations into non-linear filters.

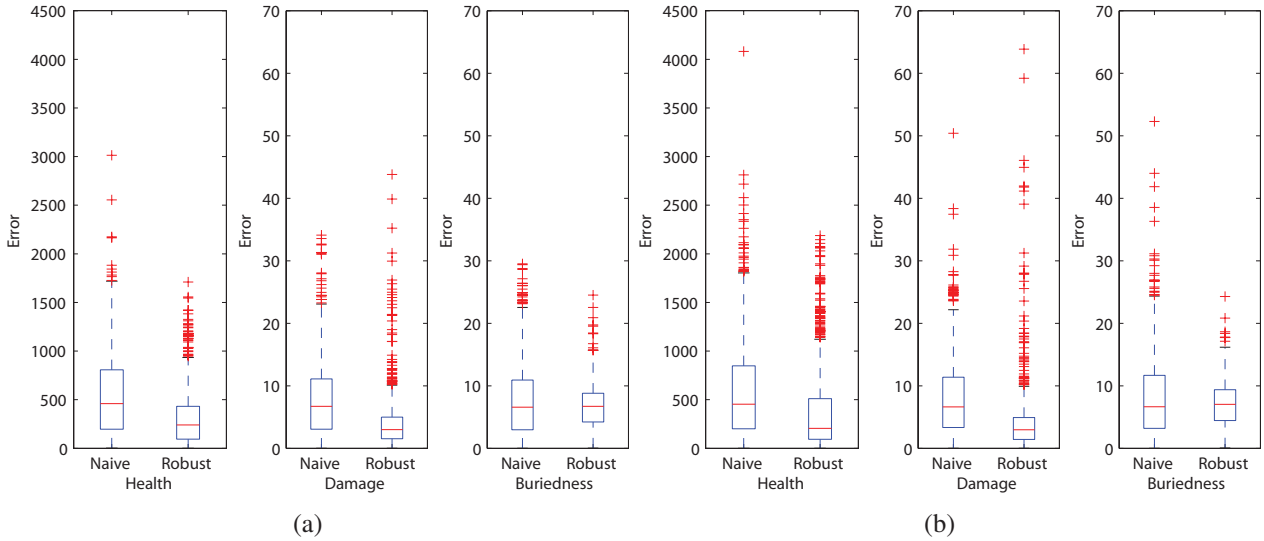


Figure B.5: Absolute error of robust and naive estimation applied to all civilians. Figure B.5(a) show box plots where noise is added equally to all reports. Figure B.5(b) show box plots where more noise is added to police reports. The box plots show lower quartile, median, upper quartile and outliers.

APPENDIX C: A TWO-STAGE GENETIC PROGRAMMING APPROACH FOR NON-PLAYER CHARACTERS

Developing intelligence for non-player characters is a complex task. Non-player characters need to be situationally aware of their environment, and make intelligent decisions based on their perceptions. Scripting techniques are limited and unflexible. In this study, we present a two-stage approach for developing non-player characters. In the first stage, tactical behaviors are learned in the context of situational models consisting of pre-determined environment situations and scoring measures. In the second stage, the tactical behaviors from the first stage are combined to form game strategies. The process is semi-automatic as it requires the game developer to provide relevant situational models as well as general outlines of the desired strategy. We test our approach in a turn-based artificial life game.

C.1 Introduction

Non-Player Characters (NPC) are used in many of today's games. One important factor deciding a game's economical success resides within its computer controlled NPCs. For a game to be considered fun, NPCs need to be intelligent but not too intelligent: they should exhibit human-like weaknesses and preferences. This equilibrium is very difficult to maintain as it depends on the human player's personal definition of what is exciting. Ideally game NPCs would always adapt to the player's idea of fun. Although the idea of "fun" is difficult to quantify, it is obvious that NPC's with scripted behavior are considered fun only at the first encounter, which limits the replayability of the game, and ultimately, its commercial success. The only reliable way to make NPC appear

intelligent is to provide it with genuine intelligence: the ability to make appropriate, non-scripted tactical decisions based on the environment and to assemble those in long term strategies.

In this study we present a two-stage genetic programming based approach to the development process of NPCs. The environment we consider is a simple game called Feed-Fight-Multiply (FFM) (see Section 3.5.3). In the first stage, tactical behaviors are learned using genetic programming and situational models consisting of pre-determined environment situations and scoring measures. In the second stage, the resulting behaviors are combined to form NPC game strategies. The process is semi-automatic as it requires the game developer to provide situational models as well as strategy configurations whereas the learning process is performed automatically.

C.2 Related Work

Schultz and Grefenstette [SG92] presented an approach to evolve reactive behavior for autonomous agents. Schultz and Grefenstette introduced a learning system called SAMUEL which utilized genetic algorithms to generate production rules describing behaviors for autonomous vehicles.

Koza [Koz90] showed that a tree-based representation containing LISP S-Expressions can be used in order to evolve executable code to mimic the behavior of an artificial ant trying to solve the Santa Fe Trail problem. The main problem with Koza's approach is that constraints must be introduced in the tree structure for evolutionary operators such as crossover and mutation to work properly.

In [Lar04] an approach that utilizes gramatical evolution [RCO98] was introduced. This approach differentiates from Koza's tree based genetic programming approach in the sense that individuals are represented as integers instead of trees. The integers are mapped to an arbitrary programming language that descibes the behavior of an individual. The main benefit of such approach reside in its flexibility as no contraints need to be included for the mutation and crossover operators.

C.3 The Genetic Programming Paradigm

Genetic Programming (GP) [Koz90] is an evolutionary algorithm in which the units of evolution are programs. Each program is represented by a tree, where the leaves are *terminals* while the nodes are *functions*. For example, the expression 7+(3*2) can be represented by a tree structure shown in Figure C.1. Tree structures can easily be parsed, compiled and executed as program code. The standard operations of genetic algorithms (crossover and mutation) can be easily defined on a tree structure. Initially, genetic programming relied only on crossover operations, and relied on a large initial population in order to keep diversity within the population [Mit96], [Fre94]. Other researchers, however, argued that mutation can be useful and have suggested several versions of genetic mutation operators. A simple version of a genetic mutation operator is to pick a random point in the tree and replace the structure below that point with a new randomly generated structure.

The genetic programming algorithm is similar to the standard genetic algorithm presented by Holland [Hol75]. The main difference lies in the elements used by the algorithm: the function set

and the terminals. The function set is a series of functions (arithmetical, boolean, etc) and statements (conditional, control, etc) that are available to the genetic programming algorithm [BNK98]. The terminals are a series of constants and variables that which serve as leaves of the tree. In the context of the example given in Figure C.1, the operators + and * would be defined as functions in the function set while the constants (7, 3, 2) would be defined as terminals. Furthermore, a fitness function must be defined in order to determine how well a specific individual can solve the problem trying to be resolved.

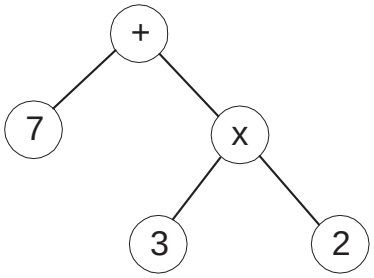


Figure C.1: Example tree structure that returns 7+(3*2).

The algorithm is controlled by a set of control parameters: the population size, crossover probability, mutation probability (if used), selection method and the number of generations for which the algorithm will run. As a result of crossover and mutation operators, the individual trees will grow in size and so it is necessary to determine a maximum size of the individual.

The genetic programming algorithm can be implemented as either generational genetic programming [Koz90], or as steady state genetic programming [Rey92] based on the genetic algorithm equivalent [Sys91].

The generational genetic programming algorithm can briefly be described as the following. After defining the function set and terminals, we construct the initial population. This is for most cases a random process, but for some applications we might start from a set of pre-determined individuals. Each individual is evaluated by the fitness function.

Depending on the selection algorithm a set of individuals is exposed to the genetic operators (crossover and in some cases mutation). The control parameter for crossover and mutation probability is used to determine which of the selected individuals to be exposed to the genetic operator, basically creating a set of new individuals. The algorithm is iterative and each of the individuals within the population is yet again evaluated by the fitness function. The algorithm continues until it has reached predetermined number of generations.

C.4 Learning Tactical and Strategic Behaviors

Our approach for creating NPCs is divided in two stages. In the first stage, we evolve tactical behaviors for specific situations. In the second stage, we combine the tactical behaviors into game strategies. The same approach can be generalized to almost any game, by adapting the nodes of the GP learning algorithm to the properties of the game.

The first stage aims to create the basic set of tactical behaviors needed to play the game in different situations. Such behaviors could be to locate food resources, to eat them, to avoid or engage in hostile conflicts with opponent characters in the game. A fitness function that determines if a tactical behavior is good or bad must be provided by the designer in order for GP to converge

to a solution. Note that these tactical behaviors alone can not be used to successfully play the game. Typical outputs gathered from the GP in this training stage consist of speed and direction of movement.

In the second stage the tactical behaviors determined in stage one are used in the training process to form game strategies. The strategies are evolved using GP and its user-defined fitness function. The choice of the fitness function determines the nature of the evolved strategy. For instance, if the fitness function favors strategies engaging in conflicts or attacking other characters in the game then the GP will prefer individuals with high attack measures and eventually it will converge to form this game strategy. Once the GP has converged and the best individual has been found it can be used to determine which tactical behavior to run based on the NPCs contextual perception. Hence, the individual evolved acts much like a Finite State Machine (FSM) where the states are represented by tactical behaviors and the transition rules are built into the individual in the form of control structures. The remainder of this section discusses the GP configuration used when training tactical behaviors and strategies for the FFM game.

C.4.1 Stage 1 - Evolving Tactical Behaviors

To evolve tactical behaviors we need to define a set of situational models and their desired characteristics. A situational model is defined by an environment, in our case a FFM game map, and a specialized fitness function. The situational models are tailored for each tactical behavior that is being learned. In our experiments, we created models for four types of tactical behaviors:

- *Eat-food*: Attract to food
- *Attack*: Attract to any other character
- *Flee*: Repel from any other character
- *Wander*: Attract to unexplored parts of map

All behaviors incorporate obstacle avoidance. The experiments were run using a generational genetic programming algorithm evolving for 50 generations with a population size of 100. Tournament selection with a size of 7 was used as selection algorithm. Crossover and mutation probability was set to 90% and 10% respectively. As for mutation, a single point mutation operator was chosen. The single point mutation selects a random node in the tree and replaces the sub-tree of that node with a new randomly generated sub-tree. The only constraint on the newly generated subtree is to be compatible with the selected node. The crossover operator was set to one-point, which is equivalent to the one-point operator used in many standard genetic algorithms.

The function set and terminals are described in Table C.1 for all four experiments. The terminal nodes marked with (*) depict terminal nodes that incorporate significant external processing. For instance, the food direction terminal is externally calculated based on the closest food resource available to the NPC and the unexplored direction terminal is calculated based on an external data structure which maintains visited waypoints in the world. The external processing is performed in helper functions. The obstacle functions, one for each direction, return a boolean value showing if

an the NPC can move in the given direction or not. The remaining functions are used as control structures. They are used to incorporate decisions in the tree.

Table C.1: T = Terminal, F = Function. Terminal nodes marked with (*) depict nodes that require significant external processing in the form of helper functions.

Name	Type	Description
*FoodDirection	T	Direction to the closest food.
*AgentDirection	T	Direction to the closest opponent NPC.
*UnexploredDirection	T	Direction to the closest unexplored region.
RandomDirection	T	Random direction
CurrentDirection	T	Current moving direction
ObstacleNorth	F	Check if obstacle is north
ObstacleSouth	F	Check if obstacle is south
ObstacleEast	F	Check if obstacle is east
ObstacleWest	F	Check if obstacle is west
Not	F	Operand used to invert boolean expressions
If	F	Conditional statement
Equals	F	Operand used in conditional statements
NotEquals	F	Operand used in conditional statements

C.4.2 Stage 2 - Evolving Game Strategies

Game strategies can be created by combining tactical behaviors in finite state machines. With the tactical behaviors already created, the challenge is to find the appropriate *transition rules*. We used GP to evolve the transitions rules for FSM-like structures. The main benefit of the approach is that many types of strategies can be created simply by adjusting the fitness function of the GP. The GP configuration used to demonstrate this feature is shown in Table C.2.

Table C.2: T = Terminal, F = Function.

Name	Type	Description
StateWander	T	Constant depicting the wander behavior
StateAttack	T	Constant depicting the attack behavior
StateEat	T	Constant depicting the eat behavior
StateFlee	T	Constant depicting the flee behavior
StateRandom	T	Constant depicting any behavior
Add	F	Addition operator
Mul	F	Multiplication operator
Sub	F	Subtraction operator
If	F	Conditional statement
Equal	F	Operand used in conditional statements
LesserEqual	F	Operand used in conditional statements
GreaterEqual	F	Operand used in conditional statements
Random	F	A random value
NumberOfAgents	F	Number of NPCs in sensor
NumberOfFood	F	Number of food resources in sensor
EnergyLevel	F	Energy level of the NPC

The terminal nodes consist of StateWander, StateAttack, StateEat, StateFlee and StateRandom. The StateRandom terminal is a randomly selected state and its primary function in the GP is to maintain population diversity. The terminal nodes depict tactical behaviors as described in stage 1. Addition, subtraction and multiplication functions enable the GP to find individuals operating on NPC sensor data. The NPC sensor data used in this GP is incorporated in the NumberOfFood, NumberOfAgents and EnergyLevel functions. These functions return data based on the NPCs situation in the game. Many interesting behaviors can be formed using this simple set of terminal nodes and functions. We created two types of strategies based on work performed in stage 1:

- *Balanced*: Avoid specialization

- *Aggressive*: Specialize on attacking

The balanced strategy seeks to create an NPC that doesn't specialize on any type of behaviors. Other than the terminals and functions seen in Table C.2 the GP configuration is exactly the same as described in stage 1.

C.5 Experimental Results - Stage 1

C.5.1 Eat-Food Behavior

The situational model for this experiment consisted of a 200x200 pixel game map. The map contains static obstacles covering approximately 20% of the map and 50 randomly distributed food units. The fitness function calculates a score between 0-100, where 0 is best or optimum score and 100 is worst score. An ideal individual is defined as an NPC that does not collide with obstacles and that eats all the food resources that are available. The score is derived based on the number of failed move sequences, the length of the failed move sequences and the amount of food eaten. Failed move sequences are first squared, to penalize agents that collides with obstacles over longer sequences, and then summed together to form a final sum of squares value. The sum of squares value is then divided by a pre-determined maximum value, in our case empirically set to 250000. As a result, a value between 0 and 1 is found for the failed moves scoring mechanism. Similarly, the actual amount of food eaten is divided by maximum amount of food, in our case 50, to generate a final score value between 0 and 1. To summarize, the final fitness value is calculated

using the following formula:

$$Fitness = \frac{(1 - S_{Food})}{2} + \frac{S_{FailedMoves}}{2} \tag{C.1}$$

Note that the food score is inverted as it is desirable to eat a lot of food. The two scoring components have equal influence or weight on the total fitness. Each agent is allowed to execute for 1000 simulation cycles to determine its fitness value. In order to avoid situations where an individual generates good fitness values purely based on its random initial location we calculated an average fitness from four runs using the same individual. Figure C.2 shows a fitness graph over 50 generations. The best individual over the experiment was found in generation 41 and it had a fitness value of 0.01 (very close to the optimum). Average fitness at the last generation was 0.3759 and best individual of the last generation had a fitness of 0.05. The randomly generated individuals in the initial population, in generation 0, perform very poorly, with an average fitness of 0.923. This is an expected result as the initial population is completely randomly generated.

As the results of the experiment, an operational eat behavior of the NPC was evolved. Although the optimal solution was not found, the best individual was able to, in most cases, effectively avoid obstacles and feast on the available food. Approximately 8 hours of computation time was required for the GP to complete 50 generations on a 1.8GHz Pentium-4 processor.

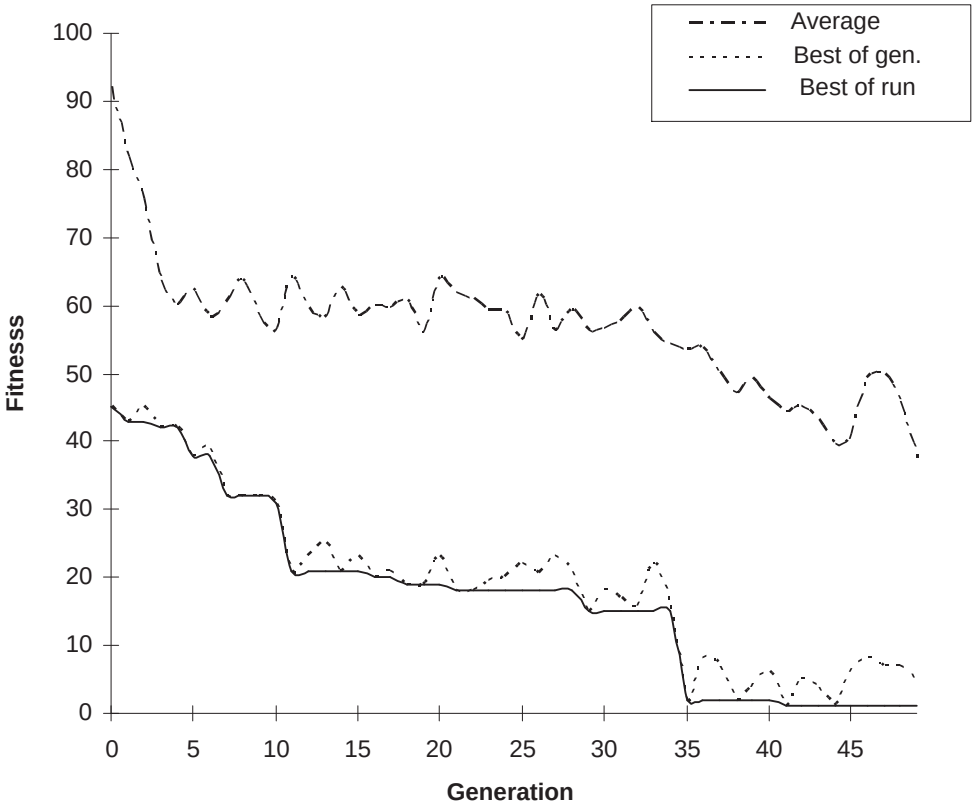


Figure C.2: Fitness graph for average fitness, best individual fitness of generation and best individual fitness of run for the Eat-Food behavior experiment.

C.5.2 Other Tactical Behaviors

The attack and wander behavior is very similar to the eat-food behavior. Therefore, we decided to reuse the best eat-food individual and merely replace FoodDirection terminals with AgentDirection terminals. Using this approach our NPC will follow opponent characters and attack them whenever it is possible. Similarly, if FoodDirection terminals are substituted with UnexploredDirection terminals the behavior will specialize to explore unexplored terrain. The flee behavior can

also be derived from the individual generated by the eat-food behavior experiment. For simplicity we simply replace all FoodDirection terminals with the inverted AgentDirection terminal. This generates the desired behavior where our NPC repels from other characters in the game.

C.6 Experimental Results - Stage 2

As we did not manage to evolve tactical behaviors which reliably avoid being stuck on obstacles, we decided to augment the evolved behaviors with helper functions. The helper functions used in stage 1 were extended to make use of a classic A* path-finding algorithm in order to avoid getting stuck on obstacles when traversing the environment. The game strategies, balanced and aggressive, were generated in a 400x400 pixel game map containing obstacles and 100 food resources. In addition, three types of opponent NPCs, two of each type, were added to the game during training. The first NPC type simply wanders around in the environment. The second type of NPC can attack and it is also able to eat food resources. The third NPC type can only flee if it is attacked. The purpose of the opponent NPCs is to generate hostile and conflicting situations from which strategies resolving these situations can be evolved. Strategies are formed based on weights that depict the importance of each specialized behavior. The fitness function is calculated using the following formula:

$$
\begin{aligned}
Fitness &= S_{Food} \cdot w_{Food} + S_{Attack} \cdot w_{Attack} \\
&+ S_{Flee} \cdot w_{Flee} + S_{Wander} \cdot w_{Wander}
\end{aligned} \tag{C.2}
$$

Food score is a measure of how much food compared to the total amount available was actually consumed. Attack score is a measure of how many attacks were successful. Flee score is a measure of how many flee commands were successfull. Finally, the wander score determines how much of the map that was covered. Each of these scores have values between 0-1. The maximum amount of successful attack and flee attempts are empirically chosen based on the game configuration. More opponent NPCs increase the chances of hostile situations to occur, therefore empirical measures must explicitly be defined. This is never an issue for the food score and map coverage measurements as the maximum values can be calculated in advance. Each individual in the population represent a valid game strategy and they are evaluated by executing them in the FFM game for 1000 simulation cycles each. Four NPCs using the same game strategy are allowed to execute concurrently to eliminate individuals that were successful purely based on their random initial location. The scores are averaged between the four NPCs. The four game strategy NPCs were not allowed to attack each other. The balanced game strategy was evolved using equal weights, 0.25 for each weight type. The aggressive strategy on the other hand was configured with higher weight values for attack and flee, 0.40, and remaining weights set to 0.1. Figure C.3 show a fitness graph for the average population for both game strategies.

As we can see from this figure, the attack strategy converged around generation 15, while the balanced strategy around generation 25. Approximately 24 hours of computation time was required for the GP to complete 50 generations on a 1.8GHz Pentium-4 processor.

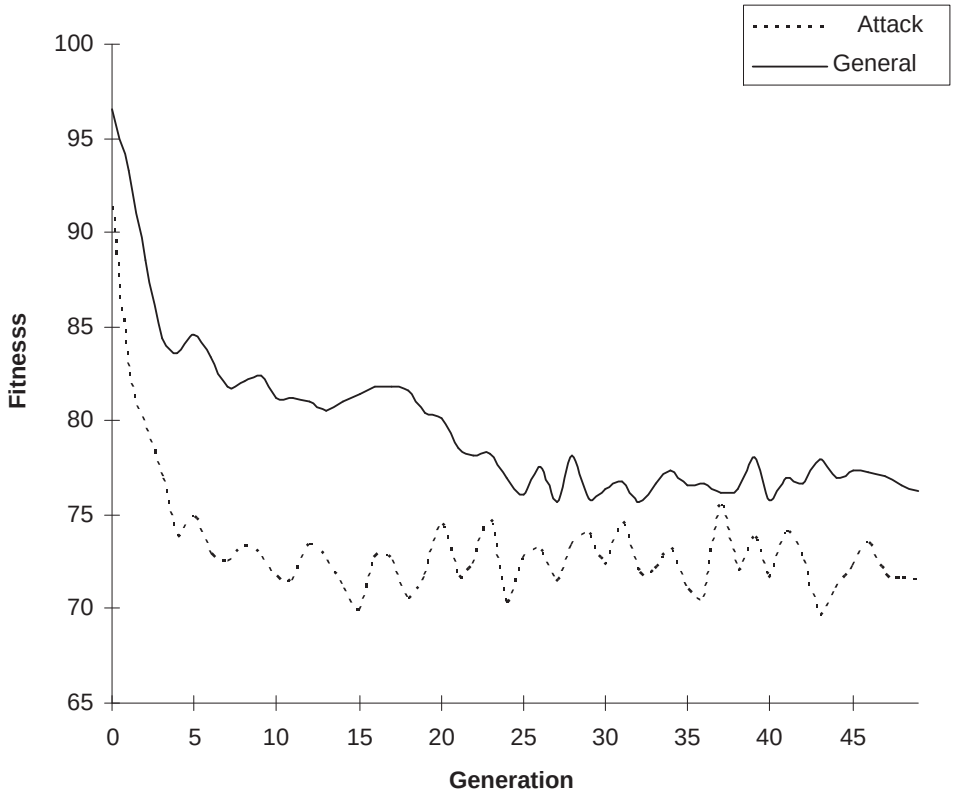


Figure C.3: Fitness graph for average fitness over 50 generations for the general strategy and the attack strategy. The attack strategy converges around generation 15 and the general strategy converges around generation 25.

C.6.1 Validation of Game Strategies

In order to validate the game strategies that were evolved, we allowed them to execute in the FFM map from Figure 3.12 for 20 individual runs each for 4000 simulation cycles. Each run was set up with 5 balanced strategy NPCs, 5 aggressive strategy NPCs, and 6 hand coded NPCs out of which 3 exhibited aggressive while 3 defensive behavior. The desired results are that the aggressive strategy will have high numbers of successful attacks and that the balanced strategy will not specialize on any type of behavior. Table C.3 and Table C.4 show the results of the validation runs. The aggressive strategy was able to perform on average 23.45 successful attacks. This is more then twice as much as the balanced strategy was able to do. The aggressive strategy involves no eating food and few attempts to flee when in conflict. The map coverage average of 4% for the aggressive strategy is very low. Thus, this specialized strategy is waiting for opponents to arrive in the sensor range and then it attacks.

Table C.3: Validation data for the attacker strategy.

	Average	Min	Max	Conf. Int. 95%
Attack	23.45	0.00	71.00	3.21
Eat	0.00	0.00	0.00	N/A
Flee	2.96	0.00	16.00	0.81
Coverage	0.04	0.01	0.15	0.0041

The balanced strategy that was evolved has on average 154.04 successful eat attempts, map coverage of 23% and 10.57 successful attack attempts. Note that it takes 4 successful eats to empty

Table C.4: Validation data for the general strategy.

	Average	Min	Max	Conf. Int. 95%
Attack	10.57	0.00	58.00	2.19
Eat	154.04	4.00	409.00	22.95
Flee	0.00	0.00	0.00	N/A
Coverage	0.23	0.02	0.49	0.03

a food resource with the game configuration used here. The balanced strategy, as expected, is more general then the attacker strategy as it involves eating, attacking and more extensive wandering.

C.7 Conclusions

Our experiments show that the genetic programming paradigm was successful in generating tactical behaviors, if provided with helper functions to guide the NPC in the FFM game. As with all evolutionary approaches better results may have been achieved by tweaking the GP configuration, for instance by using larger population sizes. These behaviors are able to eat food, attack opponents, flee and avoid obstacles. The tactical behaviors we evolved are non-optimal, for instance sometimes they get stuck on obstacles. We have described an approach which makes it easy to create various game strategies for NPC using tactical behaviors. Simply by adjusting a set of weights many types of NPC game strategies can be created. In this study, two types of game strategies, aggressive and balanced, were successfully created and validated. The balanced strategy was able to eat food, attack and wander around in the game. The aggressive strategy specialized on attacking opponent characters in the game.

In the experiments presented in this study, the NPCs have been evolved in a non-competitive environment. We believe that the evolved agents would evolve a different behavior in a different environment such as one containing competitive opponents. In our experiments we did not take into account the lack of food, or the risk of getting attacked and terminated. The nature of the FFM game might be too simplistic in such that complex game strategies are not easily evolved due to the small number of tactical behaviors. By adding more actions to the FFM game, such as spell casting or any similar game-like action, more options will become available. One option would be to have a number of tactical behaviors merged into one more complex behavior. These combined tactical behaviors are symptomatic with an agent being in a certain role or having certain characteristics. The developer would then be able to assign the agent a role or characteristics which would form a platform of behaviors of the agent.

APPENDIX D: MATHEMATICAL FOUNDATIONS

D.1 Linear Algebra

- Scalar: x
- Vector (column): $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & ... & x_n \end{bmatrix}$
- Matrix ($M \times N$): $\mathbf{A} = \begin{bmatrix} x_{11} & x_{12} & ... & x_{1n} \\ x_{21} & x_{22} & ... & x_{2n} \\ ... & ... & ... & ... \\ x_{m1} & x_{m2} & ... & x_{mn} \end{bmatrix}$
- Transpose: $\mathbf{A}^T$
- Symmetric Matrix: $\mathbf{A} = \mathbf{A}^T$
- Conjugate Transpose: $\mathbf{A}^+$
- Determinant: $|\mathbf{A}|$
- Matrix Inverse: $\mathbf{A}^{-1}$
- Identity Matrix: $\mathbf{I} = \begin{bmatrix} 1 & 0 & ... & 0 \\ 0 & 1 & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & 1 \end{bmatrix}$
- $|\mathbf{A}| = 0 \Rightarrow \mathbf{A}$ not invertible. If a matrix is not invertible it is also called ”Singular”.
- Eigenvectors and eigenvalues of a matrix $|\mathbf{A}|$ satisfies the following equation: $\mathbf{A}\phi_i = \lambda_i \phi_i$

- We can find the eigenvalues by solving: $|(\mathbf{A} - \lambda\mathbf{I})| = 0$ for n roots.
- Determinant of $\mathbf{A} = |\mathbf{A}| = \prod_{i=1}^{N} \lambda_i$.
- $|\mathbf{A}| = 0 \Rightarrow$ at least one $\lambda_i = 0$.
- $trace(\mathbf{A}) = \sum_{i=1}^{N} x_{ii} = \sum_{i=1}^{N} \lambda_i$
- $rank(\mathbf{A})$ = number of non zero λ_i's.
- If $\mathbf{A}$ is symmetric ($\mathbf{A} = \mathbf{A}^T$), λ_i's are real.
- If $\mathbf{A}$ is symmetric ($\mathbf{A} = \mathbf{A}^T$) and $\lambda_i \neq \lambda_j$ the $\phi_i\phi_j = 0$.
- Eigenvalues of a symmetric matrix form an orthonormal set.
- Condition number: $\lambda_{max}/\lambda_{min}$
- Positive definite matrix: $x^T\mathbf{A}x \geq 0$ for all x. This implies $\lambda_i > 0$.
- Positive semi-definite matrix: $\lambda_i \geq 0$
- Negative definite matrix: $\lambda_i < 0$
- Negative semi-definite matrix: $\lambda_i \leq 0$

D.2 Vector Calculus

- Scalar function: $f(x_1, x_2, ..., x_n) = f(\mathbf{x})$

- Derivative of scalar function: $\nabla_{\mathbf{x}} f = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \cdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$

- Vector function: $\mathbf{f}(x_1, x_2, ..., x_n) = \mathbf{f}(\mathbf{x})$, where $\mathbf{f}$ is an $m \times 1$ vector and $\mathbf{x}$ is $n \times 1$ vector.

- Derivative of vector function: $\nabla_{\mathbf{x}} \mathbf{f} = \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial \mathbf{f}_1}{\partial x_1} & \frac{\partial \mathbf{f}_1}{\partial x_2} & \cdots & \frac{\partial \mathbf{f}_1}{\partial x_n} \\ \frac{\partial \mathbf{f}_2}{\partial x_1} & \frac{\partial \mathbf{f}_2}{\partial x_2} & \cdots & \frac{\partial \mathbf{f}_2}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial \mathbf{f}_m}{\partial x_1} & \frac{\partial \mathbf{f}_m}{\partial x_2} & \cdots & \frac{\partial \mathbf{f}_m}{\partial x_n} \end{bmatrix}$

- $\frac{d}{d\mathbf{x}}(\mathbf{A}\mathbf{x}) = \mathbf{A}$

- $\frac{d}{d\mathbf{x}}(\mathbf{y}^T\mathbf{A}\mathbf{x}) = \mathbf{A}^T\mathbf{y}$

- $\frac{d}{d\mathbf{x}}(\mathbf{x}^T\mathbf{y}) = \mathbf{y}$

- $\frac{d}{d\mathbf{x}}(\mathbf{x}^T\mathbf{A}\mathbf{x}) = (\mathbf{A} + \mathbf{A}^T)\mathbf{x}$, if $\mathbf{A} = \mathbf{A}^T \Rightarrow 2\mathbf{A}\mathbf{x}$

D.3 Least Squares Method

Let $\mathbf{A}\mathbf{x} = \mathbf{b}$ depict a linear equation system where $\mathbf{A}$ is $m \times n$ matrix, $\mathbf{x}$ is a $n \times 1$ vector and $\mathbf{b}$ is a $m \times 1$ vector. There are three different configurations of such equation system:

- If $m = n$, the system only has one unique solution. In this case the equation system is solved using matrix inversion.

- If $n > m$, then the system has infinite solutions. This case can be handled, but not by the least squares method.

- If $m > n$, then there exists more equations than variables. To solve this equation system the Minimum Mean Square Error (MMSE) method can be used:

  - Define an error vector: $\mathbf{e} = (\mathbf{Ax} - \mathbf{b})$
  - Minimize the L_2-norm of the error vector: L_2-norm of $\mathbf{e} \Rightarrow \mathbf{J} = |\mathbf{e}|^2 = \mathbf{e}^T\mathbf{e}$
  - $\mathbf{J} = \mathbf{e}^T\mathbf{e} = (\mathbf{Ax} - \mathbf{b})^T(\mathbf{Ax} - \mathbf{b}) = \mathbf{x}^T\mathbf{A}^T\mathbf{Ax} - 2\mathbf{b}^T\mathbf{Ax} + \mathbf{b}^T\mathbf{b}$
  - Minimize by calculating the derivative and setting it to 0: $\nabla_{\mathbf{x}}\mathbf{J} = \mathbf{2A^TAx} - \mathbf{2A^Tb} + \mathbf{0} = \mathbf{0}$.
  - Solving this equation yields: $\mathbf{x} = (\mathbf{A^TA})^{-1}\mathbf{A^Tb}$, where $(\mathbf{A^TA})^{-1}\mathbf{A^T}$ is called the pseudoinverse.
  - In the special case where $m = n$ and $\mathbf{A}$ is invertible: $(\mathbf{A^TA})^{-1}\mathbf{A^T} = \mathbf{A}^{-1}(\mathbf{A^T})^{-1}\mathbf{A^T} = \mathbf{A}^{-1}$, which is the solution from the first configuration mentioned above.

D.4 Gradient Descent

Let $f(\mathbf{x})$ be a scalar function of a vector $\mathbf{x}$. The gradient decent procedure identifies the $\mathbf{x}$ that minimizes $f(\mathbf{x})$:

- Start with initial selection $\mathbf{x}'$
- Compute gradient of $\mathbf{x}'$, $\frac{df(\mathbf{x}')}{d\mathbf{x}}$

- Update, $\mathbf{x}' = \mathbf{x}' - k\left[\frac{df(\mathbf{x}')}{d\mathbf{x}}\right]$, where k is a scaling factor.

Furthermore, the gradient descent method can be used as an iterative solution to the least squares problem:

- $\mathbf{Ax} = \mathbf{b}$
- $\mathbf{e} = (\mathbf{Ax} - \mathbf{b})$
- $\mathbf{J} = \mathbf{e}^T\mathbf{e} = (\mathbf{Ax} - \mathbf{b})^T(\mathbf{Ax} - \mathbf{b}) = \mathbf{x}^T\mathbf{A}^T\mathbf{Ax} - 2\mathbf{b}^T\mathbf{Ax} + \mathbf{b}^T\mathbf{b}$
- $\nabla_{\mathbf{x}}\mathbf{J} = \mathbf{2A^TAx - 2A^Tb}$
- $\mathbf{x^{n+1} = x^n - k(n)\left[2A^TAx - 2A^Tb\right] = x^n - k(n)A^T\left[Ax - b\right] = x^n - k(n)A^Te}$

D.5 Probability Theory

- Let χ be a continuous random variable that takes on values over some interval (a, b).
- Cumulative distribution function (CDF): $F(a) = \text{Probability}\{\chi \leq a\}$
  - $F(\infty) = 1.0$
  - $F(-\infty) = 0.0$
- Probability density function (PDF): $p(x) = \frac{dF(\chi)}{d\chi}$
- For discrete random variables we have probability mass functions.

- Expected value of a function of a random variable: $g(\chi) = E(g(\chi)) = \int_{-\infty}^{\infty} g(\chi) * p(\chi) d\chi$
  - Mean: $E(\chi) = \int_{-\infty}^{\infty} \chi * p(\chi) d\chi$
  - Variance: $\sigma_{\chi}^2 = E(\chi^2) - E^2(\chi)$
- The joint CDF of two random variables (r.v.) x and y is given by: $F_{x,y}(a, b) = Pr(x \leq a, y \leq b)$
- The joint PDF is given by: $p_{x,y} = \frac{\partial^2 F_{x,y}(x,y)}{\partial x \partial y}$
- Marginal PDFs are given by:
  - $p_x(x) = \int_{-\infty}^{\infty} p_{x,y}(x, y) dy$
  - $p_y(y) = \int_{-\infty}^{\infty} p_{x,y}(x, y) dx$
- Two random variables are statistically independent if: $p_{x,y}(x, y) = p_x(x) * p_y(y)$
- Two random variables are uncorrelated if: $E(x * y) = E(x) * E(y)$
- Independence guarantees uncorrelatedness. The converse is not true except for Gaussian random variables.
- Covariance: $cov(x, y) = E((x - E(x)) * (y - E(y))) = E(x * y) - E(x) * E(y)$
- Correlation coefficient: $\rho = \frac{cov(x,y)}{\sigma_x \sigma_y}$
  - ρ approaching 1 or -1 means strong correlation. ρ approaching 0 means no correlation.

- Joint probability for N independent random variables $x_1, x_2, ..., x_N$: $p(\mathbf{x} = \prod_{j=1}^{N} p_{x_j}(x_j)$
  - $\mathbf{x}$ is a column vector.
  - $p_{\mathbf{x}}(\mathbf{x})$ is a N-variate distribution.
- Covariance matrix: $\mathbf{\Sigma} = \mathbf{E}((\mathbf{x} - \mu)(\mathbf{x} - \mu)^{\mathbf{T}}) = \mathbf{E}(\mathbf{x}\mathbf{x}^{\mathbf{T}}) - \mu\mu^{\mathbf{T}}$
  - The covariance matrix is always symmetric and positive definite.
  - Σ is diagonal when random variables are statistically independent.

D.5.1 Bayes Theorem

- Let A and B be two events with individual probabilities $P(A)$ and $P(B)$.
- Let $P(AB)$ denote the intersection of A and B.
- Conditional probability:
  - $P(A|B) = \frac{P(AB)}{P(B)}$
  - $P(B|A) = \frac{P(AB)}{P(A)}$
- If events A and B are independent then, $P(A|B) = P(A)$ and $P(B|A) = P(B)$.
- Let x and y denote two continuous random variables, then the conditional PDF is given by Bayes theorem: $p_{x|y}(x|y) = \frac{p_{x,y}(x,y)}{p_y(y)}$
- Hence, $p(x|y) = \frac{p(x,y)}{p(y)}$ and $p(y|x) = \frac{p(x,y)}{p(x)}$.

- $p(x|y) = \frac{p(y|x)p(x)}{p(y)}$

D.5.2 Distribution Functions

- Poisson distribution:
  - Discrete random variables $n = 1, 2, ..., \infty$
  - $Pr(n = k) = \frac{\lambda^k e^{-\lambda}}{k!}, k > 0$
  - $E(n) = \lambda, var(n) = \lambda$
- Binomial distribution:
  - Discrete random variable $n = 1, 2, ..., N$
  - $Pr(n = k) = \frac{N!}{k!(n-k)!}\theta^k(1-\theta)^{N-K}, k = 0, 1, ..., N$
  - $E(n) = N\theta$
  - $var(n) = N\theta(1-\theta)$
- Uniform distribution:
  - $p(x) = \begin{cases} \frac{1}{b-a} & a \leq x \leq b \\ 0 & otherwise \end{cases}$
  - $E(x) = \left(\frac{a+b}{2}\right)$
  - $var(x) = \left(\frac{(b-a)^2}{12}\right)$
- Univariate Gaussian distribution:

- $p(x) = \frac{1}{\sqrt{2x\sigma^2}} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}}$
- $E(x) = \mu$
- $Var(x) = \sigma^2$

LIST OF REFERENCES

[BCJ03] Todd M. Bacastow, Brian R. Cook, Kam-Chuen Jim, and C. Lee Giles. "Don't move: the T-Rex effect in the predator-prey world." In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems (AAMAS)*, pp. 924–925, New York, NY, USA, 2003. ACM Press.

[Bil97] J. Bilmes. "A Gentle Tutorial on the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models." *Technical Report, University of Berkeley*, 1997.

[BKT06] L. Bölöni, M.A. Khan, and D. Turgut. "Agent-based coalition formation in disaster response applications." In *Proceedings of the IEEE Workshop on Distributed Intelligent Systems*, pp. 259–264, June 2006.

[BLE07] L. Bölöni, L. J. Luotsinen, J. N. Ekblad, T. R. Fitz-Gibbon, C. Houchin, J. Key, M A. Khan, J. Lyu, J. Nguyen, R. Oleson, G. Stein, S. Vander Weide, and V. Trinh. "A comparison study of 12 paradigms for developing embodied agents." *Software: Practice and Experience (SPE)*, 2007.

[BNK98] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. "Genetic Programming - An Introduction: On the Automatic Evolution of Computer Programs and Its Applications." In *Morgan Kauffman Publishers Inc.*, 1998.

[BPS70] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains." *Annals Mathematical Statistics*, **41**(1):164–171, 1970.

[BT05] L. Bölöni and D. Turgut. "YAES - a modular simulator for mobile networks." In *Proceedings of the 8-th ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM 2005)*, pp. 169–173, October 2005.

[CWB95] Francine Chen, Lynn Wilcox, and Dan S. Bloomberg. "A comparison of discrete and continuous hidden Markov models for phrase spotting in text images." In *ICDAR*, pp. 398–402, 1995.

[DGN01] M. Dietl, J. Gutmann, and B. Nebel. "Cooperative sensing in dynamic environments." In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pp. 1706–1713, Maui, Hawaii, 2001.

[DHS00] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2000.

[DLR77] A. Dempster, N. M. Laird, and D. Rubin. "Maximum likelihood from incomplete data via the EM algorithm." *Journal of Royal Statistical Society, Series B (Methodological)*, **39**(1):1–38, 1977.

[DNC01] M.W.M.G. Dissanayake, P. Newman, S. Clark, H.F. Durrant-Whyte, and M. Csorba. "A solution to the simultaneous localization and map building (SLAM) problem." *IEEE Transactions on Robotics and Automation*, **17**(3):229–241, June 2001.

[FGE06] H. Fernlund, A. Gonzalez, and J. Ekblad. "Detecting Incorrect Intentions of Trainees by Context Deviations." In *Proceedings of 2006 Conference on Behavior Representation in Modeling and Simulation (BRIMS)*, 2006.

[FGI03] A. Farinelli, G. Grisetti, L. Iocchi, S. Lo Cascio, and D. Nardi. "Design and evaluation of multi agent systems for rescue operations." In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 4, pp. 3138–3143, October 2003.

[For73] G. David Forney. "The Viterbi Algorithm." In *Proceedings of the IEEE*, volume 61, pp. 268–278, 1973.

[Fox96] E. Foxlin. "Inertial head-tracker sensor fusion by a complementary separate-bias Kalman filter." *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pp. 185–194, 267, March 1996.

[Fre94] Gruau Frederic. "Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm." *Ph.D. thesis*, 1994.

[FS05] A. Farinelli and P. Scerri. "Low-Overhead Cooperative Detection of False Sensor Readings." In *Proceedings of AAMAS workshop: Challenges in the Coordination of Large Scale Multi-Agent Systems (LSMAS)*, pp. 11–16, Utrecht, July 2005.

[GBC02] Z. Guessoum, J. P. Briot, S. Charpentier, O. Marin, and P. Sens. "A fault-tolerant multi-agent framework." In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pp. 672–673, New York, NY, USA, 2002. ACM Press.

[GK04] Gery Gutnik and Gal Kaminka. "Towards a Formal Approach to Overhearing: Algorithms for Conversation Identification." In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 78–85, Washington, DC, USA, 2004. IEEE Computer Society.

[Gra84] R. M. Gray. "Vector Quantization." *IEEE ASSP Magazine*, pp. 4–29, April 1984.

[Hol75] J. H. Holland. "Adaptation in Natural and Artificial Systems." In *University of Michigan Press, Ann Arbor*, 1975.

[HS05] Asaad Hakeem and Mubarak Shah. "Multiple Agent Event Detection and Representation in Videos." In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pp. 89–94. AAAI Press / The MIT Press, 2005.

[HSS04] Asaad Hakeem, Yaser Sheikh, and Mubarak Shah. "CASEE: A Hierarchical Event Representation for the Analysis of Videos." In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-04)*, pp. 263–268. AAAI Press / The MIT Press, 2004.

[HV99] K. Han and M. Veloso. "Automated robot behavior recognition applied to robotic soccer." In *Proceedings of International Joint Conference on Artificial Intelligence Workshop on Team Behaviors and Plan Recognition (IJCAI-99)*, 1999.

[IB01] S. S. Intille and A. Bobick. "Recognizing planned, multi-person action." *Computer Vision and Image Understanding*, **81**(3):414–445, 2001.

[Int99] S. S. Intille. "Visual Recognition of Multi-Agent Action." *Ph.D. thesis*, 1999.

[JR90] B. H. Juang and L. R. Rabiner. "The Segmental K-Means Algorithm for Estimating Parameters of Hidden Markov Models." *IEEE Trans. Acoustics. Speech. and Signal Processing*, **38**(9):1639–1641, 1990.

[Kal60] R. E. Kalman. "A New Approach to Linear Filtering and Prediction Problems." *Transaction of the ASME - Journal of Basic Engineering*, pp. 35–45, March 1960.

[Kau90] H. Kautz. "A Circumscriptive Theory of Plan Recognition." In *Intentions in Communication*, pp. 105–133, Cambridge, MA, 1990. MIT Press.

[KKS06] Gregory Kuhlmann, William B. Knox, and Peter Stone. "Know Thine Enemy: A Champion RoboCup Coach Agent." In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, pp. 1463–68, July 2006.

[KM90] M. Koifman and S. J. Merhav. "Autonomously Aided Strapdown Attitude Reference System." *Journal of Guidance and Control*, **14**:1164–1172, 1990.

[Koz90] John R. Koza. "Genetically breeding populations of computer programs to solve problems in artificial intelligence." In *Proceedings of the Second International Conference on Tools for AI*, pp. 819–827. IEEE Computer Society Press, Los Alamitos, CA, USA, 1990.

[KPT02] Gal A. Kaminka, David V. Pynadath, and Milind Tambe. "Monitoring Teams by Overhearing: A Multi-Agent Plan Recognition Approach." *Journal of Artificial Intelligence Research*, **17**, 2002.

[KTN99] H. Kitano, S. Tadokoro, I. Noda, H. Matsubara, T. Takahashi, A. Shinjou, and S. Shimada. "RoboCup Rescue: search and rescue in large-scale disasters as a domain for autonomous agents research." *IEEE International Conference on Systems, Man, and Cybernetics (IEEE SMC '99)*, **6**:739–743, 1999.

[Lar04] Francois D. Laramee. "Advanced Genetic Programming: New Lessons from Biology." In *AI Game Programming Wisdom 2*, pp. 661–668, 2004.

[LC03] Xiaohui Liu and Chin-Seng Chua. "Multi-agent Activity Recognition Using Observation Decomposed Hidden Markov Model." In *Third International Conference on Computer Vision Systems ICVS*, volume 2626 of *Lecture Notes in Computer Science*, pp. 247–256, Graz, Austria, 2003. Springer.

[LEW05] L. J. Luotsinen, J. N. Ekblad, A. S. Wu, A. Gonzalez, and L. Bölöni. "A Two-Stage Genetic Programming Approach for Non-Player Characters." In *FuturePlay 2005 online proceedings http://www.futureplay.org/papers/paper-181_luotsinen.pdf*, October 2005.

[LMZ05] Tie Yan Liu, Wei Ying Ma, and HongJiang Zhang. "Effective Feature Extraction for Play Detection in American Football Video." In *Proceedings of 11th International Conference on Multi Media Modeling (MMM'05)*, pp. 164–171, 2005.

[LS01] Baoxin Li and M. Ibrahim Sezan. "Event Detection and Summarization in Sports Video." In *Proceedings of the IEEE Workshop on Content-based Access of Image and Video Libraries (CBAIVL'01)*, pp. 132–138. IEEE Computer Society, 2001.

[LT03] Francois Legras and Catherine Tessier. "LOTTO: group formation by overhearing in large teams." In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pp. 425–432, New York, NY, USA, 2003. ACM Press.

[Mac03] Dana Mackenzie. "Ensemble Kalman Filters Bring Weather Models Up to Date." *Society for Industrial and Applied Mathematics (SIAM) News*, **36**(8), October 2003.

[May79] P. S. Maybeck. *Stochastic Models, Estimation, and Control*, volume 1. Academic Press, New York, 1979.

[Mit96] M. Mitchell. "An Introduction to Genetic Algorithms." In *The MIT Press*, 1996.

[NTM04] R. Nair, M. Tambe, S. Marsella, and T. Raines. "Automated assistants for analyzing team behaviors." *Journal of Autonomous Agents and Multiagent Systems (JAAMAS)*, **8**(1):69–111, 2004.

[PLL96] M. V. Nagendra Prasad, Victor Lesser, and Susan Lander. "Learning Organizational Roles in a Heterogeneous Multi-Agent System." *Proceedings of the Second International Conference on Multi-Agent Systems*, pp. 291–298, January 1996.

[PMJ02] Milan Petkovic, Vojkan Mihajlovic, Willem Jonker, and S. Djordjevic-Kajan. "Multi-Modal Extraction of Highlights from TV Formula 1 Programs." In *Proceedings of IEEE International Conference on Multimedia and Expo (ICME'02)*, volume 1, pp. 817–820, 2002.

[Qui88] J. R. Quinlan. "Simplifying Decision Trees." In B. Gaines and J. Boose, editors, *Knowledge Acquisition for Knowledge-Based Systems*, pp. 239–252. Academic Press, London, 1988.

[Rab90] L. R. Rabiner. "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition." In A. Waibel and K.-F. Lee, editors, *Readings in Speech Recognition*, pp. 267–296. Kaufmann, San Mateo, CA, 1990.

[RB04] Silvia Rossi and Paolo Busetta. "Towards Monitoring of Group Interactions and Social Roles via Overhearing." In *Cooperative Information Agents VIII*, volume 3191, pp. 47–61. Springer Berlin / Heidelberg, 2004.

[RCO98] Conor Ryan, J. J. Collins, and Michael O'Neill. "Grammatical Evolution: Evolving Programs for an Arbitrary Language." In *EuroGP 1998*, pp. 83–96, 1998.

[Rey92] Craig W. Reynolds. "An Evolved, Vision-Based Behavioral Model of Coordinated Group Motion." In Meyer and Wilson, editors, *From Animals to Animats (Proceedings of Simulation of Adaptive Behaviour)*. MIT Press, 1992.

[RVK02] Patrick Riley, Manuela Veloso, and Gal Kaminka. "An Empirical Study of Coaching." In H. Asama, T. Arai, T. Fukuda, and T. Hasegawa, editors, *Distributed Autonomous Robotic Systems 5*, pp. 215–224. Springer-Verlag, 2002.

[SG92] Alan Schultz and John J. Grefenstette. "Using a genetic algorithm to learn behaviors for autonomous vehicles." In *Proceedings American Institute of Aeronautics and Astronautics Guidance, Navigation and Control Conference*, pp. 739–749. AIAA, 1992.

[SMB01] Ashley Stroupe, Martin C. Martin, and Tucker Balch. "Distributed Sensor Fusion for Object Position Estimation by Multi-Robot Systems." In *IEEE International Conference on Robotics and Automation (IRCA)*, volume 2, pp. 1092–1098. IEEE, May 2001.

[SOM05] N. Schurr, S. Okamoto, R. Maheswaran, Paul Scerri, and M. Tambe. "Evolution of a Teamwork Model." *Cognitive Modeling and Multi-Agent Interactions*, 2005. Forthcoming.

[SS06a] Gita Sukthankar and Katia Sycara. "Robust Recognition of Physical Team Behaviors using Spatio-temporal Models." In *Proceedings of Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2006.

[SS06b] Gita Sukthankar and Katia Sycara. "Simultaneous Team Assignment and Behavior Recognition from Spatio-temporal Agent Traces." In *Proceedings of Twenty-First National Conference on Artificial Intelligence (AAAI-06)*, 2006.

[Ste99] James Stewart. *Calculus (4th Edition)*. Brooks/Cole Publishing Company, 1999.

[Suk07] Gita Sukthankar. "Activity Recognition for Agent Teams." *Ph.D. thesis*, 2007.

[Sys91] G. Syswerda. "Scheduling optimization using genetic algorithms." In *Handbook of Genetic Algorithms*, pp. 332–349, 1991.

[Tam96] M. Tambe. "Tracking dynamic team activity." In *National Conference on Artificial Intelligence (AAAI96)*, 1996.

[Tam97] M. Tambe. "Towards Flexible Teamwork." *Journal of Artificial Intelligence Research*, **7**:83–124, 1997.

[TR95] Milind Tambe and Paul S. Rosenbloom. "RESC: An Approach for Real-time, Dynamic Agent Tracking." In *International Joint Conference on Artificial Intelligence IJCAI*, pp. 103–111, 1995.

[TZ98] M. Tambe and W. Zhang. "Towards flexible teamwork in persistent teams." *Proceedings of the International Conference on Multi-Agent Systems (ICMAS)*, 1998.

[VAC06] Manuela Veloso, Nicholas Armstrong-Crews, Sonia Chernova, Elisabeth Crawford, Colin McMillen, Maayan Roth, and Douglas Vail. "A Team of Humanoid Game Commentators." In *Proceedings of Humanoids 2006*, December 2006.

[VAH99] Dirk Voelz, Elisabeth André, Gerd Herzog, and Thomas Rist. "Rocco: A RoboCup Soccer Commentator System." In *Proceedings of RoboCup-98: Robot Soccer World Cup II*, pp. 50–60. Springer-Verlag, 1999.

[VVL07] Douglas L. Vail, Manuela M. Veloso, and John D. Lafferty. "Conditional Random Fields for Activity Recognition." In *Sixth International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2007)*, May 2007.