Bar-Ilan University
Department of Computer Science

ON INTEGRATED MULTI-AGENT INTENTION RECOGNITION SYSTEMS

by

Nirom Cohen-Nov Slapak

Advisor: Dr. Gal A. Kaminka

Submitted in partial fulfillment of the requirements for the Master's degree
in the Department of Computer Science, Bar-Ilan University

Ramat-Gan, Israel
August 2008

This work was carried out under the supervision of

Dr. Gal A. Kaminka

Department of Computer Science, Bar-Ilan University.

Acknowledgements

I wish to express my gratitude to my supervisor, Dr. Gal A. Kaminka for his truly scientist intuition that guides me through this exciting and emerging research area, and for opening my eyes to the fascinating world of agent architecture. For without his perspective on relative fields (e.g. Psychology and signal processing), I would never have finished to write this work. Through all these years his guidance has been most valuable.
Deepest gratitude are also due to the members of MAVERICK group, Natalie Fridman and Ari Yakir without whose knowledge and assistance this study would not have been successful.
I wish to express my love and gratitude to my beloved husband, Alon; for his understanding & endless love, through the duration of my studies.

Contents

List of Figures

List of Tables

List of Algorithms

Abstract

Intention recognition is the ability to reason and infer information about others, based on observations of their behavior. Unfortunately, to date there have been only a handful of investigations into the integration of intention recognition into agent architectures. In particular, there are open questions as to the effect of intention recognition on the computational resources available to the agent. The issue is of particular importance in modern virtual environments, where the agent may be interacting with multiple other agents. This work tackles this question analytically and empirically. First, we examine run-time considerations, and offer a novel view of plan-recognition as a sampling process. Under this view, an existing work can be viewed as if trying to reduce the computational load on the agent by reducing the number of hypotheses it considers. In contrast, we reduce the ***Frequency*** by which the recognition process is sampled. We provide an analytical model allowing selection of a fixed-frequency recognition, and examine a number of heuristics for dynamically-changing plans. In the second part of the thesis, we consider a method of integrating plan recognition processes, called ***Mirroring***, in which the executable knowledge of the agent is re-used, as is, for recognition.

Chapter 1

Introduction

Intention recognition is the ability to monitor observable actions of other agents and from those, infer their unobservable goals and plans [13, 38, 16]. Previous research has shown that intention recognition is an important and beneficial capability [37, 16, 21, 48, 53, 17, 2, 34, 60]. For example, in a hostile environments, an agent should know it's opponent's intentions [60, 50, 35, 64] and accordingly decide its own actions for better results. In teamwork, intention recognition is beneficial for detecting disagreements and coordinating without communication [26, 59, 52, 34]. In real time tracking systems, intention recognition is important to detect a suspicious behavior for surveillance and activity recognition [7, 58]. In natural language, intention recognition improve user-computer communication by assimilating ongoing dialogue and reasoning on the acquired knowledge [12, 23, 17, 2]. Another critical role of intention recognition is the ability to identify failures at agent activities, to improve the system performances and robustness [34]. Recent work [31, 20] on crowd behavior modeling by social comparison theory strongly suggests that knowing about others, at all times (simultaneously with self-modeling), is a basic need.

Most of the previous research in intention recognition has focused on standalone plan recognition systems. For example, AHMEM [11] is a general framework for online probabilistic plan recognition which was implemented in a surveillance domain. Monitoring by Overhearing [32]; a monitoring approach that applies plan-recognition based on team members' routine communications. ISAAC

[49] is an off-line automated team analysis system, that applied team modeling based on machine learning. Additional applications include modeling and analyzing behavior of live insect colonies [10] using an off-line system that applies vision-based observations.

Assimilating intention recognition capability in agent architecture has many advantages: 1. It is essential for the flexibility and efficiency of the modeling task. 2. Resources saving eliminating the need for knowledge maintenance, by performing recognition on the same data structures; and eliminating the need for synchronizing two databases. In addition, the need to maintain two plan library databases and a special API for data transferring between the executer agent and the observed agent prevents software development and debug processes from being easily accomplished. 3. Intention recognition capability in agent architecture may increase the opportunities for other social mechanisms, such as a social comparison mechanism [20], self-tracking (where an agent tracks itself to achieve fault tolerance) [33], and an integrated imitation mechanism [18, 42, 20].

Unfortunately, despite the demonstrated need for integrating intention recognition in agents, there have been only a handful of investigations, into the integration of intention recognition into agent architectures. Thus, the integration of intention recognition capabilities into agent architectures is an open challenge.

Examples for works which face the challenge of integrating the intention recognition into agent architecture are: Chalupsky's work [15], on SIMBA (SIMulative Belief Ascription), implements mechanism in formal and logical language; Laird's work [39], adds anticipation to an AI bot for the game Quake II; Tambe and Rosenbloom's work [60], on REal-time Situated Commitments, implements intention recognition in the intelligent pilot agent participating in a real-world synthetic air-combat environment; Kaminka and Tambe's work [34], on RESL implements detecting failures algorithms in teams of cooperating agents via social relationships between the agents. Rao and Murray's work [50], provides reactive plan recognition within the framework of the agent's mental state, and applies it in a pilot agent which operates in an air-combat modeling.

This research examines the integration of intention recognition into cognitive architectures. The key challenge faced in such integration is reducing the computational load of the recognition process, which is fueled by the need to reason about the inherent ambiguity of the recognition results. In general, more than one hypothesized explanation exists for observations and each observed agent may trigger multiple recognition hypotheses. Thus, the computational load of recognition is exacerbated in the presence of multiple observed agents.

In an integrated recognition system the agent's computational load (hereinafter *Computational Load*) consists, as in Equation 1.1, of the load results from self executing (hereinafter *Execution Load*) and from the load results from the recognition algorithm it activates on the observed agents (hereinafter *Recognition Load*). Our research objective is to evaluate the integration of the recognition ability into the agent architecture, by evaluating the *Recognition Load*.

$$\textit{Computational Load} = \textit{Execution Load} + \textit{Recognition Load} \tag{1.1}$$

Integrated intention recognition architecture must deal with the following questions: When to perform the intention recognition process (all the time [60], or according to an event [39]); and whom to perform the intention recognition process on (one agent [60], or on all agents [34]). In general, there would be more than one explanation that fits the observations. Because of real-time requirements, we cannot wait for the observed actions to resolve ambiguities, so the mechanism may need to commit to a single chosen hypothesis and be able to correct it [60], or to maintain all the possible hypotheses at the same time [34].

Integrated intention recognition systems must therefore be selective in their recognition process. We argue that previous work in this area can in fact be categorized according to the way monitoring selectivity is addressed. Some [60, 34] limit the number of maintained hypotheses for others, some trigger the recognition processes selectively [39], or limit the focus to specific agents [34, 28]. Nevertheless, none of these approaches has yielded an architecture with built-in intention-recognition capabilities.

This research explores a novel approach to monitoring selectivity in the presence of multiple agents. The key to this approach is to reduce the time in which recognition is carried out. Instead of executing the recognition process with every sense-think-act cycle as previous approaches do, we choose instead to apply selectivity in terms of the time in which the process is triggered. Our approach is called, hereinafter, *Temporal Monitoring Selectivity* (details in Chapter 3). The *Temporal Monitoring Selectivity* approach can be viewed as a sampling technique, in which one chooses the time points for using the observations to generate hypotheses, see for details Section 3.1.

Two major sampling heuristics can be implemented in *Temporal Monitoring Selectivity*, one is designed as *static heuristic* and the other *dynamic heuristic*. In the *static heuristic*, which is the most commonly-used sampling scheme, the recognition algorithm is periodically triggered with a fixed rate which is determined in advance. The *static heuristic* assumes no knowledge of the recognition algorithm. Using *static heuristic* with a fixed rate, smaller than the maximum rate, the system can trigger the recognition algorithm, directly reducing the *Recognition Load*. But, this heuristic cannot guarantee the quality of the recognition. *Static heuristic* involves a tradeoff decision for the users to make: How much are they willing to lose in recognition quality in order to gain a decrease in *Recognition Load*.

The *dynamic heuristic* dynamically adjusts the rate, in which the recognition process is triggered based on information about others, and the recognition load. Some heuristics in which the sampling rate is dynamically changed were suggested and their performances were compared to the fixed-rate heuristic. It was shown that the improvement obtained by dynamic sampling can indeed be significant. We show that using *dynamic heuristic* to trigger the recognition algorithm, achieves a decrease in *Recognition Load* while not affecting recognition quality, Section 3.3.

We present analytical models to evaluate different heuristics for the *Temporal Monitoring Selectivity* approach in an integrated recognition system. First we evaluate the *Recognition Load* and the recognition quality when running the integrated system using different values of fixed rates. We start with the maximum rate in which the integrated system can trigger the recognition algorithm and finish with the lowest rate which still provides some degree of accuracy. We use the *Recognition Load* and the recognition quality of the fixed rates as a baseline for the integrated system. Then, we suggest several heuristics, whose performance will be compared with this baseline.

In the second phase of this research the *Mirroring* approach is presented. *Mirroring* is a specific approach to providing intention recognition at the architectural level, in which the executable procedural knowledge of the agent is re-used, as is, for recognition (details in Chapter 4). The *Mirroring* approach offers significant savings in computational resources compared to most existing approaches (where a separate recognition knowledge plan library is utilized, e.g., as in [26, 34, 39]). It also provides greater opportunity for social mechanisms that rely on comparisons between the executer agent and the observed agent (e.g., for identifying failures [34], or for social comparison [31]). In this phase we show how the *Temporal Monitoring Selectivity* approach has yielded in an architecture with built-in intention recognition capabilities. We describe *Mirroring* in detail, and discuss the architectural requirements that are needed to allow it to work. The *Mirroring* approach is implemented in M-DIESEL a prototype architecture built on top of Soar [42].

The *Temporal Monitoring Selectivity* approach is evaluated on two simulators. The *suspicious behavior recognition* simulator which executes the integrated intention recognition system SBR [7], and the VR-Forces simulator which executes the integrated intention recognition architecture M-DIESEL. We conduct experiments to measure the load and the quality of the two systems using different heuristics and conclude that the dynamic policy significantly outperforms the fixed frequency sampling (details in Chapter 5).

Chapter 2

Related Work

This work is obviously related in general to plan recognition. However, the literature on plan recognition is vast, and most of it is not directly related — our work is complementary. We therefore point the reader to [38, 16, 13] for general introduction to plan recognition, and focus here on more closely related work.

2.1 Monitoring Selectivity

An open challenge in all integrated intention recognition systems is to manage resources, while maintaining recognition results that support the agent's reasoning processes, which require information about what others are doing. It has been hypothesized that a trade-off may exist between the load on the agent and the recognition quality [60]. The intuition for this trade-off is that the more resources the agent has to spend on recognition (up to a point), the faster it can converge to its resolution. Therefore, improving the recognition while minimizing the load is not likely to be a solution. Instead, some monitoring selectivity is required for optimization.

Therefore, integrated intention recognition systems must be selective in their recognition process. Indeed, previous investigations addressing integration of intention recognition can be viewed as taking different approaches to provide heuristics to the problem of monitoring selectivity.

One general approach is focused on reducing the number of hypotheses about which the agent reasons, thereby reducing the computational load involved in generating hypotheses and reasoning. RESC [60] was developed to provide soft real-time recognition responses, by eliminating all hypotheses but one (which is selected based on a combination of hypothesis-ranking heuristics). Similarly, a related algorithm called RESL [34] was used to maintain only 1–2 hypotheses, to detect disagreements in a team of agents. Thus RESC and RESL sacrifice recognition completeness–they cannot guarantee that the correct hypothesis is returned in their results. In contrast to these works, our method pursues all hypotheses but restricts the time in which the recognition process is allowed to run.

Laird [39] takes a different approach to monitoring selectivity. He explores an integrated intention recognition system (observing a single agent), which is triggered only under task-dependent conditions, typically when the agent has sufficient time and computation resources to carry out the recognition and when the result is likely to be useful. Moreover, the recognition process terminates if multiple hypotheses are available, since the agent has no way to select between them. By dynamically triggering the recognition process, Laird's approach is a precursor to the heuristics that we develop in this research. However, in contrast to this earlier work, our focus is on domain-independent heuristics for multiple agents. Furthermore, we support multiple hypotheses for these agents.

Another approach to monitoring selectivity in the context of multiple agents, is to limit the number of agents that are tracked. Kaminka and Tambe [34] and later Kaminka and Bowling [28] have provided a number of analytical bounds to the number of agents that need to be monitored, for specific recognition tasks. Our approach here is complementary to these works.

Our approach relies on the ability of the intention recognition system to freeze itself and resume again, while taking into account missing observations. Previous works propose operation models to address the lack of continuity in the observations stream. Kaminka et. al. [32], suggest using a temporal model to simulate execution while no observations are received. Avrahami-Zilberbrand et. al. [9, 7], propose the use of a special tag on unobservable plans to allow the recognition algorithm to ignore lack of observations for them.

2.2 Dynamic Sampling

Dynamic sampling utilizes the option of varying the sampling rates according to the situation of the system, thus obtaining intention recognition with improved efficiency. The basic idea is the following: Rather than allocating a fixed amount of resources to each recognized agent, apply a dynamic recognition procedure. By using a dynamic approach, the recognition is controlled and can be changed any time according to actual needs.

When observing a single agent, the recognition algorithm can be ruled according to the preference in resources allocation between the observed agent and the executed agent. This possibility may be particularly appealing when a large number of agents has to be observed in parallel. Rather than giving each agent a fixed share of the recognizer resources (either in terms of observer units or time slots), we may want to divide the recognizer resources among the agents depending on their state and the importance of an immediate updating of the state of each of them. Thus, when the recognition of a given agent is known fairly well, most of the resources may be shifted to the recognizing of the other agents.

In this section we survey several areas in which dynamic sampling is used and achieves improvement, compared with fixed sampling. It should be emphasized that the dynamic sampling procedure is suitable for those situations in which the amount of resources of the sampling equipment (e.g., radar, agent architecture) is adjustable.

Target Tracking

In the example of controlling N objects using radar [6, 63], the same amount of samples are executed by the fixed rate heuristic to the objects. The dynamic sampling heuristic, on the other hand, suggests allocating most of the resources to a one specified object for a short and random time and then shifting the resources to another specified object.

Assaf and Ritov [6], apply the dynamic sampling technique to a typical problem in optimal control theory, that of tracking and controlling the position of an object. They first present the results for the basic model (with a fixed sampling rate) then the dynamic sampling formulation of the problem is presented as an optimization problem in the state space. They write the formal optimality equation with the objective of minimizing the average cost per time unit. After that, they propose a heuristically sub-optimal sampling and control heuristic. The performance of that heuristic is then compared with the fixed-rate heuristic and it is shown that the improvement obtained by dynamic sampling is indeed significant. We also formulate our problem, present the results of fixed sample rates and suggest several dynamic heuristics in which their performance are compared with the fixed-rate heuristic.

Previous works in *Target Tracking* area vary the sampling rate subject to a restriction. Assaf and Ritov [6], keep the long-run average dynamic sampling rate at the same level of the fixed rate sampling procedure. In Zhang et. al. work [63], when having a fixed sampling interval T, the kind of dynamic sampling considered is that the system samples at interval $T1$ and $T2$, alternatively. But to make the overall sampling rate equal and to have a fair comparison, they have a constraint of $T1 + T2 = 2T$. In this work we do not use a cost depending on the sampling rate which is used, and we do not keep an average sampling rate at the same level as the fixed rate sampling.

Data Mining

Data mining involves sorting through large amounts of data and picking out relevant information using algorithms which may also use dynamic sampling approaches. As data warehouses grow the computational efficiency of data mining algorithms on large databases becomes increasingly important [27].

Sampling a database involves a decision about a tradeoff between accuracy and a decrease in running time of a data mining algorithm. Using a sample from the database can speed up the data mining process, but this is only acceptable if it does not reduce the quality of the mined knowledge. Static Sampling assumes no knowledge of the running algorithm. It applies some fixed criteria to the sample to determine if it is suitably representative of the original large database. This helps improve the efficiency by directly reducing the size of the training data. However, this strategy can not guarantee the learning accuracy. Dynamic sampling refers to the use of knowledge about the behavior of the mining algorithm in order to choose a sampling size. We expect to directly reduce the computational load when using fixed sampling on the recognition process but also to downgrade in the recognition outcomes.

John and Langley [27] introduce the 'Probably Close Enough' (PCE) criterion to describe the desired properties of a sample, as a way of evaluating sampling strategy. When they believe that the performance of their data mining algorithm on a sample is probably close to what it would be if they ran it on the entire database, then they would be satisfied with the sample. PCE is similar to the Probably Approximately Correct bound in computational learning theory.

In this work we introduce the *Recognition Error* parameter, which expresses the quantity of *close enough* between the evaluated sampling and the system highest fixed rate sampling.

Given this framework, many learning algorithms can be used to estimate *close enough*. John and Langley [27] choose the naive Bayesian classifier. Experimental studies suggest that naive Bayes tend to learn more rapidly, in terms of the number of training cases needed to achieve high accuracy, than most induction algorithms (Langley and Sage [40]). In addition, they like to reduce the naive Bayes computational complexity even further by incorporating dynamic sampling.

Another good example of dynamic sampling are the peepholing algorithm described by Catlett [14] and races of Moore and Lee [41].

Detecting a Change

Another subject area in which dynamic sampling is used for improving the performance is *Detecting a Change*. The problem originally arose out of considerations of quality control. When a process is 'in control', observations are distributed according to Fo. At the unknown point v, the process jumps 'out of control' and ensuing observations are distributed according to Fl. The objective is to detect that a change took place as soon as possible' after its occurrence, subject to a restriction on the rate of false detections.

In our scope of interest, in integrated intention recognition, the objective is to detect that a change in the observed agent's behavior took place 'as soon as possible' after its occurrence. This problem also is important in the study of impacts of treatment, since the point when an action of the executer agent on the observed agent might take effect is usually unknown [54] and should take into account in the executer's decision system.

In order to evaluate and compare procedures, [3, 45, 46, 5, 4] formalize a restriction on false detections, as well as formalize the objective of detecting a change 'as soon as possible' after its occurrence. We also, determine criteria to express the different between the actual change in the observed agent behavior and the sampled one, as we consider an actual change in the observed agent behavior, as the one occurring with the system highest fixed rate sampling on the recognition process.

Pointers for the solution to the optimal stopping problem may be found in the references of [3, 45, 46, 5, 4]. For an up to date survey and a more complete list of references, see Pollak [45, 46].

The standard procedures with a constant sampling rate are the Page (or CUSUM) procedure [43, 44] and the Roberts-Shiryayev procedure [55]. The idea of dynamic sampling has been used by Girshick and Rubin [24] from a Bayesian decision point of view. Recently, Assaf [3] considered a Bayesian dynamic sampling procedure in the surveillance context and which re-initiated interest. In [3] a heuristic derivation of the optimal rate and stopping time for the dynamic sampling problem is given. The performance of the resulting heuristic is approximated by a family of sub-optimal heuristics.

Srivastava and Wu [57] propose that the dynamic sampling plan as a procedure which takes fewer samples when no change is expected but takes more samples when a change is expected, should be more efficient than the corresponding fixed sampling plan. Our dynamic heuristics in this work is based on that distinction [57]. The density of the recognition samples will be controlled by the uncertainty level about the observed agent behavior, thus, take more samples when a change in agent behavior is expected.

Assaf, Ritov, Pollak and Yakir in [3, 5, 4] and Srivastava and Wu in [57] show comparisons between the dynamic sampling procedures and previous results obtained for a fixed sampling rate. The comparisons show that employing dynamic sampling heuristics can result in a dramatic reduction in the maximal expected delay to the fixed rate one.

Signal Analysis

Modern digital data processing of functions (or signals or images), uses a discretized version of the original signal f that is obtained by sampling f on a discrete set. The question then arises whether and how f can be recovered from its samples. Therefore, the objective of research on the sampling problem is twofold. The first goal is to quantify the conditions under which it is possible to recover particular classes of functions from different sets of discrete samples. The second goal is to use these analytical results to develop explicit reconstruction schemes for the analysis and processing of digital data.

Since infinitely many functions can have the same sampled values, the sampling problem becomes meaningful only after imposing some a priori conditions on f (the original one). The standard assumption is that the function f on R^d belongs to the space of band-limited functions B. In fact, all band-limited functions have infinite support since they are analytic. However, non-band-limited functions that retain some of the simplicity and structure of band-limited models are more amenable to numerical implementation and are more flexible for approximating real data. Since our recognition process is time limited, we assume that our signal is band-limited as was shown in [1]. Since, the behavior plan library of the observed agent is finite, the number of the potential behaviors which returned by the recognition algorithm is finite, too. Therefore, we can give a lower bound on the maximal distance between two sampling points needed for reconstructing a function from its samples as was required in [1].

In the same way of reconstructing [1], we derive mathematical models, based on fixed samplings, of the recognition process. A model help us to analytically determine the *best* fixed sampling for reconstruct the entire recognition process.

2.3 *Mirroring*

An important finding of neuroscience occurred in the last decade. Mirror neurons (MN) were discovered in the frontal lobes of macaques [19, 51]. MN are active when the monkeys perform actions; however, these neurons are also fired when the monkeys watch someone else perform these actions. Due to brain evolution, there is strong assumption that this observation/action matching system also exists in the human brain. Today, MN play a major role in explaining some human abilities, such as empathy, language learning, imitation, and theory of mind. In the MN mechanism in the human brain, the same neurons are involved either in the action and in the recognition [19, 51], enhancing the theory that this is an inherent nature ability and also explain how it is done simultaneously for several observed humans. This work tries to emulate intention recognition behavior as in the MN system.

In the following, just the relevant previous works, which integrate intention recognition capability into the agent's architecture, are related.

Chalupsky's work [15] on SIMBA (SIMulative Belief Ascription), implements a *Mirroring* mechanism in formal and logical language. SIMBA maintains a knowledge base, which includes the beliefs of the executed agent and the observed agent, as it believes they are. SIMBA maintains multi-reasoning contexts, one for each agent. SIMBA uses other agent beliefs in conjunction with the executed agent's reasoning to simulate the observed agent's reasoning.

Laird [39] and Tambe & Rosenbloom [60], maintain a state of the observed agent and assume that the observed agent has the same goals and tactics as the executed agent, thereby using the same behavior hierarchy, knowledge, and rules of the executed agent. Laird [39] maintains the observed agent's *state* as a substate of the executed agent, which means that the prediction of the observed agent is not done simultaneously with the executed agent performing. On the contrary RESC maintains a separated state of the observed agent; thus, the execution and the recognition can be performed simultaneously corresponding to the *Mirroring* technique.

Laird [39] predicts the observed agent just in case the executed agent will use this prediction. This intention recognition time performing corresponds to a dynamic frequency function that depends on self-cost. On the contrary RESC performs the intention recognition process all the time, corresponding to a constant high frequency.

Laird [39] stops prediction when there are a number of hypotheses. By not confronting multiple hypotheses, it reduces the computational complexity. RESC commits to a single hypothesis at a time; on one hand, it reduces the computational complexity by maintaining heuristics to commit a single hypothesis, but, on the other hand does backtracking on failure to commit another hypothesis and does not stop, as Anticipation. In contrast to the two previous works, our *Mirroring* architecture maintains multiple hypotheses for the current state;

Unlike Laird [39] and RESC [60] which perform intention recognition of one agent at a time, our *Mirroring* architecture deals with multi-agents recognition. In addition it maintains a recipe for each observed agent.

Chapter 3

Temporal Monitoring Selectivity

In this chapter we present our approach for reducing the computational load which is added when integrating the recognition process into a complete agent system that also caries out tasks while monitoring. This approach focuses on temporal monitoring selectivity, i.e. reducing the load by selecting times in which to trigger the recognition process. Section 3.1 discusses our approach of sampling the the observation stream, instead of triggering the process in every system cycle. Section 3.2 discusses constant sampling rate and our method to find the optimal static sampling rate. Section 3.3 discusses dynamic sampling method and the suggested heuristics for dynamically changing the sampling rates in order to reduce the recognition load. Section 3.4 discusses our approach when the integrated system is required to recognized multiple agents.

Hereafter this work uses the following notations: An agent architecture comprising the intention recognition capability will be termed integrated intention recognition architecture (IIRA). The primary agent that executes the intention recognition will be termed executer agent. The recognized agent will be termed observed agent.

3.1 Monitoring as sampling

A plan recognition algorithm involves a mapping from the observed action sequence into some plan representation that identifies the goal of the plan. Since there may be multiple plans (i.e., hypotheses) available to explain the observations, the recognition algorithm yields a list of potential hypotheses. One of the algorithm's key challenge is to disambiguate among those competing hypotheses to the degree possible. In principle, some degree of ambiguity is inherent and cannot be eliminated without an oracle which selects the correct hypothesis out of the set.

A block diagram of an IIRA which contains a recognition process is shown in Figure 3.1. The recognition process consists of collecting the observations by sensors and calling the recognition algorithm with those observations. The recognition process returns the system a list of hypotheses (depending on the recognition algorithm the hypotheses list might be ranked, e.g., probabilistically). The system triggers the recognition process with every system cycle.

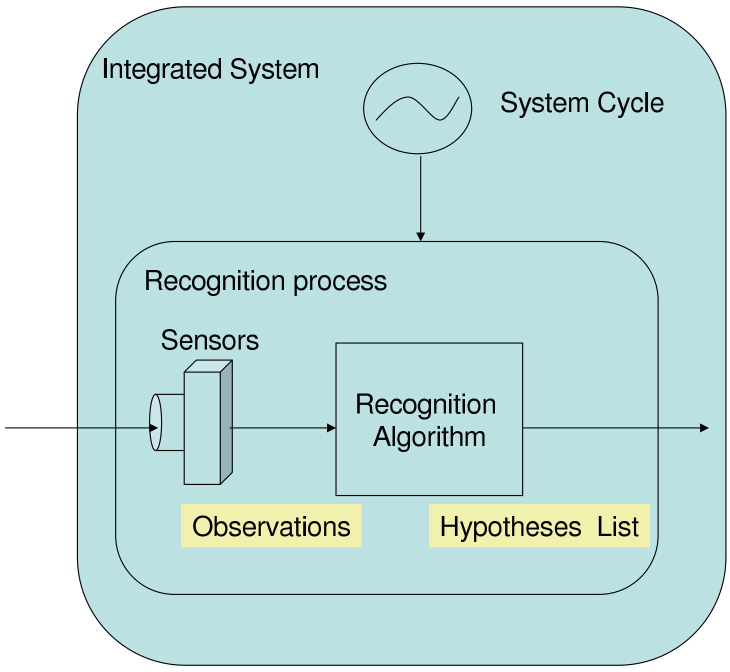


Figure 3.1: A block diagram of an IIRA using system cycle.

The algorithm utilized by the IIRA to trigger the recognition process is presented in the Algorithm 1. The IIRA calls the *Main* routine every system cycle. An IIRA needs to implement a *RecognitionProcess* routine which is called by the *Main* routine (*Main* routine, line 1 in the Algorithm 1).

Algorithm 1 Triggering recognition process with every system cycle

```
– Main routine
1: $m\_hypothesesList \leftarrow RecognitionProcess()$
– RecognitionProcess routine
1: $m\_observations \leftarrow CollectsObservations()$
2: $hypothesesList \leftarrow Activates(Algorithm(m\_observations))$
3: return $hypothesesList$
```

Our objective is to reduce the computational load, hereafter *Recognition Load*, resulting from the recognition process which is integrated in the system. To be able to measure the *Recognition Load*, we suggest measuring the load per unit time that the recognition process consumes. Measuring the load of the recognition process can be done differently depending on the system implementation, the recognition algorithm, and on the architecture capability to provide information about the system resources used during run time. For the purpose of being independent of a particular implementation, we measure load based on the size of the hypotheses list.

The size of the hypotheses list at each time point expresses the recognition load at a specific time point. Therefore, we suggest that a function of the size of the hypotheses list, can be a good indication of the recognition load of the entire process. This relies on an assumption, that the computational load of the recognition algorithm which involved in the recognition process is affected by the number of the hypotheses it maintains. Several algorithms fulfill this assumption [22, 7, 34].

We propose a formulation for the size of hypotheses list (as a measure of the recognition load). Lets $h : \mathbb{N} \rightarrow \mathbb{N}$ be the *hypotheses number function* such that $h[n]$ is the number of hypotheses maintained by the IIRA for the observed agent at time n, where: $n \in [0..N]$ denotes the time index and N is the total time used up by the recognition algorithm for the agent.

Suppose an observed agent chooses one of five different waypoints to head towards, that are positioned along a road. The executer agent, observing the other agent's movements, wants to know which waypoint the observed agent runs toward. The executer agent executes an intention recognition process. The *hy-*

potheses number function of the recognition process starts with five maintained hypotheses (one for each possible waypoint) and should decreased over time and finally converges to one (the waypoint the observed agent chose to run to), as is shown in Figure 3.2.

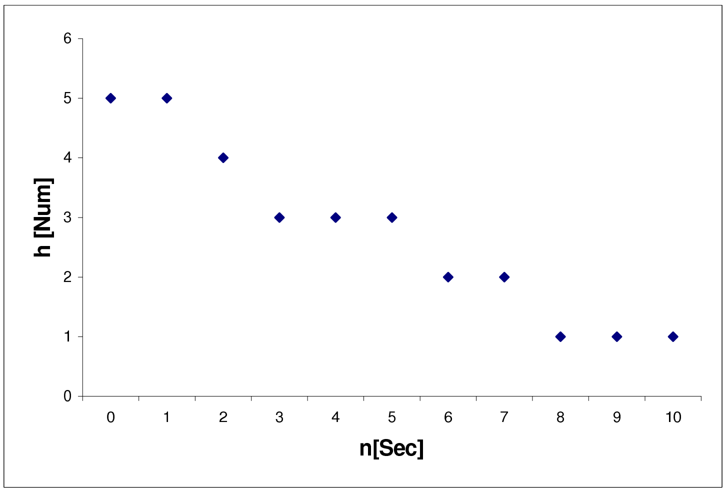


Figure 3.2: Calling recognition process every system cycle.

Activating recognition process at discrete time points by the IIRA implies a specific time lattice, consisting of the recognition activation times for observed agent. Let $L = [n_1, n_2, ..., n_J]$ be the time lattice of the agent, where $n_j \in \mathbb{N}$, $j \in \{1, ..., J\}$ are the time points when the recognition process monitors the observed agent.

In the example shown in Figure 3.2 the IIRA activates the recognition process every 1 seconds, therefore the relative time lattice is $L = [0, 1, 2, ..., 10]$, where $j \in \{1, 2, .., 11\}$ and $J = 11$. At time 2 the recognition process handles 4 hypotheses; at time 6 it handles 2 hypotheses.

The area under that function expresses the load of the recognition process over run time. Let $\overline{H}$ be the average number of hypotheses, which the recognition process maintains over the scenario run time. The value of $\overline{H}$ can be easily calculated from the *hypotheses number function*, dividing the number of the maintained hypotheses over time by the scenario runtime as follows:

$$\overline{H} = \frac{\sum_{j=1}^{J} h[L[j]]}{N} \tag{3.1}$$

Where:

h is the *hypotheses number function* of the observed agent.

J is the last monitoring time of the observed agent.

N is the total run time of the observed agent.

Definition 3.1.1. *The* Recognition Load *of the observed agent is defined as:*

$$RecognitionLoad = \overline{H} \tag{3.2}$$

In the example shown in Figure 3.2 the recognition process handles 30 hypotheses over 11 seconds. Hence, we can say that the *Recognition Load* is 2.727 hypotheses per second.

Our objective is to reduce the *Recognition Load*. One way is to reduce the amplitude of the *hypotheses number function*. Thus, reducing the size of the hypotheses list which is yielded by the recognition algorithm. RESC [60] and RESL [34] are examples of IIRA which use that way of reduction the number of hypotheses. They eliminate all hypotheses but one or two (which are selected based on hypothesis-ranking heuristics).

We allow any recognition technique to be used, such that all available knowledge is brought to bear. Even so, in the general case, the set of remaining hypotheses will include more than one hypothesis. Thus in the general case, we require that the recognition algorithm be minimal and complete in that at least one of the hypotheses is always the correct one [34, Definition 3].

Definition 3.1.2. *Given an observed agent A and any behavior B that may be executed by A, a* complete recognition algorithm *is a recognition algorithm which guarantees to include the correct hypothesis B in the set of hypotheses referred to A.*

Definition 3.1.3. *Given a set of hypotheses H referred to any observed agent A, a* minimal complete recognition algorithm *is a complete recognition algorithm which cannot eliminate any hypothesis from H unless it use an oracle.*

Under these requirements, the following theorem holds.

Theorem 3.1.4. *Eliminating an hypothesis from the potential hypotheses list eliminates the completeness of the recognition system.*

Proof. We assume that the IIRA has brought to bear any knowledge of the future behavior of the observed agent. Thus, there is no guarantee that one of the eliminated hypotheses is not the correct hypothesis. Therefore the set of hypotheses may fail to include the correct hypothesis. □

Note that the requirement of completeness is critical. If the system builder allows for incomplete recognition, then a simple solution to the recognition load issue would be to always select a single hypothesis and discard the rest.

Corollary 3.1.5. *A minimal complete recognition system, cannot reduce the amplitude of the graph of* hypotheses number function*, in order to reduce the area under it.*

Since we cannot hope to reduce the amplitude of the *hypotheses number function*, we take an alternative approach focusing on the *frequency* in which the IIRA calls the recognition process.

Theorem 3.1.6. *Activating the recognition process with the system highest frequency (every system cycle), consumes the maximal* Recognition Load.

Proof. Any activation of the recognition process consumes some resources from the system for the sake of collecting observations, feeding them into the recognition algorithm, and executing it. Using a triggering rate which is less than the highest triggering rate (i.e., system cycle) can only decrease the number of activations through run time, hence, can only decrease the resource consumption.

By using a smaller frequency from the system cycle, for activating the recognition process, the number of times of handling the observations, decreases, and therefore, the area under the graph of the *hypotheses number function* decreases as well. In other words, the *Recognition Load* on the system is reduced. □

Suppose we have the same scenario as demonstrate in Figure 3.2 but now the IIRA activates the recognition process every 2 seconds instead of every second.

The relative time lattice is $L_1 = [0, 2, 4, 6, 8, 10]$, $j \in \{1, 2, .., 6\}$ and $J_1 = 6$, as shown in Figure 3.3. At time 3 no hypothesis was handled. At time 4 the recognition process handles 3 hypotheses. Over 11 seconds the IIRA handles 16 hypotheses. Thus, the *Recognition Load* of the recognition process is 1.454 hypotheses per second, while it was 2.727 hypotheses per second when using the highest system frequency.

Decreasing the frequency of calling the recognition process reduces the amount of resources consumes from the system. This reduction in resource usage may potentially reduce the system's accuracy, as new observations may be ignored, if they take place when the recognition process is not triggered. We discuss this accuracy later in this chapter.

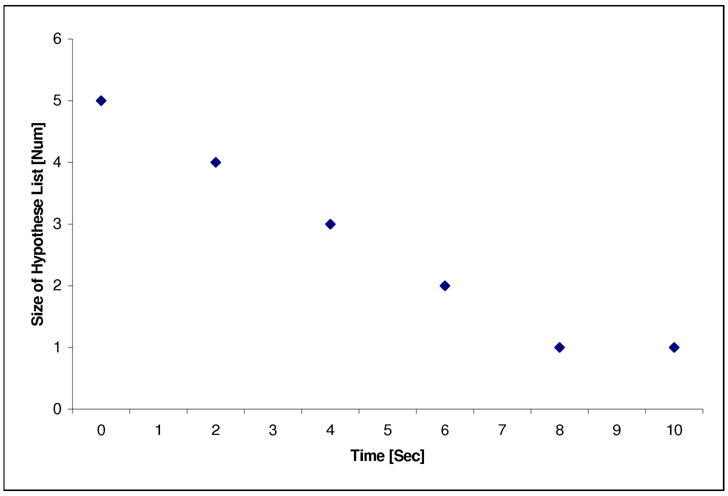


Figure 3.3: Calling recognition process every 2 seconds.

By using a smaller frequency than the system cycle, for triggering the recognition process, we are in fact proposing a sampling approach to the recognition process. The sampling frequency will be expressed by the *Sampling Interval*, hereafter F_t, the number of seconds between two calls of the recognition process by the IIRA. Let F_t^* be the *Minimal Sampling Interval*, hence, when the F_t equals to the system cycle. Figure 3.4 presents a block diagram of an IIRA which triggers the process not with every system cycle, but every F_t.

The algorithm utilized by the IIRA to trigger the recognition process is pre-

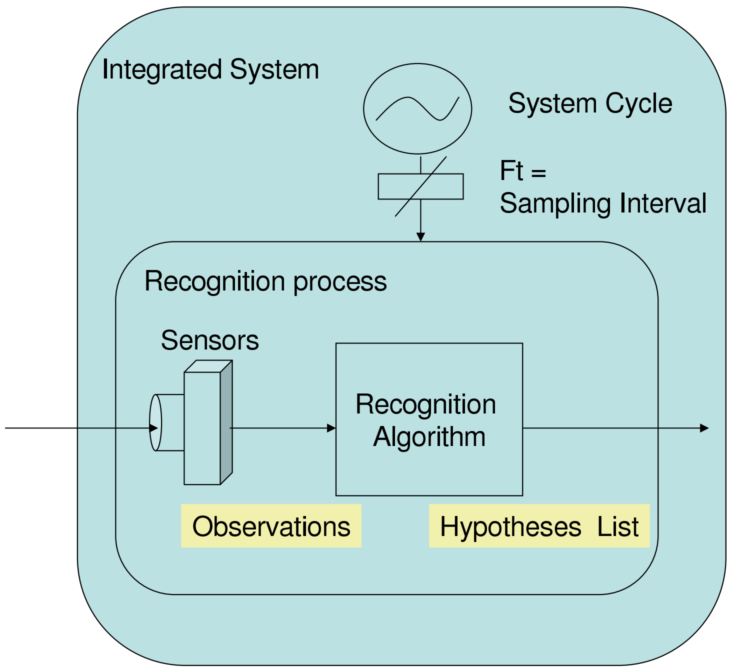


Figure 3.4: A block diagram of an IIRA using sampling interval.

sented in Algorithm 2. In order for an IIRA to use a sampling interval, F_t should be initialized and maintained as well as *duration* variable which holds the time from the last call for the recognition process. The IIRA calls the *Main* routine every system cycle. If the time exceeds the F_t, the system calls the recognition process again (*Main* routine, lines 1–2 at Algorithm 2).

When using smaller and smaller frequency we can reduce more and more the *Recognition Load*, but then we may hurt the recognition results, by missing potential hypotheses which could be returned from the recognition algorithm if we were to call it with the missed observations.

This case is demonstrated in Figure 3.5. Suppose we have the same scenario as demonstrate in Figure 3.2 but now the IIRA calls the recognition process every 3 seconds instead of every second. Because of the smaller frequency the IIRA misses the third time point, where the recognition process should returns 4 hypotheses, as can see in Figure 3.2 at time point 2.

We assume that the system's highest rate does not allow an observed agent to change its behavior without being observed. Under this assumption, the following

Algorithm 2 Triggering recognition process with sampling interval

```
– Initialization routine
1: $m\_previousTrigger \leftarrow clock()$
2: $F_t \leftarrow SamplingInterval$
– Main routine
1: $duration \leftarrow clock() - m\_previousTrigger$
2: if $duration > F_t$ then
3:     $m\_hypothesesList \leftarrow RecognitionProcess()$
4:     $m\_previousTrigger \leftarrow clock()$
– RecognitionProcess routine
1: $m\_observations \leftarrow CollectsObservations()$
2: $hypothesesList \leftarrow Activates(Algorithm(m\_observations))$
3: return $hypothesesList$
```

theorem holds.

Theorem 3.1.7. *When using the highest system rate to activate the recognition process the recognition results is complete.*

Proof. When using a sampling rate which is less than the maximal one, there is a chance to miss a change in the observed agent behavior, i.e., to eliminate an hypothesis. Based on Theorem 3.1.4, the result would be incomplete. □

There is an ideal sampling interval in which the amount of resources consumes from the IIRA is minimal while the recognition performance are maximally achieved. This ideal sampling interval triggers the recognition process with every change in the observed agent's intentions, i.e., with every change in the hypotheses list.

Figure 3.6 shows the *hypotheses number function* of the related ideal sampling interval for the one which is shown in Figure 3.2. Only five observations are required to yield the same information contained in the first *hypotheses number function*. In the case shown in Figure 3.6, the IIRA handles 15 hypotheses over 11 seconds, thus, the *Recognition Load* of the recognition process is 1.363 hypotheses per second. This sampling achieves minimal average load, while still capturing all the changes points, thus no reduction in the quality of the recognition results.

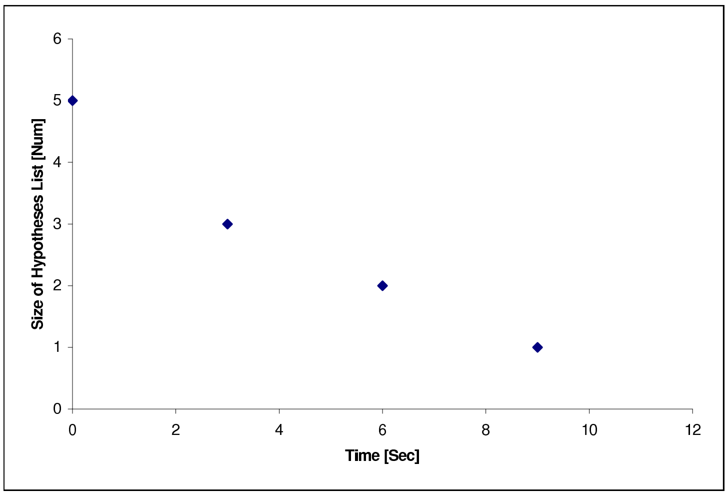


Figure 3.5: Calling recognition process every 3 seconds.

To be able to evaluate a sampling interval against the highest sampling rate, we need to measure recognition quality, by measuring *Recognition Error*. A good sampling interval is the one in which its performance approximates the highest sampling rate in terms of error, but reduces the computational load. The *Recognition Error* parameter represents how much the result of a sample is close to one achieved with the highest sampling rate. When the results are close, the *Recognition Error* of such a process should be close to zero.

An ideal recognition process recognizes all the observed agent's intentions, all the changes in the hypotheses list, ideally, at their changed time occurrences. Thus, two components must be evaluated and combined in measuring *Recognition Error*: One expresses the quality of the timing of the recognition process (how quickly does it recognize the changes); the other expresses the sensitivity to changes of the recognition process (does it recognize all the changes?).

We propose a practical common way to evaluate the *Recognition Error* parameter of a recognition process sample, by measuring the *MSE* (Mean Squared Error) between two *hypotheses number function*: One representing the evaluated sample and the other representing the system's highest sampling rate of the recognition process. The measured error expresses differences in time and the magnitude dimensions of the two *hypotheses number function*.

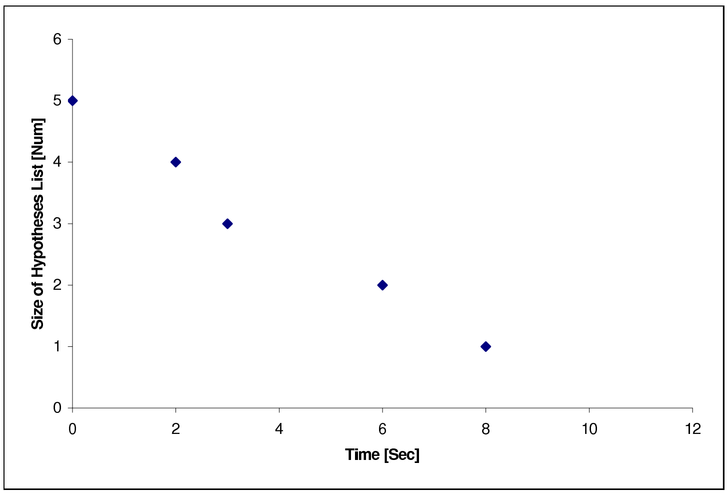


Figure 3.6: The ideal sampling interval of the example shown in Figure 3.2.

But how can we compare two samples each with a different lattice of time points?

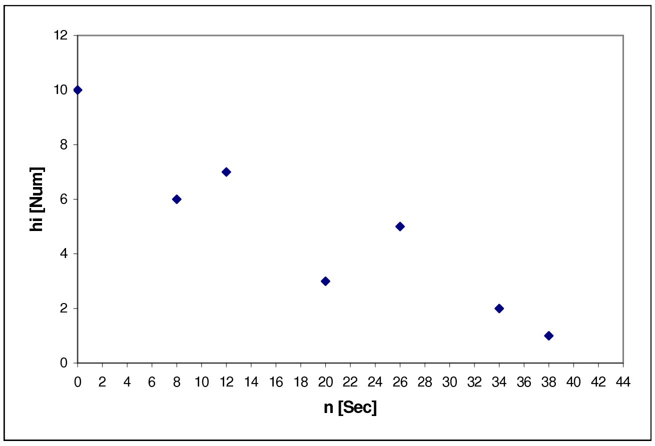
Assumption 3.1.8. *Between two activations of the recognition process we assume that the IIRA considers the last returned hypotheses list as the current correct list. Hence, the current hypotheses list related to the* hypotheses number function*, at the last time point.*

Based on the above assumption we use the Zero-Order-Hold (ZOH) interpolation of the *hypotheses number function* to expand the function over a full time points lattice (Figure 3.7). Let ZOH($h[n]$) be the Zero-Order-Hold (ZOH) interpolation of the *hypotheses number function*, defined as follows:

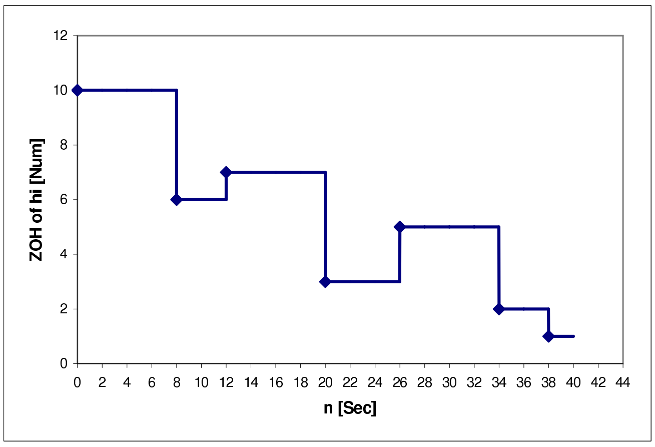
$$ZOH(h[n]) = \sum_{j=1}^{J-1} h[L[j]](U[n-L[j]]-U[n-L[j-1]]) + h[L[J]](U[n-L[J]]-U[n-N]) \tag{3.3}$$

Where:

$$U[n-n_0] = \begin{cases} 1 & \text{if } n-n_0 \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$



(a) An example of h of a sampling.



(b) ZOH of the example of h.

Figure 3.7: An example for $h[n]$ and its related ZOH($h[n]$).

Let ZOH($h^*[n]$) denote the ZOH of the *hypotheses number function* when the system uses its highest sampling rate to activate the recognition process.

Based on this, we can now compare two *hypotheses number function* using their ZOH interpolation.

Definition 3.1.9. *Let us define the* Recognition Error *to be:*

$$E = \frac{\overline{D}^2(h^*[n], h[n])}{N} \tag{3.4}$$

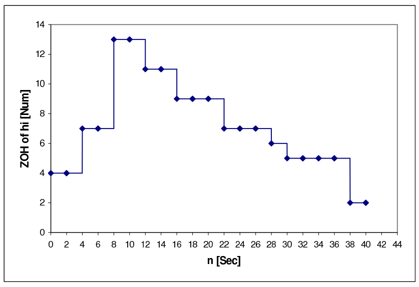
Where:

$$\overline{D}(h^*[n], h[n]) = \|ZOH(h^*[n]), ZOH(h[n])\|_2 = \sqrt{\sum_{n=1}^{N}(ZOH(h^*[n]) - ZOH(h[n]))^2} \tag{3.5}$$

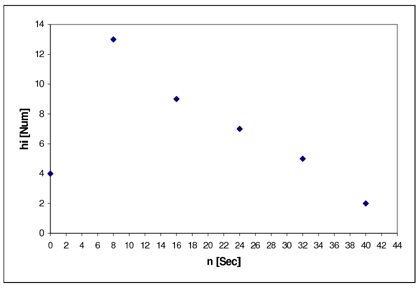
$h^*[n]$ is the *hypotheses number function* when the system uses its highest sampling rate
$h[n]$ is the *hypotheses number function* when the system uses evaluated sampling rate
N is the total time period of the signals

According to definitions of ZOH and *Recognition Error* which is defined by Equation 3.4, Figure 3.8 demonstrates the way of evaluate a sampling interval against the highest system sampling rate. Sub-figure 3.8(a) presents the ZOH of the *hypotheses number function* of a recognition process with system cycle rate (2 seconds). Sub-figure 3.8(b) presents the *hypotheses number function* of a recognition process with 8 seconds sampling interval. Sub-figure 3.8(c) presents the ZOH of the *hypotheses number function* as it shown in Sub-figure 3.8(b). Finally, Sub-figure 3.8(d) presents the differences between the two ZOH of samplings. The *Recognition Error* is easily calculated and equals to 2.2.

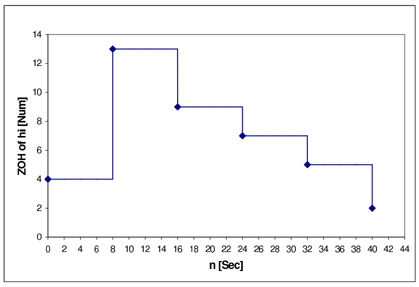
We now turn to describing the sampling which is performed just when a change in the observed agent's intention occurs, resulting in either a change in the size or contents of the hypotheses list. On one hand this sampling responds to all changes of the agent's intention. On the other hand, this process is computationally efficient. This sampling signal is called hereafter: The *Changing*



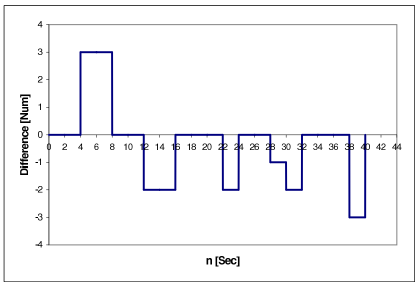
(a) ZOH(h^*).



(b) h of 8-rate sampling.



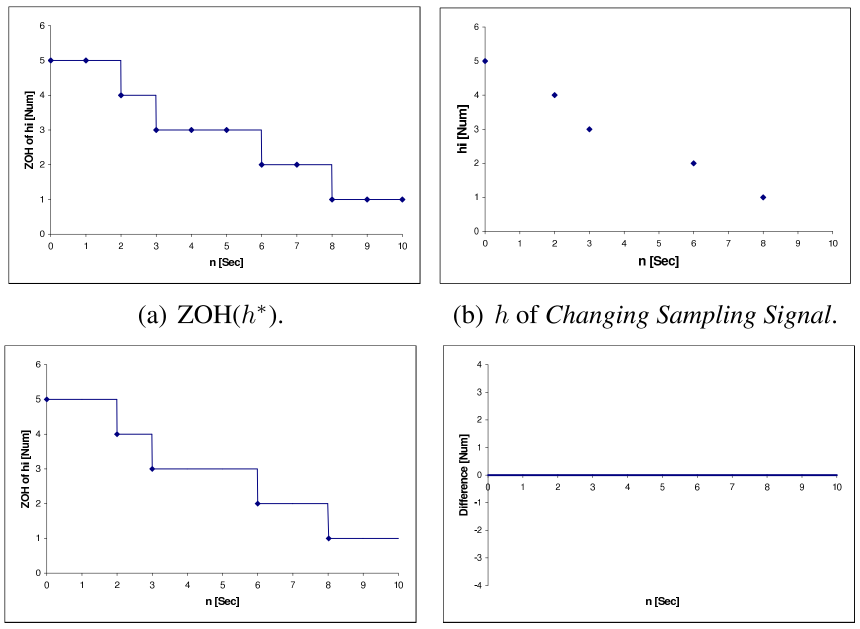
(c) ZOH(h) of 8-rate sampling.



(d) ZOH(h^*)-ZOH(h) (notice change in y-axis range).

Figure 3.8: An example for calculating a *Recognition Error*.

Sampling Signal.



(a) ZOH(h^*).

(b) h of *Changing Sampling Signal.*

(c) ZOH of *Changing Sampling Signal.*

(d) ZOH(h^*)-ZOH(h).

Figure 3.9: *Recognition Error* of the *Changing Sampling Signal.*

To be able to prove that *Changing Sampling Signal* is the ideal sampling signal, we assume:

Assumption 3.1.10. *A change in the contents of the hypotheses list, as well as a change in the size of the list are reflected by the* Changing Sampling Signal. *For simplicity, our examples for* Changing Sampling Signal*, only include changes in the size of the hypotheses list.*

Theorem 3.1.11. *The* Changing Sampling Signal *has an ideal sampling rate.*

Proof. Assume for contradiction that the *Changing Sampling Signal* does not have the ideal sampling rate. Hence, at least one of the following cases occurs:

- Case 1: The *Changing Sampling Signal* does not achieve the maximal recognition results, as the system highest sampling signal achieves. Based on the definition, the *Changing Sampling Signal* detects all the changes as they occur when using the highest system sampling rate. The ZOH of an *hypotheses number function* is an interpolation which keeps on the previous value until the value is changed, thus the ZOH signal is changed just at the changed points of the *hypotheses number function*. The *hypotheses number function* of the *Changing Sampling Signal* includes only the changed points of the highest sampling signal, thus the related ZOH performs the same ZOH of the highest sampling signal. According to Equations 3.4 and 3.5 the distance between two identical ZOH is zero, hence, the *Recognition Error* of the *Changing Sampling Signal* is equal to zero. Thus, the *Changing Sampling Signal* achieves the same recognition results as the highest system sampling rate does, which performs the maximal outcomes, according to Theorem 3.1.7.

- Case 2: Assume for contradiction that the *Changing Sampling Signal* does not consumes minimal resources. Hence, there is another sampling signal, S', which utilizes minimal resources with the same recognition results. Its relative *hypotheses number function*, $h[n]$, has a reduction in either the amplitude (i.e., S' results in a smaller number of hypotheses in at least one time point) or it triggers the recognition process at different times (and possibly less times).

  In the case of reduction in the amplitude, Corollary 3.1.5, implies that the system is not minimally complete, hence this case is impossible.

  In the case of different sampling times, there exists at least one time point n_0 which is different between S' and the *Changing Sampling Signal*. For n_0, either it misses a change in the observed agent's intention (and therefore its recognition performance is less than the *Changing Sampling Signal*), or n_0 does not express a change in the agent's intention, hence, this sample is a redundant one and can be thrown. If the sample is thrown, its means that the system has a prior knowledge that this sample is redundant compared to the *Changing Sampling Signal*. Since we assume that our system does not have a prior knowledge, this is also impossible.

Therefor, the *Changing Sampling Signal* has the ideal sampling rate. □

We cannot sample the recognition process with the ideal sampling rate because we cannot predict when exactly the number of the potential hypotheses might be changed. Thus, in the next sections we present methods to approximate the ideal sampling rate.

3.2 Fixed sampling interval

In general, there is no way to know the *Changing Sampling Signal* because the executer cannot predict the time points the observed agent might change its plans. Therefore, we look for methods to approximate the ideal sampling rate. At this section we will discuss methods to approximate the ideal sampling rate when using fixed sampling rates. A fixed sampling rate means using a constant rate of recognition.

As we defined already, F_t stand for the sampling interval (in seconds). F_t is constant, when using fixed sampling interval. Thus the recognition process is triggered every pre-determined interval. Where $F_t = F_t^*$ the system then uses the system highest rate sampling and is essentially equivalent to RESL [34], which maintains knowledge of all individual hypotheses, for all agents, at all times.

By determining the sampling interval parameter, the time lattice of a recognition process can be controlled. A block diagram of such an IIRA which triggers the recognition process every fixed sampling interval is shown in Figure 3.10. The algorithm utilized by the IIRA to trigger the recognition process is identical to the one which is presented in Algorithm 2.

As it was shown in the previous section it is obvious that when a system using a smaller sampling rate than its system highest sampling rate the amount of resources consumes from the system is decreased, though possibly at the cost of quality (recognition results). Using any smaller sampling rate (than the system cycle) decreases the amount of resources.

We want to find an optimal fixed sampling interval according to recognition performance and available system resources. To do this, we present an analytic model of the IIRA when using fixed sampling rates. The model estimates two system variables. The first one is the computational load resulting from activating the recognition process (the *Recognition Load*, Equation 3.2). In addition, our objec-

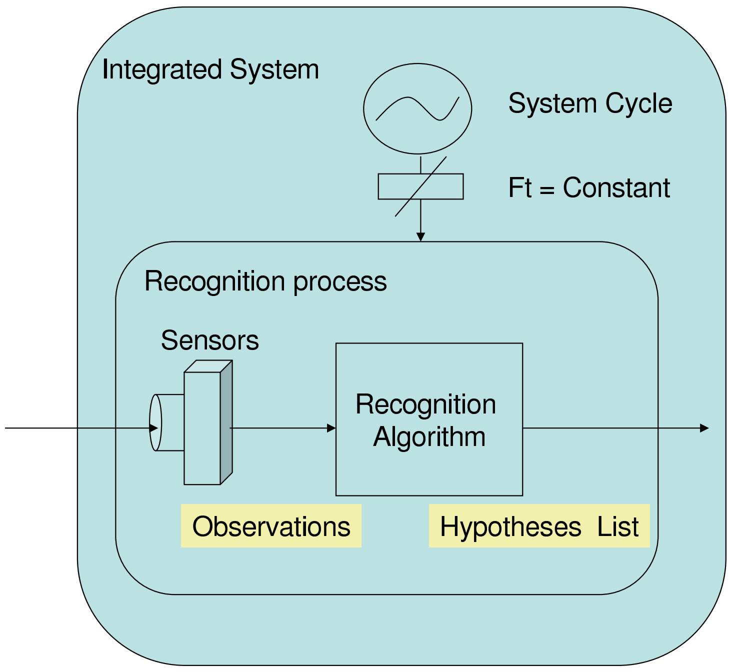


Figure 3.10: A block diagram of an IIRA using fixed sampling interval.

tive is not just to decrease the *Recognition Load* but to decrease it while satisfying the recognition quality as a system requirement. Hence, the second variable is the *Recognition Error* measure (Equation 3.4). After we have a models for *Recognition Load* and *Recognition Error* as functions of fixed sampling interval, we can analytically find the fixed sampling interval that achieves the optimal performance for the two system criteria as required from the system.

This section deals with fixed sampling interval, therefore, the primal parameter of the model that affects the *Recognition Load* and the *Recognition Error* is the time between two consequent samples. Two theoretic models for the *Recognition Load* and for the *Recognition Error* are discussed in the following subsections.

3.2.1 *Estimated Recognition Load* model for the fixed sampling interval

The objective of the *Temporal Monitoring Selectivity* approach is to reduce the computational load, *Recognition Load*, consumed from the agent computational resources (*Computational Load*) for the recognition process:

$$\textit{Computational Load} = \textit{Execution Load} + \textit{Recognition Load} \tag{3.6}$$

We suggest the following analytic model for estimating of *Recognition Load*. We use the widehat notation to distinguish the estimation model from the value that is measured by Equation 3.2. The model is calculated as a function of the fixed sampling interval:

$$\textit{Estimated Recognition Load} = \widehat{RL} = \frac{r}{F_t} \tag{3.7}$$

Where:

Computational Load is the total load on the executer
F_t stands for the sampling interval
r stands for the computational load generated by the recognition process.

An executer agent consumes *Computational Load* from the computer at every time unit. This load is produced by two loading sources: the *Execution Load*, which is the computational load generated by the execution process of the agent, and the *Recognition Load*, which is the computational load generated by the recognition process of the agent. According to our approach the recognition process is no longer triggered at every system time unit, but at a fixed sampling interval. Therefore, the computational load per time unit, resulted by the recognition process, can be calculated as its computational load divided by the sampling interval, hence $\frac{r}{F_t}$.

Equations 3.6 and 3.7, imply that the *Estimated Recognition Load* vanishes when the sampling interval approaches infinity and the *Computational Load* converges to the *Execution Load*.

3.2.2 *Estimated Recognition Error* model for the fixed sampling interval of a linear decreasing signal

We now turn to defining an analytic model for the *Recognition Error* imposed by applying a fixed sampling interval. Evidently, the fixed sampling interval affects the data in a way similar to the downsampling operator in signal processing, such as in the Nyquist and Shannon theorems (signal processing techniques that are described in [47]).

However, utilizing signal processing techniques is highly dependent on several restrictive assumptions that are not always correct in our case. Nyquist and Shannon theorems were originally stated on continuous signals. To avoid misleading implementation of these techniques, one should assume that the number of hypotheses is large enough to neglect the quantization effect, in other words, that the signal is smooth.

Since the list of hypotheses, which is returned by the recognition algorithm, cannot be assumed to include a large number of hypotheses (the recognition algorithm is minimal complete). We do not focus on smooth signals. Further investigation into the smooth case is beyond the scope of this work, but it appears to be a promising direction to exploit downsampling error analysis techniques for monitoring optimization.

For a small number of hypotheses signal (which is our case), a better approximation can be obtained when a number of typical characteristics are defined. Such characteristics are: A steady state, a linear decreasing signal and a linear increasing signal. When dealing with few hypotheses, most of the signals can be approximated to a degree by a combination of these three characteristics. The first characteristic is degenerate and has a trivial solution of no error no matter what the sampling interval is. The second and the third characteristics are analogous, and entail the same model derivation. We therefore present a model for one of these.

We now discuss a simplified signal case, of a linear decreasing characteristic of the *hypotheses number function*. In this subsection an analytical model for the linear decreasing signal is presented. This model is divided into two parts: One is when the sampling interval is larger then the time between two consequent changes in the number of hypotheses, and one is when the sampling interval is strictly smaller then the time between two consequent changes in the number of hypotheses. We show in Section 5.2.2 that this model can also be an approximation for a non-linear decreasing signal.

A linear decreasing signal is the form describing a linear decreasing function of the number of hypotheses over time. An example of a linear decreasing signal is shown in Figure 3.11. The signal starts with 6 hypotheses and every four seconds the number of hypotheses is decreased by one. The linear signal contains six changes in its hypotheses number over 24 seconds, Therefore the width of each change is $\frac{24}{6} = 4$ second.

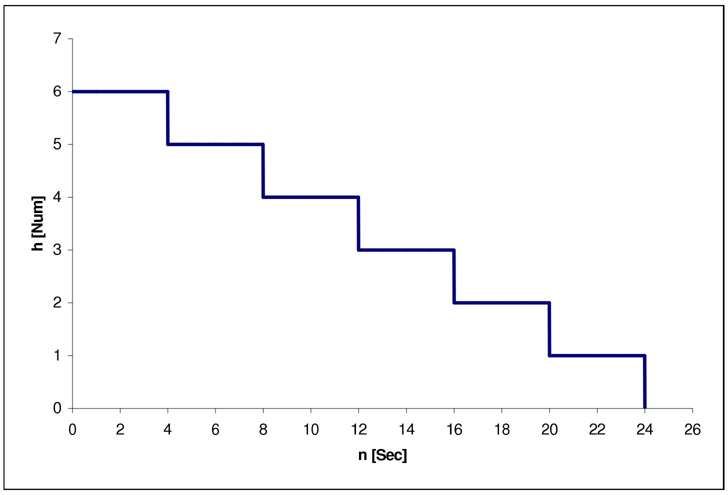


Figure 3.11: An example of a linear decreasing signal.

In order to develop the error model for the linear decreasing signal, several parameters should be defined:

- F_t denotes the sampling interval, the number of time units between consequent samples.
- C_t denotes the average of time units between changes (how fast the signal changes).
- N denotes the total time of the signal.

- H denotes the total number of hypotheses changes over N.
- $k = \frac{F_t}{C_t}$ denotes a normalized sampling interval.
- ϕ denotes the signal phase (see below).

Clearly, in a linear decreasing scenario C_t can be calculated according to:

$$C_t = \frac{N}{H} \tag{3.8}$$

Substituting 3.8 in the definition of k yields:

$$k = \frac{F_t}{C_t} = \frac{F_t H}{N} \tag{3.9}$$

When analyzing a signal and its relative sampled signals the **phase** term ϕ must be taken into consideration. There is no absolute time at which all the signals start, therefor the phase expresses the associated starting point of each of the signals. In most cases the phase is measured from the starting point of a reference signal. In our work the reference signal is chosen to be the one that was generated with the highest system rate.

Two separate linear models are described: One for when the sampling interval F_t is higher than the time between two consequent changes in the number of hypotheses C_t ($k \geq 1$); and one for when the sampling interval is strictly smaller than the time between two consequent changes in the number of hypotheses ($k < 1$).

3.2.2.1 The $k \geq 1$ case

As aforesaid, in this section we deal with linear decreasing signals. For intuition purpose, in the following examples we assume a signal that is decreased by one hypothesis every one second. Figures 3.12 to 3.14 display sampled signals with different sampling intervals. In each of the figures three curves are plotted. The curve with the triangle points depicts sampling with phase $\phi = 0$. The curve with the square points depicts sampling with phase $\phi = 0.333$. The curve with the diamond points depicts sampling with phase $\phi = 0.666$.

In this work, the MSE was chosen for calculating the error between two signals (Equation 3.4). It is calculated by multiplying the squares of the differences by the time period over which the difference lasts, and normalizing by dividing the result by the time period of the signals.

Figure 3.12 shows the sampled signal when sampling the signal with $k = 1$. As can be seen in Figure 3.12 the error is affected only by the difference in phase and is equal to $E = \frac{1^2\phi}{1}$. Figure 3.13 shows the sampled signal when sampling the

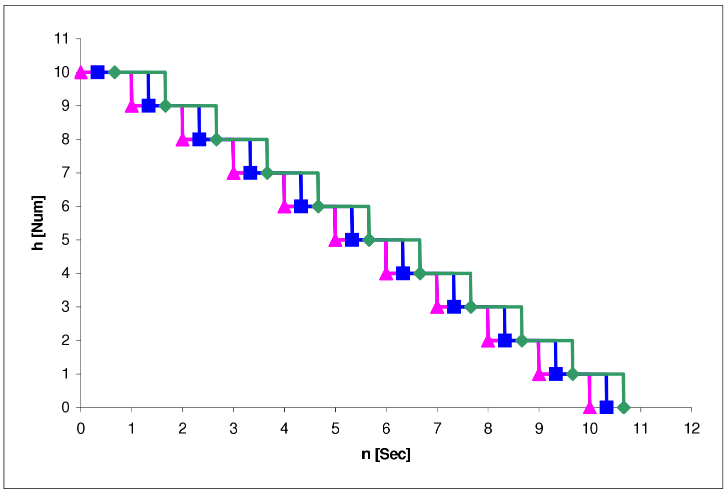


Figure 3.12: An example of a sampled signal with k=1.

signal with $k = 2$. As can be seen in Figure 3.13 the error is $E = \frac{1^2+2^2\phi}{2}$ the first component in the numerator results from the sampling error, the second results from the phase error and the denominator expresses the period of time over which the error is calculated.

Figure 3.14 shows the sampled signal for $k = 3$ and the error is $E = \frac{(1^2+2^2)+3^2\phi}{3}$.

The widehat notation is used to distinguish the estimation model from the value that is measured by Equation 3.4.

Theorem 3.2.1. *A general equation for estimating the* Recognition Error *as a functions of k and ϕ for the $k \geq 1$ case is:*

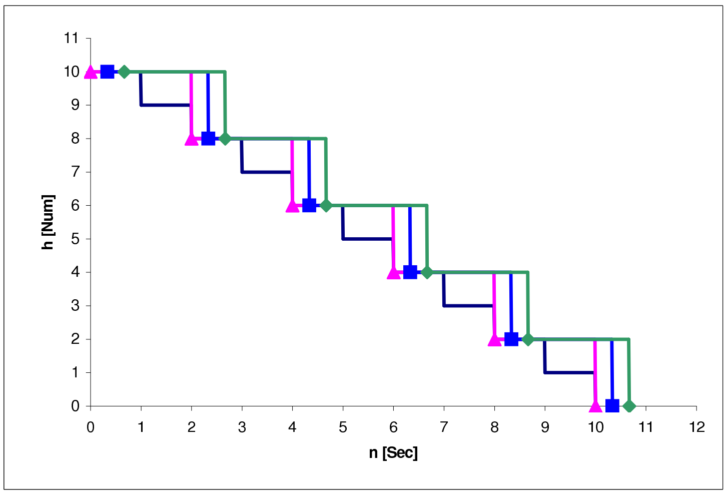


Figure 3.13: An example of a sampled signal with k=2.

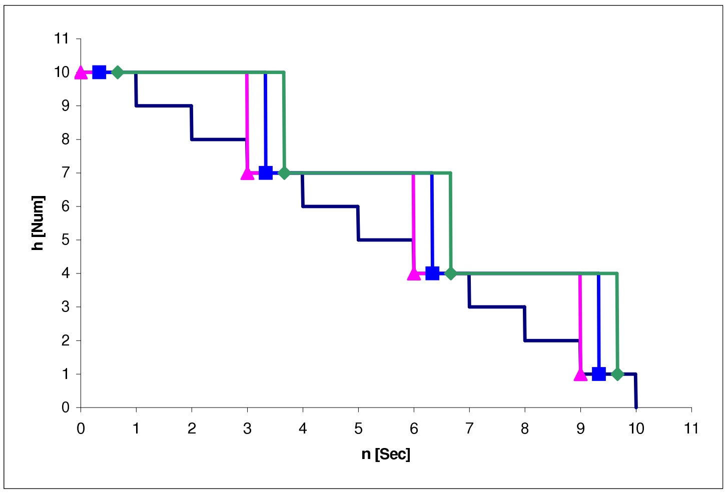


Figure 3.14: An example of a sampled signal with k=3.

$$EstimatedRecognitionError = \widehat{RE}(k, \phi) = E_0 + \frac{2k^2 - 3k + 1}{6} + k\phi \quad (3.10)$$

Where:

$k \geq 1$

$0 \leq \phi \leq 1.$

E_0 *is the initial error.*

Proof. The *Estimated Recognition Error* is composed of three parts: One results from a sampling error, one results from a phase error, and one results from initial conditions. In this case $k \geq 1 \Rightarrow C_t \leq F_t$, therefore, sampling causes missing changes. The number of missing changes is depend on k and equals to $k - 1$. For example, in Figure 3.14 k is 3 and the number of missing changes is 2.

The square of the error imposed by one missing change is $1^2 \times C_t$. The square of the error imposed by two missing changes is $2^2 \times C_t$, and the square of the error imposed by the n-th missing change is $n^2 \times C_t$. The sum of the squares of the errors over one sample is $\sum_{n=1}^{k-1} n^2 \times C_t$, and the MSE is therefore:

$$\frac{\sum_{n=1}^{k-1} n^2 \times C_t}{F_t} = \frac{\sum_{n=1}^{k-1} n^2}{k} \quad (3.11)$$

Since:

$$\sum_{n=1}^{k-1} n^2 = \sum_{n=1}^{k} n^2 - k^2 = \frac{k(k+1)(2k+1)}{6} - k^2 = \frac{k(2k^2 - 3k + 1)}{6} \quad (3.12)$$

the sampling error is: $\frac{2k^2-3k+1}{6}$.

The second error component resulting from the phase error, contributes an error of k that lasts ϕ seconds. Thus, the MSE of the phase is : $\frac{k^2\phi}{k} = k\phi$.

The third error component is the constant E_0 resulting from the initial condition of an IIRA which has a minimal sampling interval ${F_t}^*$, and will therefore vary between systems. When the system uses its minimal sampling interval the *Estimated Recognition Error* is expected to be zero, i.e., $\widehat{RE}|_{F_t = {F_t}^*} = 0$. In order to meet this requirement, and since the sampling error component for ${F_t}^*$ is zero, E_0 needs to be $-k\phi$. $\square$

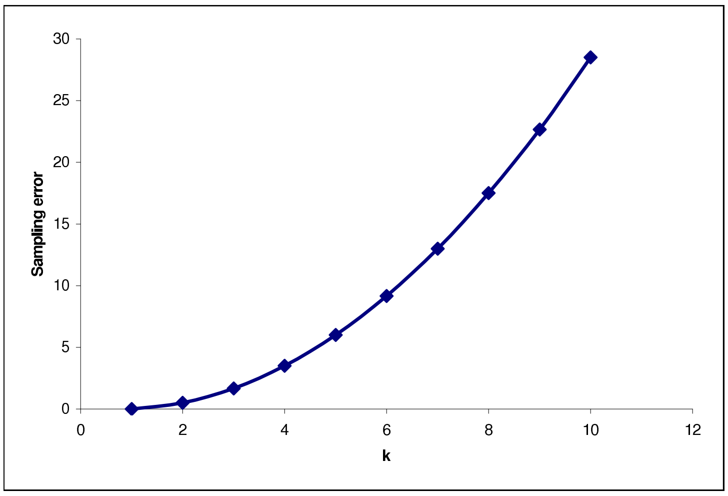


Figure 3.15: The *Estimated Recognition Error* model as a function of $k \geq 1$.

A graph describing the sampling error for $k \geq 1$ (the second term of Equation 3.10) is shown in Figure 3.15. The *Estimated Recognition Error* is polynomial in k, hence, reducing the *Recognition Load* by increasing k by a factor 2, increases the *Estimated Recognition Error* by factor of 4. That trade-off must be carefully considered by the system in the light of its cost.

3.2.2.2 The $k < 1$ case

Theorem 3.2.2. *When $k < 1$, the phase is in range of* $[0, k]$.

Proof. When $k = \frac{F_t}{C_t} < 1 \Rightarrow F_t < C_t$. Suppose that a change occurs at time τ_0 and the last sampling before that change occurs at time τ. The phase between the signals can be either one of the following cases:

- $\tau_0 - \tau = k$, hence the next sampling will occur at time $\tau + k = \tau_0$, when the change occurs, therefore $\phi = 0$.

- $\tau_0 - \tau = \varphi$, where $\varphi < k$. $k = \varphi + (k - \varphi)$ is always correct. The next sampling will occur at time $\tau + k = \tau + \varphi + (k - \varphi) = \tau_0 + (k - \varphi) < \tau_0 + k$, and therefore $\phi < k$.

- $\tau_0 - \tau = 0$, hence the next sampling will occur at time $\tau_0 + k$, k times after the change occur, therefore $\phi = k$.

$\square$

Figure 3.16 shows the three phase's ranges described above. The curve with the triangle points demonstrates sampling with $\phi = 0$. The curve with diamond points demonstrates sampling with $\phi = k$ and the curve with the square points demonstrates sampling with $0 < \phi < k$.

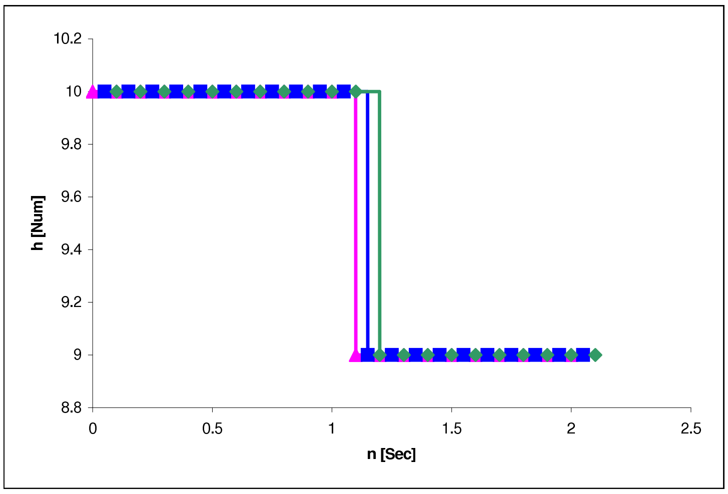


Figure 3.16: An example of a sampled signal with $k < 1$.

Theorem 3.2.3. *When the time between changes, in a signal, is much greater than the sampling interval, the sampled signal does not miss a change.*

Proof. As was shown above, for $k < 1$ the phase between the sampled signal and its signal is $\phi\epsilon[0, k]$. Hence, the next sample will occur in at most k seconds after the change. Since $k < 1$, $C_t > F_t$, and the next sample is taken before another change takes place. In other words the sampled signal does not miss a change. $\square$

Based on Theorem 3.2.3, the *Estimated Recognition Error* is only affected by the phase error and the initial condition of an IIRA which has a minimal sampling interval. For intuition purpose, in the following example we assume a signal that is decreased by one hypothesis every one second. The phase error contributes differences value which equals to 1 (hence, the 1^2 in the numerator), lasts ϕ seconds and occurs once in a change, which lasts one second (hence, the 1 in the denominator). Therefore, the *Estimated Recognition Error* model is reduced to:

$$\textit{Estimated Recognition Error} = \widehat{RE}(\phi) = E_0 + \frac{1^2\phi}{1} = E_0 + \phi \tag{3.13}$$

Where

$k < 1$

$0 \leq \phi \leq k$

E_0 is the initial error.

In order to meet the requirement that $\widehat{RE}|_{F_t = F_t{}^*} = 0$, E_0 needs to be $-\phi$.

Assumption 3.2.4. *The phase ϕ is uniformly distributed in its range $[0, k]$. The intuition behind the assumption is that there is no likely value for ϕ, since the starting point of the signal only depends on an arbitrary time when the recognition process starts. Therefore the averaged value of the phase is $\phi = \frac{k-0}{2} = \frac{k}{2}$.*

We believe that the average phase error as function of k is a good error model for the linear decreasing signal (for the $k < 1$ case). The estimated error model for the linear decreasing signal as a function of k is:

$$\textit{Estimated Recognition Error} = \widehat{RE}(k) = E_0 + \phi = E_0 + \frac{k}{2} \tag{3.14}$$

Where:

$k < 1$

E_0 is the initial error

In order to meet the requirement $\widehat{RE}|_{F_t = F_t{}^*} = 0$, E_0 needs to be $-\frac{k}{2}$.

A graph describing the recognition error as a function of $k < 1$ is shown in Figure 3.17. In this example, the error model represents a system with $F_t^* = 2$ seconds and the linear decreasing signal is characterized by $H = 9$ and $N = 136$. Therefore, $E_0 = -\frac{k}{2}|_{F_t{}^*} = -\frac{F_t{}^* H}{2N} = -\frac{2 \times 9}{2 \times 136} = -0.066$. We can see in Figure

3.17 that the *Estimated Recognition Error* is linear in k. It equals zero when using the minimal k.

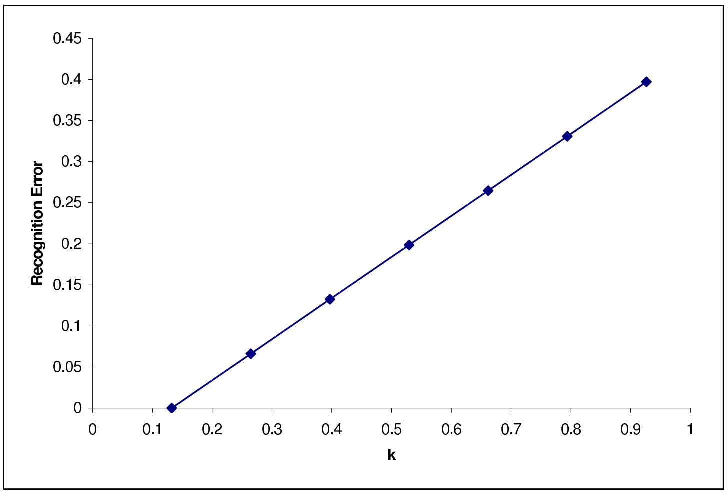


Figure 3.17: The *Estimated Recognition Error* model as a function of $k < 1$.

Substituting $k = 1$ in Equation 3.10 the error is: $\widehat{RE} = E_0 + \frac{2k^2-3k+1}{6} + k\phi = E_0 + \frac{2-3+1}{6} + \phi = E_0 + \phi$. Substituting $k = 1$ in Equation 3.14 the error is: $\widehat{RE} = E_0 + \phi$. Therefore the two models are continuous at $k = 1$.

3.2.3 Estimated models for the fixed sampling interval of a non-linear decreasing signal

Since there is no way to force a linear tendency in the behavior of the observed agent, a more suitable way to describe the behavior of the observed agent for a decreasing signal scenario is by the monotonic decreasing signal or by the non-monotonic decreasing signal.

A monotonic decreasing signal is a signal whose *hypotheses number function* always decreases but not necessarily at a constant rate. An example for such a signal is shown in Figure 3.18. The first decrease occurs after two seconds, and the second decrease occurs after four additional seconds.

A non-monotonic signal is a signal where at the end of the run time its number of hypotheses is less than it was at the beginning of the run, but its *hypotheses*

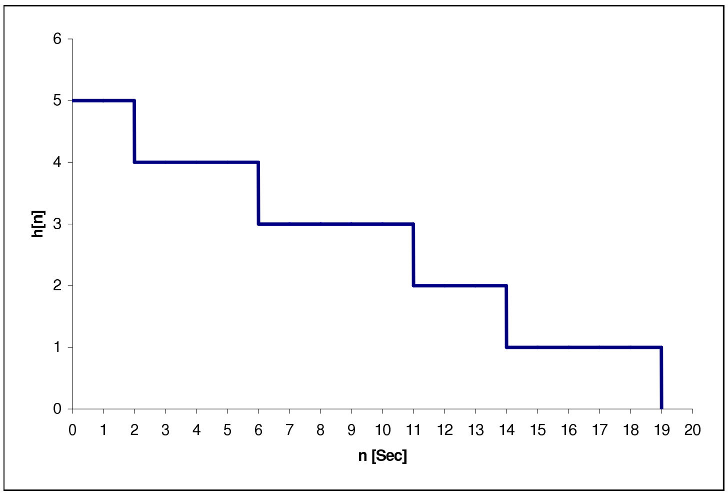


Figure 3.18: An example of a monotonic decreasing signal.

number function does not always decrease over all the run time. An example for such a signal is shown in Figure 3.19. As can be seen, at the first change the number of the hypotheses is reduced from 5 to 3 but at the second change the number is going up from 3 to 4. At the end of the run time the number of hypotheses is 2, less than at the beginning, which starts with 5 hypotheses.

Our objective is to try out the models (described in Sections 3.2.1 and 3.2.2) for the monotonic and non-monotonic decreasing *hypotheses number function* signals. We empirically show in Sections 5.2.1 and 5.2.2 that these models in some cases can be useful as an approximation for these signal's performance and in some cases can be useful as a lower bound for their performance.

3.3 Dynamic sampling

The *hypotheses number function* of the recognition process can be thought of as a stochastic signal. As the observed agent changes its behavior there is no way to perform an ideal sampling rate for a stochastic signal without having a prior knowledge of the future behavior of the observed agent.

A common approach to sample a stochastic signal is using heuristics [6, 63, 3, 45, 46, 5, 4]. In this section we present heuristics to approximate the ideal

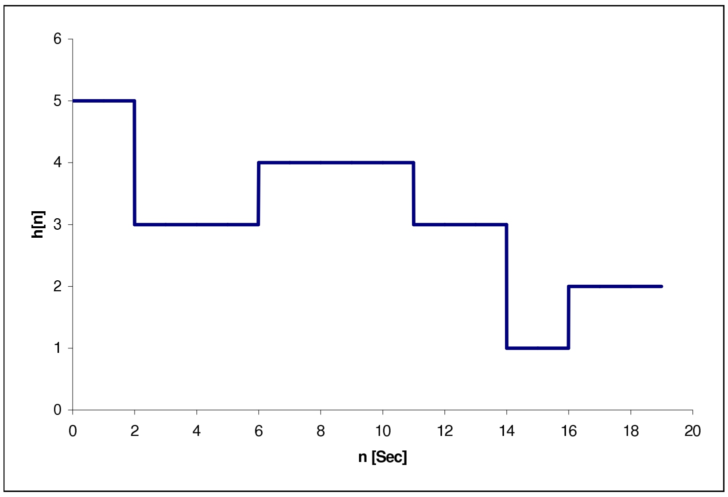


Figure 3.19: An example of a non-monotonic decreasing signal.

sampling rate in order to reduce the *Recognition Load* while keeping of a required *Recognition Error*. To evaluate the use of the heuristics, their performances are empirically evaluated and compared to the performances of the fixed sampling intervals (Chapter 5.2.3).

Definition 3.3.1. *A* dynamic sampling method*, is one where the sampling interval F_t varies in time, and can be denoted as $F_t[n]$.*

A block diagram of an IIRA which triggers the recognition process according to a dynamic sampling interval is shown in Figure 3.20. As can be seen, the sampling interval is dynamically changed according to new knowledge derived from the recognition process.

The algorithm utilized by the IIRA to trigger the recognition process is presented in Algorithm 3. The IIRA calls the *Main* routine every system cycle. In order for the IIRA to use a dynamic sampling interval, an *UpdateInterval* routine which returns a new value for the sampling interval should be implemented, as well as a *DynamicFlag* which enables the routine. The IIRA calls the *UpdateInterval* routine upon arrival of new information about the hypotheses list from the recognition process and if the *DynamicFlag* is activated (line 4 in the *Main* routine in Algorithm 3).

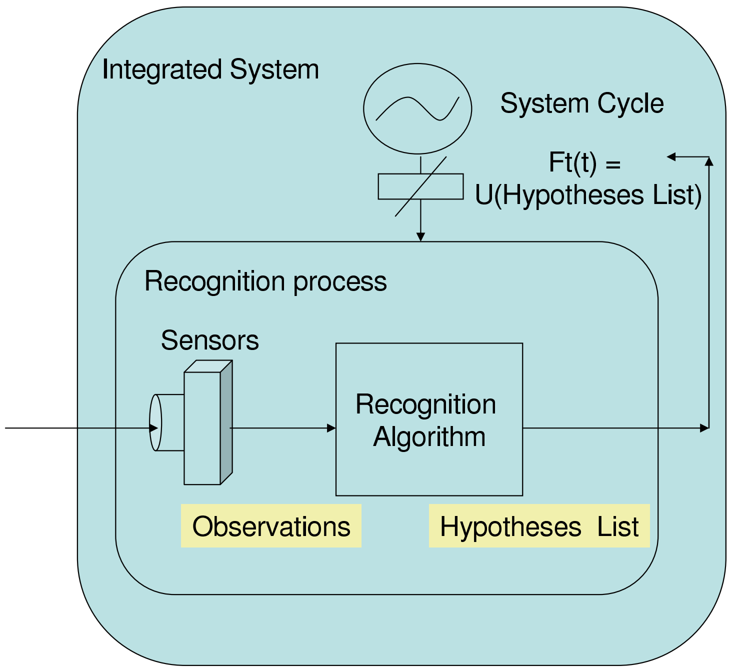


Figure 3.20: A block diagram of an IIRA using dynamic sampling interval.

In this section two fundamental heuristics which dynamically change the F_t will be suggested. Both are based on the entropy of the hypotheses list and thus we discuss entropy first.

3.3.1 What is Entropy?

Entropy, borrowed from information theory, is a quantitative measure of variance in a distribution over categorial data. This can be useful for measuring the uncertainty represented by distribution.

Relying on the *Information Theory*, we suggest referring to entropy as a measure of the uncertainty in the observed agent's intention. The uncertainty is expressed by the multiplicity in the number of hypotheses which potentially can be the agent's state. The size of the hypotheses list, which is maintained by the recognition process, can be a plausible indication on the current uncertainty level of the recognition system about an observed agent's state. Therefore, we suggest using the entropy of the *hypotheses number function* for indicating the uncertainty

Algorithm 3 Triggering recognition process with dynamic interval

```
– Initialization routine
1: $m\_previousTrigger \leftarrow clock()$
2: $F_t \leftarrow SamplingInterval$
3: $m\_dynamicFlag \leftarrow DynamicFlag$
– Main routine
1: $duration \leftarrow clock() - m\_previousTrigger$
2: if $duration > F_t$ then
3:     $m\_hypothesesList \leftarrow RecognitionProcess()$
4:     if $sizeChanged(m\_hypothesesList) \cap m\_dynamicFlag$ then
5:         $F_t \leftarrow UpdatesInterval(m\_hypothesesList)$
6:     $m\_previousTrigger \leftarrow clock()$
– RecognitionProcess routine
1: $m\_observations \leftarrow CollectsObservations()$
2: $hypothesesList \leftarrow Activates(Algorithm(m\_observations))$
3: return $hypothesesList$
– UpdateInterval routine
1: $newInterval \leftarrow U(hypothesesList)$
2: return $newInterval$
```

in the agent's hypothesized state.

Consider a recognition algorithm yielding an hypotheses list with a finite number of hypotheses $0 \leq h[n] \leq H$, where $h[n]$ is the value of the *hypotheses number function*, expresses the size of the hypotheses list at time unit n, and H is the maximal size of the *hypotheses number function*. The hypotheses list is returned with an associated probability distribution, $P[n] = (p_1, p_2, ..., p_{h[n]})$. The probability distribution has to satisfy the following properties: $\sum_{i=1}^{i=h[n]} p_i = 1$ with $p_i \geq 0\ \forall i$. According to Shannon's information measure, the entropy of this state is defined by:

$$S(p_1, p_2, ..., p_{h[n]}) = -\sum_{i=1}^{i=h[n]} p_i log_2(p_i) \qquad (3.15)$$

A simple case is when all the potential hypotheses derived by the recognition algorithm have an equal probability of $1/h[n]$ as in the case of symbolic recognition algorithms, such as [7]. In this case, the entropy of the state is:

$$S[n] = log_2(h[n]) \tag{3.16}$$

Let S_{max} be the entropy of the state which yields an hypotheses list with the maximal size H. A graph of the entropy of the process as a function of list size, in the case of equal probabilities, is shown in Figure 3.21. The entropy of the process increases with the number of potential hypotheses.

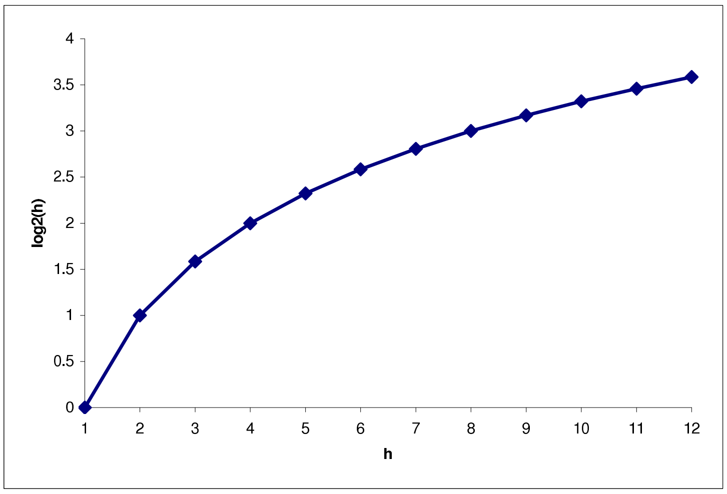


Figure 3.21: Entropy as a function of the number of hypotheses.

Based on the entropy definition above, we introduce our heuristics in the next subsections. The sampling interval of the observed agent will be dynamically changed according to different functions of its hypothesized state's entropy.

3.3.2 Uncertainty heuristic

The basic idea behind this heuristic is to ease the uncertainty of the agent's state, or in other words, to decrease the entropy of the recognition process. In order to decrease the entropy, we need to increase the sampling rate of the recognition process by increasing the amount of resources consumed by the system for the recognition process. Thus this heuristic responds to greater uncertainty (larger number of hypotheses) by increasing the sampling rate (or decreasing the sampling interval). The intuition is that when uncertainty grows, or the number of hypotheses increases, the sampling rate is increased, to give the agent more chance to reduce the uncertainty. The relationship between sampling interval and the entropy is given by:

$$F_u \propto \frac{1}{S} \tag{3.17}$$

Where S is the entropy of the state.
We suggest the following function to control F_t during run time:

$$F_u[n] = (2^{S_{max}} - 2^{S[n]})F_t^* + F_t^* = (H - h[n])F_t^* + F_t^* \tag{3.18}$$

Where:
F_t^* is the minimal sampling interval of the IIRA
H is the maximal size of the hypotheses list of the scenario
$h[n]$ is the current size of the hypotheses list
$S[n]$ is the current entropy
S_{max} is the entropy when $h[n] = H$

When the size of the hypotheses list is maximal, the uncertainty of the agent's intention is maximal, hence, the entropy is maximal. As a result, the sampling interval is set to a minimal value. Upon decreasing the number of hypotheses by one, the sampling interval is increased by F_t^*.

We demonstrate this heuristic on a system having $F_t^* = 2$ and on the scenario which is presented in Figure 3.19, thus $H = 5$. The scenario starts with the maximal size of hypotheses list, therefore, the sampling interval is set to F_t^* according to Equation 3.18: $F_u[0] = (H - h[0])F_t^* + F_t^* = (5 - 5)2 + 2 = 2$. The F_u stays constant until the next change in the size of the list occurs. This change happens at time point 2 and the number of hypotheses is changed to 3. As a result, F_u is updated to: $F_u[2] = (H - h[2])F_t^* + F_t^* = (5 - 3)2 + 2 = 6$. At time point 6, the size of the hypotheses list is changed to 4, therefore, the sampling interval is updated again accordingly to: $F_u[6] = (H - h[6])F_t^* + F_t^* = (5 - 4)2 + 2 = 4$. This value holds until time point 11 when another change in the size of the list occurs and so on, until the scenario is over.

3.3.3 Load heuristic

The second heuristic stems from the opposite intuition: Here the sampling interval is increased proportionally to the load of the process, in order to allow the agent to better manage its resources. As the number of hypotheses increases, the computational load increases and thus the sampling interval should be increased, in order to reduce the load. The relationship between the sampling interval and the entropy of the state (S) is:

$$F_l \propto S \tag{3.19}$$

Where S is the entropy of the state.
We suggest the following function for determine F_t during run time:

$$F_l[n] = 2^{S[n]} F_t^* = h[n] F_t^* \tag{3.20}$$

Where:
F_t^* is the minimal sampling interval of the IIRA
$S[n]$ is the current entropy
$h[n]$ is the current size of hypotheses list

When the size of the hypotheses list is maximal, the uncertainty of the agent's intention is maximal, hence, the load on the agent is maximal. As a result, the sampling interval is increased. Upon increasing the number of hypotheses by one, the sampling interval is increased by F_t^*.

We demonstrate this heuristic on the same system and scenario described above for the Uncertainty heuristic. The scenario (shown in Figure 3.19) starts with the maximal number of hypotheses, therefore, the sampling interval according to Equation 3.20 is set to: $F_l[0] = h[0]F_t^* = 5 \times 2 = 10$. The F_l stays constant until the next change in the size of the list occurs. This change happens at time point 2 and the number of hypotheses is changed to 3. As a result, F_l is updated to: $F_l[2] = h[2]F_t^* = 3 \times 2 = 6$. At time point 6, the size of the hypotheses list is changed to 4, therefore, the sampling interval is updated accordingly to: $F_l[6] = h[6]F_t^* = 4 \times 2 = 8$. This value holds until time point 11 when another change in the size of the list occurs and so on, until the scenario is over.

3.3.4 Combination heuristic

A combination of these heuristics is possible, where U is a function that incorporates the advantages of both heuristics described above.

$$F_c \propto U(F_l, F_u) \tag{3.21}$$

In this thesis, we examined the following function to determine F_c during run time:

$$F_c[n] = \max(F_u[n], F_l[n]) \tag{3.22}$$

We demonstrate this heuristic on the example described above. The scenario (shown in Figure 3.19) starts with the maximal size of hypotheses list, therefore, the sampling interval according to Equation 3.22 is set to: $F_c[0] = \max(F_u[0], F_l[0]) = \max(2, 10) = 10$. F_c stays constant until a change in the size of the list occurs. This change happens at time point 2 and the number of hypotheses is changed to 3. As a result, F_c is updated to: $F_u[2] = \max(6, 6) = 6$. At time point 6, the size of the hypotheses list is changed to 4, therefore, the sampling interval is updated to: $F_c[6] = \max(4, 8) = 8$. This value holds until time point 11 when another change in the size of the list occurs and so on, until the scenario is over.

3.4 Multi-agent recognition

IIRA is often required to recognize not just a single observed agent, but a number of agents. In general, more than one hypothesized explanation exists for observations and observed agent may trigger multiple recognition hypotheses. Thus, the computational load of a recognition process is exacerbated in the presence of multiple observed agents, and therefore, it becomes more crucial to reduce the computational load of the recognition process when dealing with an integrated multi recognition system. Such a system be called hereafter IIMRA.

In this section we expand our approach of *Temporal Monitoring Selectivity* for the recognition of the single agent case (described in Sections 3.1 – 3.3) to the recognition of the multiple agents case.

In Subsection 3.4.1 we discuss our approach of sampling, instead of triggering the process in every system cycle for the multi-agent recognition case. In Subsection 3.4.2 we discuss static sampling rate and our method to find the optimal static sampling rate for the multi-agent recognition case. In Subsection 3.4.3 we discuss dynamic sampling method and the suggested heuristics for dynamic sampling rates in order to reduce the computational load for the multi-agent recognition case.

3.4.1 Monitoring as sampling

In order to apply our *Temporal Monitoring Selectivity* approach on multiple observed agents, our IIMRA should maintain multiple recognition processes, a process for each observed agent. As a results, the system can decide whose process is triggered and when, in a separate and independent way.

A block diagram of an IIMRA which contains multiple recognition processes, one for each observed agent, is shown in Figure 3.22. Each of the recognition process is implemented in the same way as was presented for the single observed agent case in Section 3.1. In order to apply the sampling approach, a separate sampling interval is allocated to each of the processes of the i-th observed agent and denote by F_{ti}.

Algorithm 4 is utilized by the IIMRA to trigger each of the recognition processes. In order for the IIMRA to use a separate sampling interval, F_{ti} should be maintained for each process as well as *duration*$_i$ variable which holds the time from the last call for the i-th recognition process. The IIMRA calls the *Main* routine every system cycle. If the time exceeds F_{ti}, the system calls the recognition process again (*Main* routine, line 2–3 in Algorithm 4).

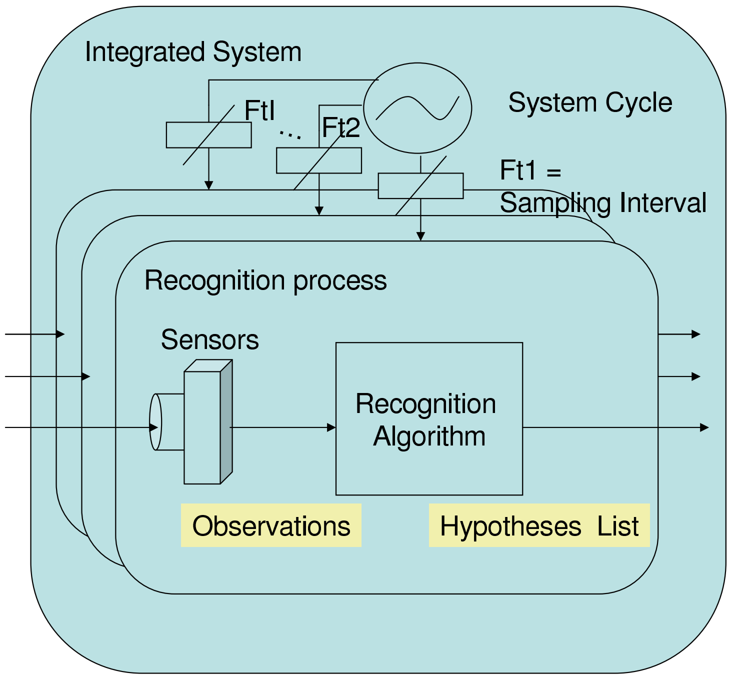


Figure 3.22: A block diagram of an IIMRA.

Algorithm 4 Triggering multiple recognition processes with sampling interval

– *Initialization routine*

```
1: for all $i$ do
2:     $m\_previousTrigger_i \leftarrow clock()$
3:     $F_{ti} \leftarrow SamplingInterval$
```

– *Main routine*

```
1: for all $i$ do
2:     $duration_i \leftarrow clock() - m\_previousTrigger_i$
3:     if $duration_i > F_{ti}$ then
4:         $m\_hypothesesList_i \leftarrow RecognitionProcess(i)$
5:         $m\_previousTrigger_i \leftarrow clock()$
```

– *RecognitionProcess(i) routine*

```
1: $m\_observations \leftarrow CollectsObservations(i)$
2: $hypothesesList \leftarrow Activates(Algorithm(m\_observations))$
3: return $hypothesesList$
```

Our objective is to reduce the computational load generated by the multiple recognition processes without downgrading the recognition results. In this part we present the formulations for *Recognition Load* and *Recognition Error*, to evaluate those two system parameters and to choose the system performance accordingly, for the multi-agent recognition case. The formulations for the multi-agent recognition case are mostly relying on the single recognition case which is described in Section 3.1.

The recognition process of the i-th agent provides the system with a list of its hypotheses. Let $h_i : \mathbb{N} \to \mathbb{N}$ be the *hypotheses number function* such that $h_i[n]$ is the number of hypotheses maintained by the IIMRA for the i-th observed agent at time n, where $n \in [0..N_i]$ denotes the time index and N_i is the total run time used up by the recognition algorithm for the i-th agent. Let $L_i = [n_1, n_2, ..., n_{J_i}]$ be the time lattice of the i-th agent, where $n_j \in \mathbb{N}$, $j \in \{1, ..., J_i\}$ are the time points when the recognition process monitors the i-th observed agent, clearly, $n_{J_i} \leq N_i \, \forall \, i$. Let $\overline{H_i}$ be the average number of hypotheses, which the recognition process maintains over the scenario run time for the i-th agent. The value of $\overline{H_i}$ is calculated by Equation 3.1 while using h_i, L_i, and N_i instead of h, L, and N as in the single case.

Since the executer agent performs I recognition processes each consumes its computational load, the total computational load consumed from the executer resources for recognition all the observed agents is the accumulated load over all the I processes.

Definition 3.4.1. *The* Recognition Load *of the IIMRA is therefore:*

$$\text{Recognition Load} = \sum_{i=1}^{I} \overline{H_i} \tag{3.23}$$

Where:

I is the number of the observed agents.

Let $\overline{E_i}$ denotes the *Recognition Error* of the i-th observed agent calculated by Equation 3.4, while using h_i^*, h_i and N_i instead of h^*, h and N as in the single case.

A common way to evaluate a process error is by measuring its MSE (Mean Squared Error). We consider the multi-agent recognition case as an expanded process, therefore we examine what the influence of the number of observed agents on the error of that process is, by averaging the *Recognition Error* of all the I processes.

Definition 3.4.2. *The* Recognition Error *of the IIMRA is:*

$$\text{Recognition Error} = \frac{\sum_{i=1}^{I} \overline{E_i}}{I} \tag{3.24}$$

Where:

I *is the number of the observed agents.*

3.4.2 Fixed sampling interval

In this subsection we describe the analytic models of the two system parameters: The *Recognition Load* and the *Recognition Error* in order to find an optimal fixed sampling interval for the multi-agent recognition case. The models rely on the models which are presented in Section 3.2 for the single case.

By fixed sampling rate we imply using a constant rate of recognition, thus F_{ti} is constant $\forall\ i$. In this work we use the same constant value for all i, hence $F_{ti} = F_t \ \forall\ i$.

Estimation Recognition Load

Assumption 3.4.3. *All the observed agents impose the same* Recognition Load, *which is denoted by* r.

In case where the above assumption fails to hold, one can use r as the the maximal *Recognition Load* of the observed agents deriving the *worst case* estimation or can use r as the the mean *Estimated Recognition Load* of the observed agents deriving the *average case* estimation. In this work we consider that the assumption holds valid for the sake of simplicity.

In recognition algorithms, the executer try to point out the behavior that is most suitable to its observations from a behavior library. The behavior library includes all the knowledge the executer has about all the observed agents. Therefore, any observed agent is recognized against the same comprehensive behavior library and by the same algorithm. This algorithm does not depend on the identity of the agent and its intention.

Definition 3.4.4. *The estimated* Recognition Load *model for IIMRA is:*

$$\text{Estimated Recognition Load} = \widehat{RL} = \frac{I \times r}{F_t} \tag{3.25}$$

Where:

Estimated Recognition Load *is the computational load which is generated by all the recognition processes.*

F_t *stands for the sampling interval*

I *stands for the number of observed agents*

r *stands for the computational load generated by the recognition process for a single agent*

Based on assumption 3.4.3, and Equations 3.6 and 3.7 for the single case, each observed agent adds its relative computational recognition load to the multi-agent recognition process. We therefore multiply the single *Estimated Recognition Load* with the number of the observed agents I.

Estimation Recognition Error

We assume that each scenario of the i-th agent has the same characteristics of number of hypotheses changes, H, and the same total run time, N. Therefore each of the i-th agent has the same normalized sampling interval: $k_i = k = \frac{F_t H}{N}$.

Based on the this assumption, $\widehat{RE_i}$ the error model of the i-th agent can be calculated in the same way as for the single case. For $k \geq 1$ case, $\widehat{RE_i}$ is calculated by Equation 3.10 and for the $k < 1$ case, $\widehat{RE_i}$ is calculated by Equation 3.14.

In order to comply with the definition of the measured *Recognition Error* of the multi-agent recognition case as is defined in Definition 3.4.2, we define the estimated *Recognition Error* model to be:

Definition 3.4.5. *The estimated* Recognition Error *model of the IIMRA is:*

$$\text{Estimated Recognition Error} = \widehat{RE} = \frac{\sum_{i=1}^{I} \widehat{RE}_i}{I} \tag{3.26}$$

Where:

I is the number of the observed agents

3.4.3 Dynamic sampling

Relying on the dynamic sampling method for the single case (Section 3.3) we define in this section the dynamic sampling method for the multi-agent case. In this work we use the same heuristics, based on the entropy definition (Section 3.3), for activating each of the multiple recognition processes. To evaluate the use of these heuristics for the multi-agent case, their performances are empirically evaluated and compared to the performances of the fixed sampling intervals (Chapter 5.3.2).

A dynamic sampling method in the IIMRA, is one in which the sampling interval F_{ti} varies with time and each recognition process of an observed agent is triggered according to its own F_{ti}.

A block diagram of such an IIMRA which triggers the recognition processes according to each dynamic sampling interval is shown in Figure 3.23. As can be seen, each of the sampling interval is dynamically changed according to new knowledge derived from its related recognition process.

The algorithm utilized by the IIMRA to trigger each of the recognition process is presented in Algorithm 5. In order for the IIMRA to use a dynamic sampling interval, an *UpdateInterval* routine which returns a new value for the i-th sampling interval should be implemented as well as a *DynamicFlag* which enables the routine. The IIRA calls the *Main* routine every system cycle. The *UpdateInterval* routine is called if there is a new information from the i-th recognition process about the hypotheses list and the *DynamicFlag* is activated (line 5 in the *Main* routine presented in Algorithm 5).

Algorithm 5 Triggering multi recognition processes with dynamic interval

– Initialization routine

```
1: for all $i$ do
2:     $m\_previousTrigger_i \leftarrow clock()$
3:     $F_{ti} \leftarrow SamplingInterval$
4:     $m\_dynamicFlag \leftarrow DynamicFlag$
```

– Main routine

```
1: for all $i$ do
2:     $duration_i \leftarrow clock() - m\_previousTrigger_i$
3:     if $duration_i > F_{ti}$ then
4:         $m\_hypothesesList_i \leftarrow RecognitionProcess(i)$
5:         if $sizeChanged(m\_hypothesesList_i) \cap m\_dynamicFlag$ then
6:             $F_{ti} \leftarrow UpdatesInterval(m\_hypothesesList_i)$
7:         $m\_previousTrigger_i \leftarrow clock()$
```

– RecognitionProcess(i) routine

```
1: $m\_observations \leftarrow CollectsObservations(i)$
2: $hypothesesList \leftarrow Activates(Algorithm(m\_observations))$
3: return $hypothesesList$
```

– UpdateInterval routine

```
1: $newInterval \leftarrow U(hypothesesList)$
2: return $newInterval$
```

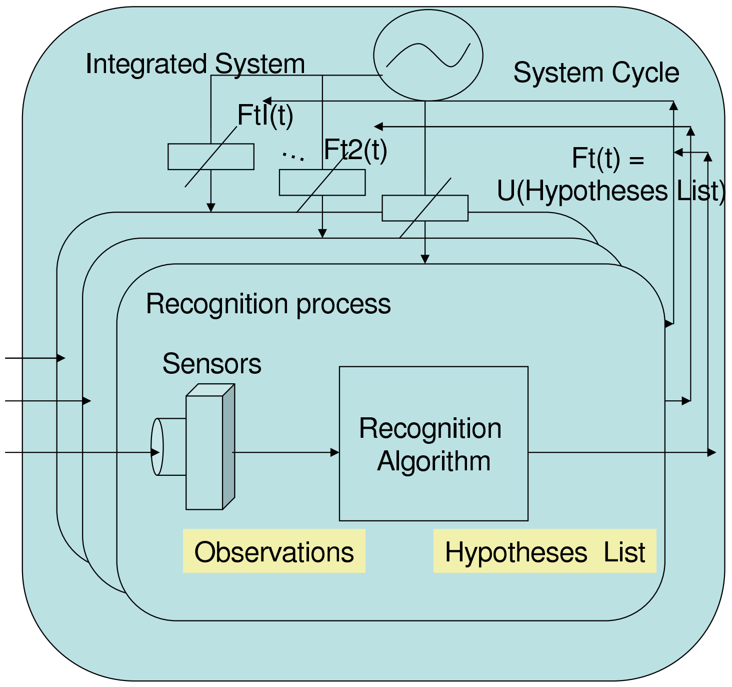


Figure 3.23: A block diagram of an IIMRA using dynamic sampling intervals.

Chapter 4

Mirroring

Based on recent researches, it is evident that intention recognition activity should be done continuously and simultaneously with the agent's model execution. This work aims at characterizing architecture for implementation of a continuous and simultaneous tracking capability.

Plan- and intention- recognition systems are built from two components: A recognition algorithm, and a plan-library (database) which it utilizes. While the previous chapter has tackled challenges associated with computation run-time, a key challenge in integration—the space requirements of the plan-library—remains unaddressed. In particular, the recognition system must tackle the following issues:

- Storing the plan library database in memory, in such a way that observations indicative of the others' knowledge/perception can be utilized.
- Adding knowledge that the executer agent acquires about possible plans, to the plan library. For instance, if the executer agents learns a new plan, it would also need to store knowledge of this plan such that it can be recognized when others are observed executing it. Or vice-versa, if the observed agent is found to be executing a novel plan, knowledge of it should ideally be acquired by the executer agent for its own goals (e.g., for imitation or programming by demonstration [18, 42, 20]).
- Comparison between the hypotehsized plans of the observed agent, and the

plan being executed by the executer agent. Such comparisons form the basis for observation-based coordination [26, 34].

This chapter addresses the integration challenges by advocating an approach, called *Mirroring*, which is independent of the *Temporal Monitoring Selectivity* approach, presented earlier. This is a specific approach to providing intention recognition at the architectural level, in which the executable procedural knowledge of the agent is re-used, as is, for recognition. This approach offers significant savings in computational resources compared to most existing approaches (where a separate recognition knowledge plan library is utilized, e.g., as in [26, 34, 39]). It also provides greater opportunity for social mechanisms that rely on comparisons between the executer agent and the observed agent (e.g., for identifying failures [34], or for social comparison [31]).

The *Mirroring* technique—inspired by mirror neurons in mammal brains [19, 51]—is based on three basic principles: (1) to reflect the knowledge of an executer agent to the observed agent's point of view; (2) to maintain the same database structure and apply the executer agent's mechanisms to the database of the observed agent; and (3) to unite the intention recognition process with the executer agent task together with the primary process.

We describe *Mirroring* in detail, and discuss the architectural requirements that are needed for it to work (Section 4.1). We evaluated its use in integrating plan-recognition capabilities into DIESEL [62, 36], an implemented teamwork and taskwork architecture, built on top of Soar [42]. DIESEL was chosen because its capabilities to support the three requirements that are needed to implement *Mirroring* (described in Section 4.1.2): an agent's *state*, recipe and task-maintenance behaviors. Hereafter, DIESEL which utilizes *Mirroring*, is called, M-DIESEL. In Section 4.2 we describe how easily the *Temporal Monitoring Selectivity* approach can be utilized in an IIMRA which applies recognition ability via *Mirroring*.

4.1 Intention recognition via *Mirroring*

In this section, an implementation of *Mirroring* is described. Specifically, we show how the same set of rules which was written for the executing agent is activated simultaneously on the executing agent plan library and re-uses to generate expected observations, which form the basis for recognition.

First the data structures and the mechanisms of the executed agent will be described (Section 4.1.1), follow by the steps, which are needed to be done, in order to get the same process on the observed agent database (4.1.2). As a result, this work shows a technique of maintaining multiple states at the same time; one state of the executed agent and one state for each of the observed agents. We describe in details a running example of the recognition process via *Mirroring* as it is utilized in M-DIESEL IIMRA for a scenario with two pilots agents. (Section 4.1.3).

4.1.1 What's in an agent?

This section describes the basic structures of an agent, and its primary components. We then describe how these components are utilized for *Mirroring*.

Agent's *state*

An agent's *state* is the data structure which maintains the agent's knowledge base. The *state* structure includes the inputs which the agent senses from its sensors, its variables which represent the results of processing these sensor readings, its plan graph, the memory of the agent, and the commands sent by the agent to its actuators.

Recipe mechanism

This work utilizes BDI or Behavior-based architectures, in which the executed agent's behavior repertoire is described by a directed acyclic connected graph [9], where vertices indicate behaviors, and edges can be of two types: vertical edges that decompose top behaviors into sub-behaviors, and sequential edges that specify the expected temporal order of execution. Each of the behaviors has preconditions, which must be true in order to be chosen to be performed, end-conditions, which must be true in order to terminate execution of the behavior. These conditions are tested against the agent's state. When a behavior is selected for execution, its application logic is applied to generate commands. The described graph is a data structure which represents the agent plan, and called recipe. The recipe mechanism maintains the agent's transitions in the graph.

When one behavior is terminated (its end-conditions are satisfied), the consequent behavior in the graph with true preconditions is proposed. When a compounded behavior is selected, its subordinate behavior that has true preconditions is proposed for running simultaneously. A set of consequent activated behaviors, starting from the root of the recipe, is marked as active path. The last behavior of an active path is the current agent behavior. For full details, please look at [25, 29, 30]

Execution monitoring via maintenance conditions

The active path represents the agent's intended behavior, rather than the real behavior of the agent. An example is when the active behavior in the executer agent's recipe is now 'walking-along-route,' but the agent is not moving due to a mechanical failure.

In order to determine the real behavior of the agent, the system needs a monitor mechanism which is able to check its progress along the behavior's graph. Monitoring its own active path provides the agent with fault detection ability.

To address this monitoring, the agent utilizes monitoring conditions on the execution of commands [33]. These monitoring conditions provide assertions on the execution: They utilize the agent's own sensors to verify that its execution and behavior are correct. Since application of different behaviors result in different commands being sent (or different sequences of commands), the monitoring conditions for execution are associated with behaviors: Each behavior has a set of monitoring conditions that assert its own execution.

Figure 4.1 presents a block diagram of the executer agent architecture. The executer maintains input layer to get observations, and an output layer to send commands to its actuators. The executer supports its own state (model of the world, beliefs, etc.). The executer architecture applies two mechanism: The recipe mechanism which maintains the plan graph of the executer agent and the execution monitoring mechanism.

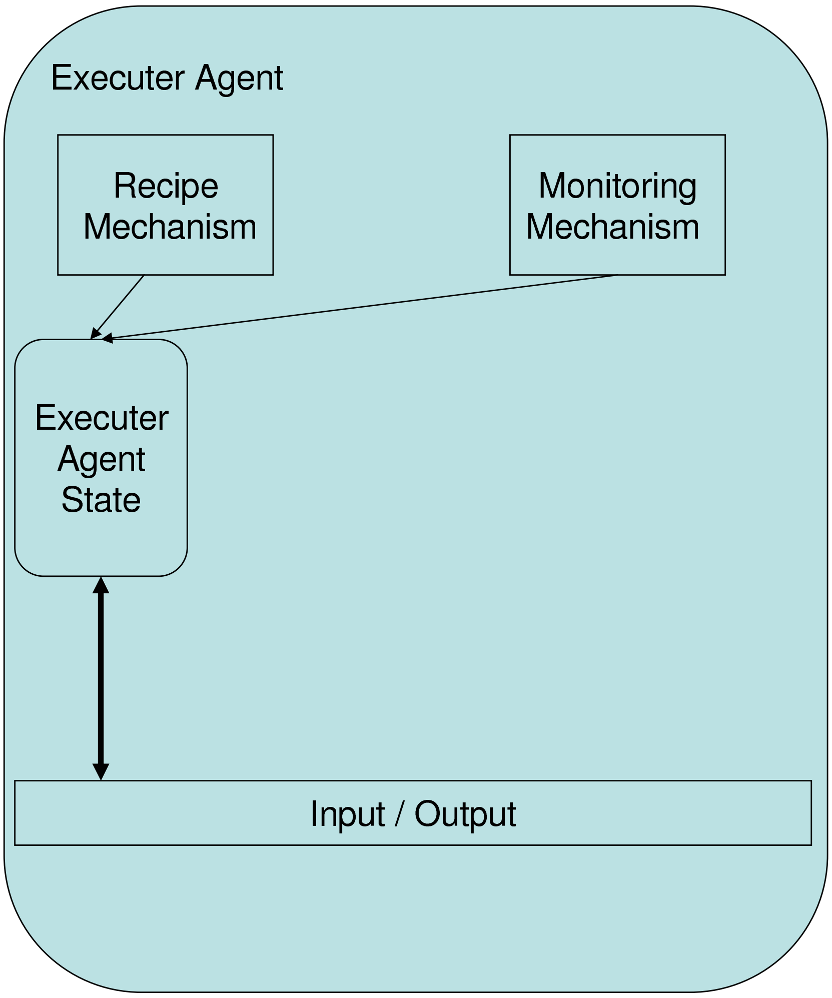


Figure 4.1: Executer agent's architecture.

4.1.2 *Mirroring* implementation

The previous subsection explains what is needed to implement an executer agent and a self-intention recognition for it. In this subsection, we explain how the same mechanisms are applied to an executer agent to achieve an IIMRA via *Mirroring*. First, we describe the steps, which are needed to be done, in order to apply the three executer's architectural components (as described in Subsection 4.1.1) also on the observed agent, and as a results achieve a mirroring process for it. In order to apply *Mirroring*, the major task is to define a set of instructions how to implement every mechanism of the executer agent, without restricting the rules for a specific *state*, but implementing them with referring to any *state*. As a result, once a new state is built the executer agent's mechanisms simultaneously run on it.

Observed Agent's *state*

The state of the observed agent represents the observed agent's hypothesized model of the world, beliefs, etc. It therefore reflects the executing agent's beliefs in the hypothesized state of the observed agent. The observed agent's *state* is the data structure which maintains the hypothesized observed agent's knowledge base. The *state* structure includes simulated inputs which the observed agent might sense from its sensors, its variables which represent the results of processing these simulated sensor readings, the memory of the observed agent, and the commands which might sent by the observed agent to its actuators. The decision when to instantiate an observed agent's state is according to an application requirement. In this work a new *state* data structure is instantiated once an observed agent first is identified. The only difference in the observed *state* is that its output layer on the *state* structure is not connected to the agent's output. This prevents the executed agent from behaving like the observed agent. In order to maintain the observed input layer a set of translating rules that interpret the executed agent inputs onto the observed agent inputs is required to be written. The translating process is performed when the observations are changed.

Recipe mechanism

Once a different observed *state* is built, the recipe mechanism runs on its behavior's graph. Due to uncertainty of the observed behavior, not only one behavior path is marked as active as in the executed *state*, but all the hypothesized behavior's paths which might be the current path are marked as active.

Monitoring mechanism

Once a different observed *state* is built, the monitoring mechanism runs on its behavior's graph. The process of reducing incorrect hypothesized paths along time is carried out utilizing the monitoring mechanism: An hypothesized path is declared as a failure hypothesis when its relative monitoring condition fails, indicating that this behavior is no longer running, as occurs when the executed agent monitors itself. All remaining plans, not marked as failure seem as hypotheses as to the correct state of the observed agent.

A blocks diagram of an agent's architecture which applies a recognition process via *Mirroring* is shown in Figure 4.2. The executer supports multiple states. One state for its own state (model of the world, beliefs, etc.), and one state for each of the observed agents. Each state is connected to the input layer to collect observations, the observed agent's state is connected through a set of translation rules in order to translate the executer observations onto the observed observations. Only the executer state is connected to the output layer, so that it be the only agent which commands its actuators. The recipe and the monitoring mechanisms run on every existing *state* data structure without distinction who it belong to.

The *Mirroring* algorithm is presented in Algorithm 6. The architecture calls the *initialization* routine to instantiate the state of the executer, agent 0 (line 1) and another state for every observed agent i-th (line 4), where the other agents are marked 1 and above. The instantiation of the i-th observed agent's state can be done according to an application requirement and not automatically when the application starts as it is demonstrated here. The *Main* routine executes its infinite loop with every system cycle. The executer's state is updated according to observations (line 3–4 in *Main* routine) and the observed agent's state is updated via translating the executer's observations (line 6 in *Main* routine).

In order to maintain the behavior's graph of the executer as well as of the observed agents the recipe mechanism is called with the relative state (line 7 in

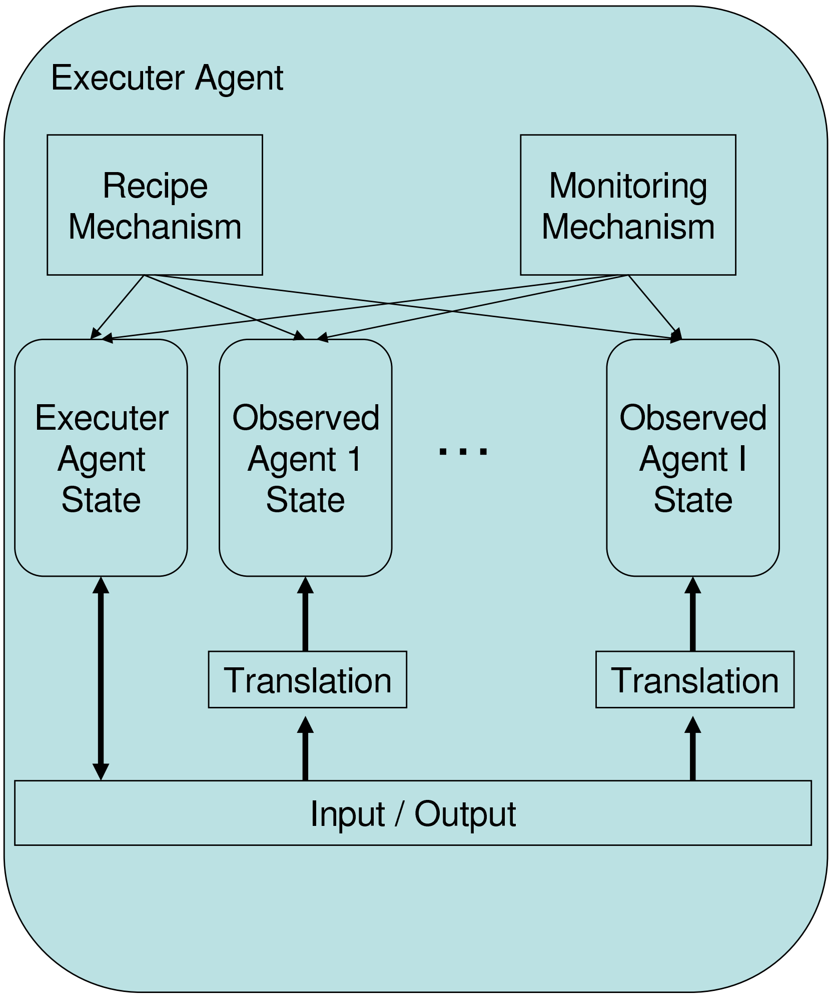


Figure 4.2: *Mirroring*: Multiple states in the executer agent's architecture.

Main routine). In order to verify the plan graph of the executer as well as of the observed agents the monitoring mechanism is called with the relative state (line 8 in the *Main* routine). Then the hypothesized behaviors list of the executer as well as of the observed agents is generated by calling the *CollectBehaviors* routine with the relative plan graph (line 9 in the *Main* routine). The *CollectBehaviors* routine passes the plan graph and collects all the behaviors which fulfil the condition: The behavior is the last behavior of an active path and does not have failed monitor conditions on it which indicates that this behavior can not really run (line 3 in the *CollectBehaviors* routine). If the hypotheses behavior list of the executer's state is empty we conclude that a fault is detected, since the executer has just one active path on its behavior's graph. The generating of the hypotheses list of an observed agent's state is the final step in the intention recognition process via *Mirroring*.

4.1.3 *Mirroring* running example

We demonstrate now the recognition process via *Mirroring*. We use an example scenario with two pilot agents (Figure 4.3). Five navigation points $1 - 5$ are selected, marked as X on the map. The executer agent, represented by the aircraft symbol E, runs a recognition process, for the observed agent's intentions, which is represented by aircraft symbol O. E executes its patrol plan and flies to navigation point 5 (Figure 4.3(a)) while O flies to navigation point 1. E wants to know if O intends to fly to navigation point 2 or to navigation point 3 or to fly to navigation point 4. We demonstrate the recognition process along the three steps scenario: O flies to navigation point 1 (Figure 4.3(a)), arrives at navigation point 1 (Figure 4.3(b)) and turns right to navigation point 3 (Figure 4.3(c)), all while E's patrol plan is on going.

E generates its own plan using the behavior graph shown in Figure 4.4(a). The solid lines indicate the actual behavior path executed by E and the dashed lines indicate unselected behaviors. Here, E is executing its plan: 'Execute-Mission' – 'Patrol' – 'Flight-on-Route' – 'Flight-to-Nav', and it flies toward navigation point 5. A monitoring condition, called 'Head-On', was defined to 'Flight-to-Nav' behavior, in order to verify the real execution of the behavior. As long as the head of the aircraft points to the navigation point, no 'failed_monitor' flag is set

Algorithm 6 Intention recognition via *Mirroring*

```
– Initialization() routine
1: $State_0 \leftarrow initiate(executer)$
2: $m\_observedAgents \leftarrow observations$
3: for all $m\_observedAgents_i$ do $\{0 \leq i\}$
4:     $State_{i+1} \leftarrow initiate(m\_observedAgents_i)$
– Main() routine
1: while $true$ do
2:     for all $State_i$ do $\{0 \leq i\}$
3:         if $State_i.executer = yes$ then
4:             $State_i.input \leftarrow observations$
5:         else
6:             $State_i.input \leftarrow Translation(observations)$
7:         $State_i.recipe \leftarrow PerformRecipeMechanism(State_i)$
8:         $State_i.recipe \leftarrow CheckMonitoringConditipons(State_i)$
9:         $State_i.m\_hypothesesList \leftarrow CollectBehaviors(State_i.recipe)$
10:        if $State_i.executer = yes \bigwedge State_i.m\_hypothesesList = \emptyset$ then
11:            $ExecuterFaultDetection()$
– CollectBehaviors(recipe) routine
1: $hypothesesList \leftarrow \emptyset$
2: for all $recipe.behavior$ do
3:     if $recipe.behavior$ is the last behavior of an active path $\bigwedge$
       $recipe.behavior.failed\_monitor = \emptyset$ then
4:         $hypothesesList \leftarrow \{P\} \bigcup hypothesesList$
5: return $hypothesesList$
```

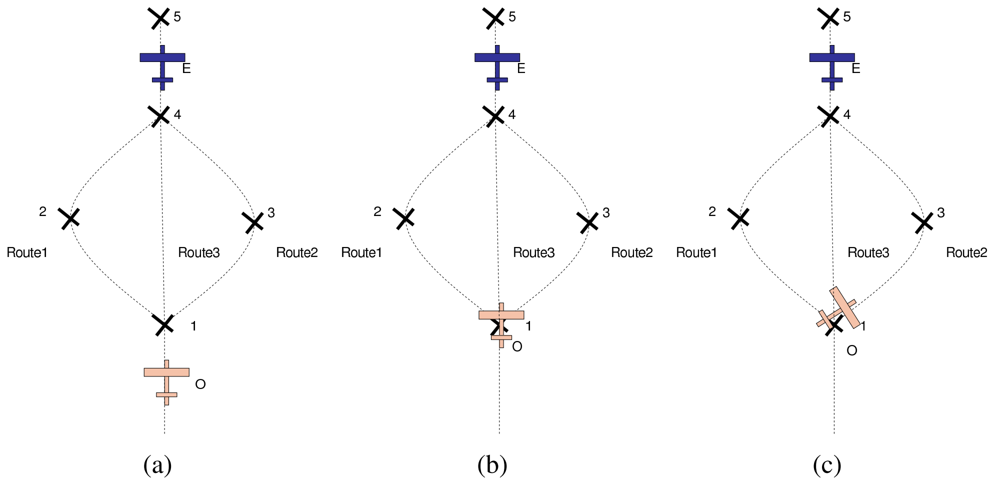


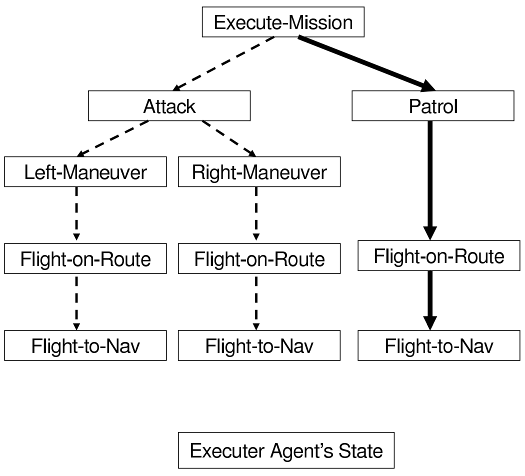
Figure 4.3: A simulated scenario.

on the related 'Flight-to-Nav' behavior, and the behavior is considered running as expected.

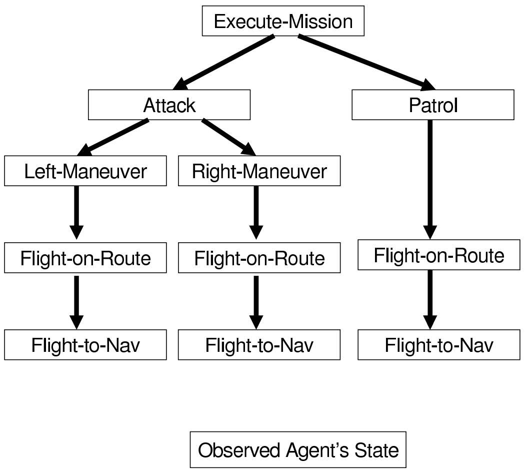
The executer agent's state contains all the relevant data structure common to all the behaviors. It maintains all of the dynamic sensor inputs regarding itself such as its heading and altitude, as well as dynamic radar input regarding O, such as heading, range, altitude, position. It also maintains dynamic sensor inputs regarding special points in its map.

E's state, for the situation in Figure 4.3(a), is shown in Figure 4.5. E flies to navigation point 5, therefore its active behavior in its recipe is the 'Flight-to-Nav' behavior ('State.Recipe.Behaviors.Flight-to-Nav.Active' = Yes) with 'Pointer to Nav' = 5 and 'Pointer to Route' = 3 which is the 'Patrol' route. Since its head points to navigation point 5, no 'failed_monitor' is set on this active behavior, therefore, no fault is detected (line 6 in *Main* routine in Algorithm 6) and this active behavior is considered as the real running one. In addition, E detects O on its radar, therefore the 'Entity' data structure is created with the values of O such as: ID=5, $X = 0$ and $Y = -100$ ('State.Input.Objects.Entity.ID' = 5, 'State.Input.Objects.Entity.Position.X' = 0, 'State.Input.Objects.Entity.Position.Y' = -100).

Mirroring allows us reuse the state topology in recognition of other agents. Although this is not a mandatory assumption [60], in our example we assume that



(a) Executer agent: Behavior graph.



(b) Observed agent: Behavior graph.

Figure 4.4: Behavior graph: Solid lines indicate active behaviors; dashed indicate unselected behaviors.

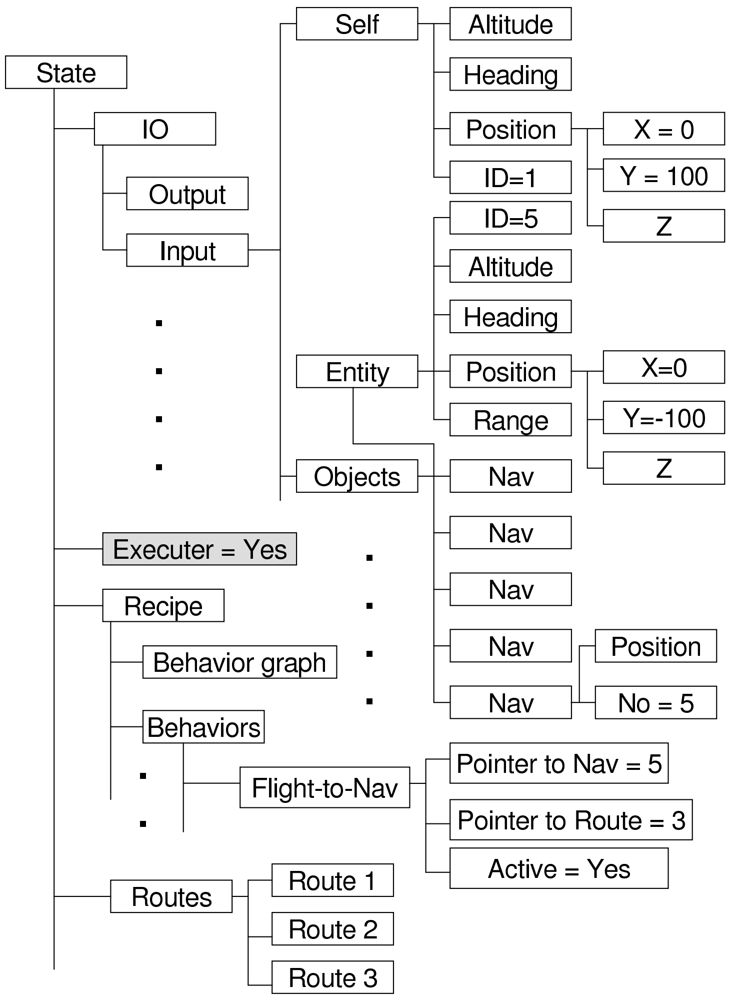


Figure 4.5: The executer's state when executing the scenario in Figure 4.3(a).

E and O posses an identical behavior library. Thus, E uses a graph such as the one in Figure 4.4(b) to recognize O's intention. Here, the graph (the solid lines in Figure 4.4(b)) represents E's state of O's current hypotheses behaviors for the situation of Figure 4.3(a). Such O's state is described in Figure 4.6.

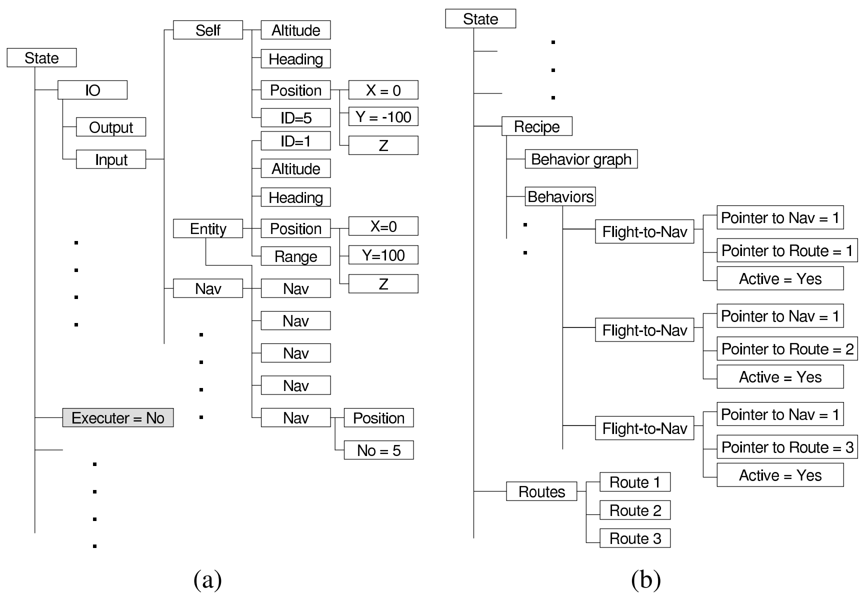


Figure 4.6: The observed's state when executing the scenario in Figure 4.3(a). Note, (b) is a continuation of (a).

The 'input' data structure in O's state is reflected by a set of translation rules, from the E's state 'input' data structure (line 1 in *Main* routine in Algorithm 6). The 'Self' data structure in Figure 4.6(a) reflects to O, consequently the ID value is 5 and Y position value is -100. The 'Entity' data structure reflects to E, consequently the ID value is 1 and the Y position value is 100 ('State.Input.Objects.Entity.ID' = 1).

As long as O flies toward navigation point 1 (Figure 4.3(a)) three plans are hypothesized:

1. 'Execute-Mission' – 'Attack' – 'Left-Maneuver' – 'Flight-on-Route' – 'Flight-to-Nav'.

2. 'Execute-Mission' – 'Attack' – 'Right-Maneuver' – 'Flight-on-Route' – 'Flight-to-Nav'.

3. 'Execute-Mission' – 'Patrol' – 'Flight-on-Route' – 'Flight-to-Nav'.

Thus, the three pathes in the O's behaviors graph (Figure 4.4(b)) are active and possible. This hypothesized plans are represented in O's state (Figure 4.6(b)) by the three active behaviors 'Flight-to-Nav'. Each of the active behaviors are aimed at navigation point 1, but this point belongs to three different routes. These three active behaviors with no 'failed_monitor' flag (since its head towards navigation point 1), constitute the hypotheses list of the observed agent (lines 3–4 in *CollectBehaviors* routine in Algorithm 6).

When O arrived at point 1 (Figure 4.3(b)), the 'Flight-to-Nav' behavior of point 1 ends for all of the three different 'Flight-on-Route' behaviors. Each of the 'Flight-on-Route' behaviors chose the next navigation point to fly to, according to their defined routes. Therefore, three new 'Flight-to-Nav' behaviors are created and activated on the O's state: One represents navigation point 2 which belongs to the 'Left-Maneuver' parent behavior. One represents navigation point 3 which belongs to the 'Right-Maneuver' parent behavior. And one represents navigation point 4 which belongs to the 'Patrol' parent behavior.

Such an updated O's state (for situation in Figure 4.3(b)) is shown in Figure 4.7. Since the head of O is towards navigation point 4, the monitoring conditions for navigation point 2 and navigation point 3 is no longer valid, and the 'failed_monitor' flag appears on the 'Flight-to-Nav' behaviors of the relative navigation points. Therefore the hypotheses list consists of one hypothesized path: 'Execute-Mission – Patrol – Flight-on-Route – Flight-to-Nav' (lines 3–4 in *CollectBehaviors* routine in Algorithm 6).

When O aircraft turns its head to navigation point 3 (Figure 4.3(c)), the O's state is updated as presented in Figure 4.8. The monitoring condition 'Head-On' is utilized to navigation point 3, therefore the 'failed_monitor' flag is removed from its relative 'Flight-to-Nav' behavior, thus the hypothesized path 'Execute-Mission' – 'Attack' – 'Right-Maneuver' – 'Flight-on-Route' – 'Flight-to-Nav' is added to the hypotheses list. Since O is no longer pointing to navigation point 4 the 'failed_monitor' flag is set to its relative 'Flight-to-Nav' behavior, and the hypothesized path 'Execute-Mission' – 'Patrol' – 'Flight-on-Route' – 'Flight-to-Nav' is removed from the hypotheses list (lines 3–4 in *CollectBehaviors* routine in Algorithm 6).

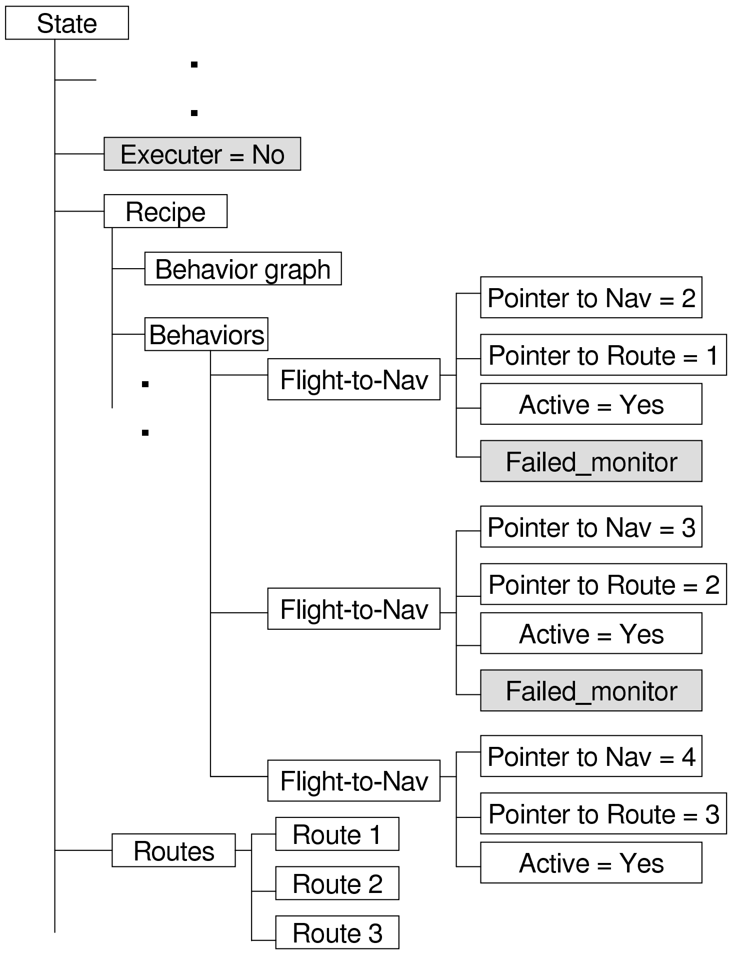


Figure 4.7: The observed's state when executing the scenario in Figure 4.3(b).

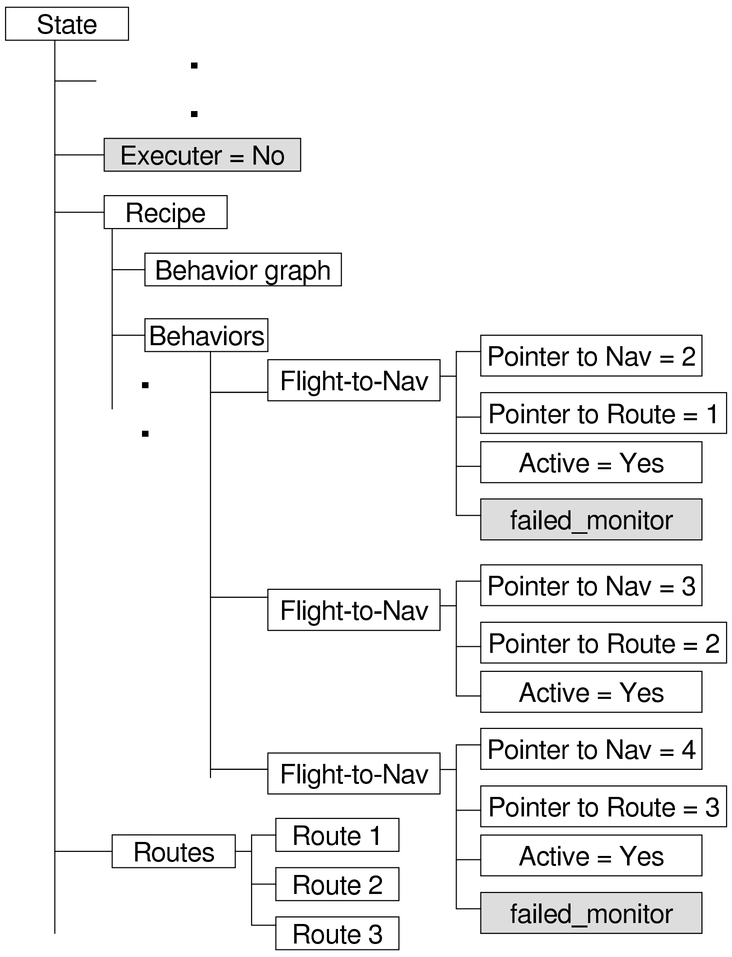


Figure 4.8: The observed's state when executing the scenario in Figure 4.3(c).

Such recognition is supported here via a uniform data structure and mechanisms for the execution of the agent's own behaviors and recognizing other agent's behaviors. In particular, executer's behavior and observed's behavior are selected and terminated in the same manner. Thus, as observed's state changes, which it does in reflecting the changing world situation, new observed's behavior may get selected in response.

In M-DIESEL all the mechanisms were developed in the executed agent's point of view, but with a reference to a virtual *state*. Once a *state* is built, the mechanisms are applied to this *state*. As a result, this work achieved performing of the executed agent simultaneously with intention recognition of multi observed-agents as a side-effect. Examples of the Soar code implementation in M-DIESEL, which applies the *Mirroring* approach to providing intention recognition at the architecture level can be found in Appendix A.

4.2 Applying *Temporal Monitoring Selectivity* in *Mirroring* architecture

The *Mirroring* architecture easily enables to apply the *Temporal Monitoring Selectivity* approach, in order to reduce the *Recognition Load*, as described in details in Section 3. The *Mirroring* already maintains a separated *state* for every observed agent, therefore, its recognition process can triggered separately from the executed process, by controlling its own F_t variable. The F_t can be a constant value or dynamically changed, thus the *Mirroring* architecture obtains all the advantages the *Temporal Monitoring Selectivity* approach achieves. In this section we describe the implementation of the *Temporal Monitoring Selectivity* in the M-DIESEL architecture. The evaluation of the *Temporal Monitoring Selectivity* approach in the M-DIESEL architecture, is described in details in Chapter 5.

In this research we use Soar's built-in mechanisms, for triggering the intention recognition process. Soar causes the rules to be executed depending on changes; When there are no changes in the conditions, the rules will not be executed. The input data structure in the observed agent's state is freezed for F_t seconds, as a result, the observed agent's state is not changed and therefore the mechanisms rules does not apply for it, thus no intention recognition is performed. A block diagram of such architecture which triggers the recognition process according to F_t variable is presented in Figure 4.9.

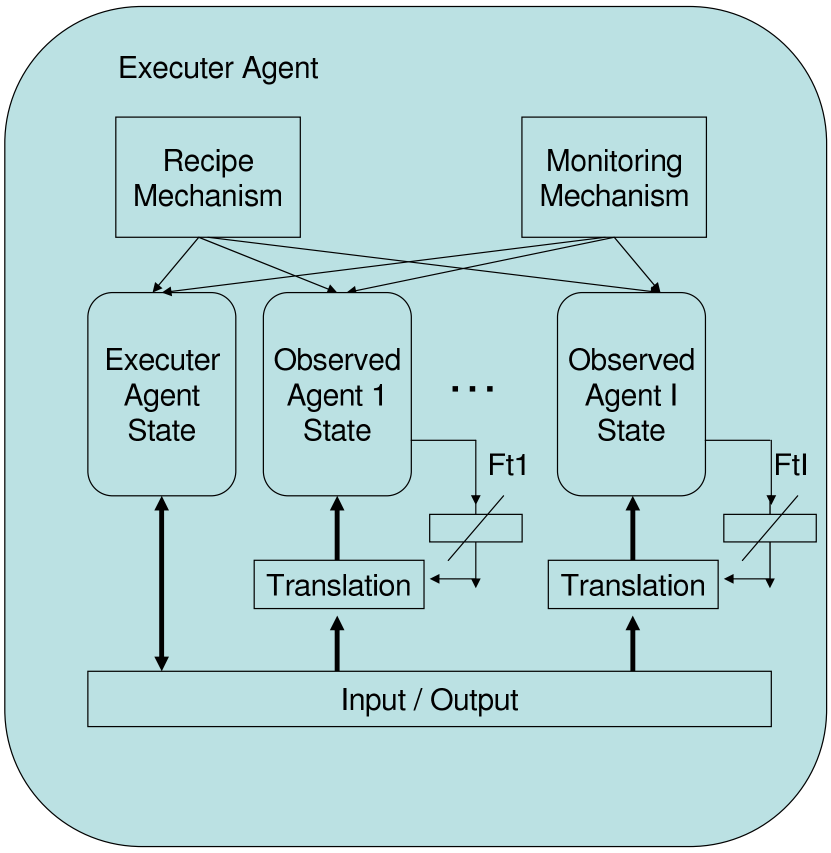


Figure 4.9: Applying *Temporal Monitoring Selectivity* in *Mirroring* architecture.

The relative triggering algorithm is presented by Algorithm 7. The observed agent's state is "frozen" for F_t seconds by preventing from the set of translation rules to be executed for F_t seconds. The *Timeout* routine is called to hold F_t (line 6 in the *Main* routine). When the next triggering time arrives, an *update_input* flag is set in the observed agent's state (line 2 in the *Timeout* routine). The translation rules which are depended on that flag are then executed and the observed agent's state is updated (line 8 in the *Main* routine). Changes in the observed agent's state, cause the mechanisms rules to be executed on it (lines 9–11 in the *Main* routine), therefore to the intention recognition process to run.

Algorithm 7 *Temporal Monitoring Selectivity* applying in *Mirroring*

– Initialization() routine

```
1: $State_0 \leftarrow initiate(executer)$
2: $m\_observedAgents \leftarrow observations$
3: for all $m\_observedAgents_i$ do $\{0 \leq i\}$
4:    $State_{i+1} \leftarrow initiate(m\_observedAgents_i)$
5:    $State_{i+1}.F_t \leftarrow SamplingInterval$
6:    $State_{i+1}.m\_nextTrigger \leftarrow clock() + State_{i+1}.F_t$
```

– Main() routine

```
1: while $true$ do
2:    for all $State_i$ do $\{0 \leq i\}$
3:       if $State_i.executer = yes$ then
4:          $State_i.input \leftarrow observations$
5:       else
6:          $Timeout(State_i)$
7:          if $State_i.update\_input = true$ then
8:             $State_i.input \leftarrow Translation(observations)$
9:       $State_i.recipe \leftarrow PerformRecipeMechanism(State_i)$
10:      $State_i.recipe \leftarrow CheckMonitoringConditipons(State_i)$
11:      $State_i.m\_hypothesesList \leftarrow CollectBehaviors(State_i.recipe)$
12:      if $State_i.executer = yes \bigwedge State_i.m\_hypothesesList = \emptyset$ then
13:         $ExecuterFaultDetection()$
```

– CollectBehaviors(recipe) routine

```
1: $hypothesesList \leftarrow \emptyset$
2: for all $recipe.behavior$ do
3:    if $recipe.behavior$ is the last behavior of an active path $\bigwedge$
      $recipe.behavior.failed\_monitor = \emptyset$ then
4:       $hypothesesList \leftarrow \{P\} \bigcup hypothesesList$
5: return $hypothesesList$
```

– Timeout(State) routine

```
1: if $State.m\_nextTrigger < clock()$ then
2:    $State.update\_input \leftarrow true$
3:    $State.m\_nextTrigger \leftarrow clock() + State.F_t$
4: else
5:    $State.update\_input \leftarrow false$
```

Chapter 5

Experiments

In this work two IIMRA were used to evaluate our proposed approaches to integrating of plan recognition: One system is called SBR and one is M-DIESEL. SBR is an IIMRA, built into a simulator for suspicious behavior recognition (Figure 5.1). It was built for research purposes at the MAVERICK lab in Bar-Ilan University. The recognition algorithm is described in the work of Avrahami-Zilberbrand et al. [9]. M-DIESEL is an IIMRA, which its recognition ability, applied by *Mirroring*, was implemented and integrated into the DIESEL architecture as part of this research. DIESEL is an agent architecture [62, 36] realized in Soar [42], and was implemented in the same lab at Bar-Ilan University for industrial usage. The M-DIESEL system was utilized in VR-Forces (Figure 5.2), a powerful and flexible simulation toolkit for generating and executing battlefield scenarios [61].

In Section 5.1 the experiment data sets are described. In Section 5.2 we evaluate our approach of *Time Monitoring Selectivity* for the recognition of a single observed agent. In Section 5.3 we evaluate our approach of *Time Monitoring Selectivity* for the recognition of multiple observed agents. Section 5.4 provides results of examining *Mirroring*.

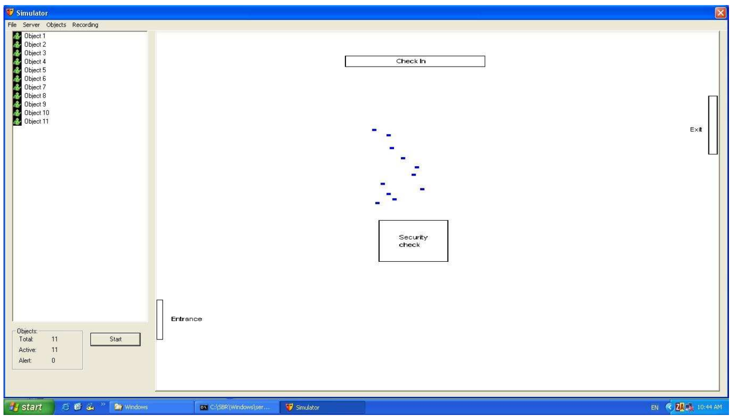


Figure 5.1: Suspicious behavior recognition simulator.

Figure 5.2: VR-Forces simulator.

5.1 Experiment data sets

In order to apply the sampling method, both systems the SBR and the M-DIESEL were updated to trigger each of the recognition processes according to a relative sampling interval, F_t. The SBR system has $F_t^* = 0.4$ seconds. The M-DIESEL system has $F_t^* = 2$ seconds.

We evaluate four scenarios: The linear, monotonic non-linear and non-monotonic scenarios (described in Sections 5.1.1 – 5.1.3) were execute 30 times on the suspicious behavior recognition simulator, and the non-monotonic scenario (described in Section 5.1.4) was execute 50 times on the VR-Forces simulator.

When an agent is situated in a simulator it executes a selected scenario (one of the four evaluated scenarios). When $I + 1$ agents are situated in a simulator each of the $I + 1$ runs the same evaluated scenario. One agent is the executer and the recognizer of the other I agents, which are the observed agents. The executer triggers the I recognition processes with the simulator minimal sampling interval. The *hypotheses number function* which were yielded by the I intention recognition processes are denoted to be the original signals. Then, the original signals were sampled with several fixed sampling interval and according to three heuristics which apply dynamic sampling interval. The original signals which yielded by the suspicious behavior simulator were sampled with F_t: 0.4, 0.8, 1.2, 1.6, 2.0, 2.4, 4.8, 3.2, 3.6, 4.0, 4.4, 4.8 seconds for the fixed sampling Interval. Those which yielded by the VR-Forces were sampled with F_t: 2, 4, 6, 8, 10, 12, 20, 30, 40, 80 seconds. The dynamic sampling was according to each of the three heuristic's equations: Equation 3.18 for the Uncertainty Heuristic, Equation 3.20 for the Load Heuristic, and Equation 3.22 for the Combination Heuristic.

All the original signals with their sampled signals which belong to one scenario were gathered to be a single experiment data set. Therefore we achieve four experiment data sets: 1. 'SBR linear scenario data set' 2. 'SBR monotonic non-linear scenario data set' 3. 'SBR non-monotonic scenario data set' 4.'M-DIESEL non-monotonic scenario data set'.

5.1.1 The linear scenario on the suspicious behavior recognition simulator

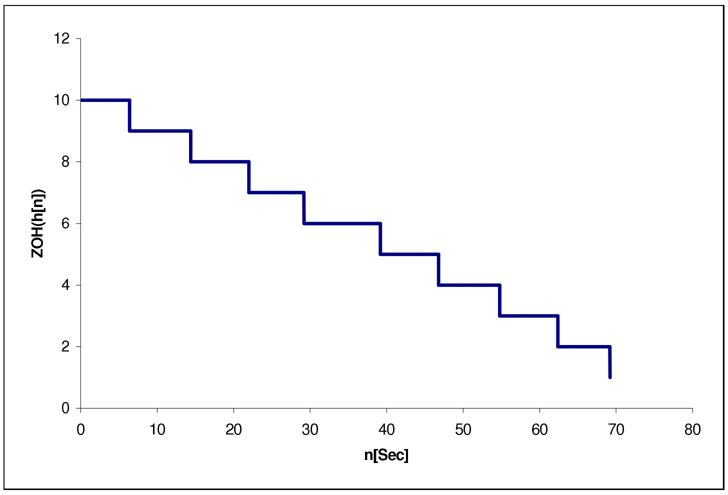


Figure 5.3: Linear scenario, SBR.

The scenario of a situated agent in the suspicious behavior recognition simulator is to walk along a straight corridor with 10 exit doors which are placed in a same distance, and exit through the last door. The intention recognition process of an observed agent maintains ten different hypotheses, one for every door the observed agent is able to exit. The intention recognition process goal is to determine the door towards which the observed agent walks. The hypotheses number starts with 10 hypotheses, then it is reduce along time when the observed agent passes nine doors till one truth hypothesis is committed, when the agent exits the last door. The ZOH of the *hypotheses number function*, which is yielded by such recognition process triggered with F_t^*, characterizes a linear signal (Figure 5.3), with $N = 70$ seconds and $H = 9$ changes.

5.1.2 The monotonic non-linear scenario on the suspicious behavior recognition simulator

The scenario of a situated agent in the suspicious behavior recognition simulator is the same scenario which described for the linear scenario (in $5.1.1$) but now

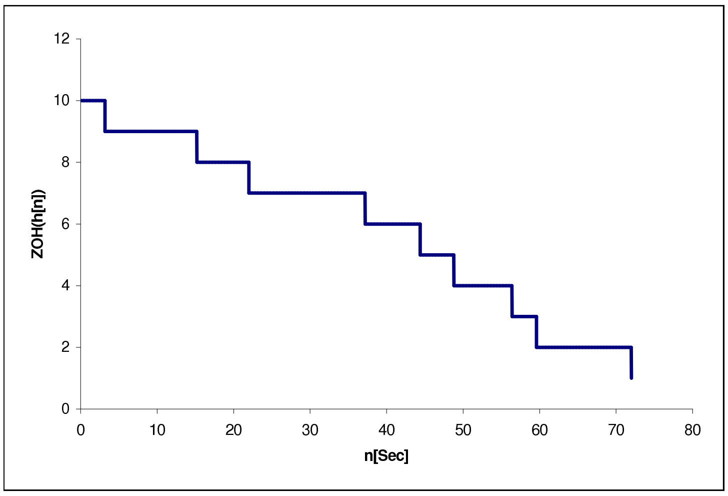


Figure 5.4: Monotonic scenario, SBR.

the ten doors are placed in varied distances. The ZOH of the *hypotheses number function*, which is yielded by the relative recognition process triggered with F_t^*, characterizes a monotonic non-linear signal (Figure 5.4), with $N = 73$ seconds and $H = 9$ changes.

5.1.3 The non-monotonic scenario on the suspicious behavior recognition simulator

The scenario of a situated agent in the suspicious behavior recognition simulator is the same scenario which described for the monotonic non-linear scenario with 5 doors (in 5.1.2) but now additional doors are hide behind corners which are placed along the corridor. The number of potential doors at each point is varied but not exceed value of 5. The ZOH of the *hypotheses number function*, which is yielded by the relative recognition process triggered with F_t^*, characterizes a non-monotonic signal (Figure 5.5), with $N = 76$ seconds and $H = 9$ changes.

5.1.4 The non-monotonic scenario on the VR-Forces simulator

The task of a situated agent in the VR-Forces simulator is to randomly chose one of five different waypoints and run towards it. The intention recognition process

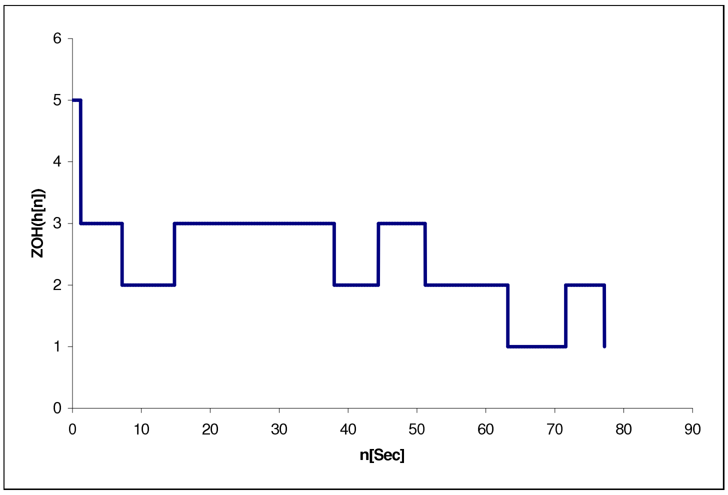


Figure 5.5: Non-monotonic scenario, SBR.

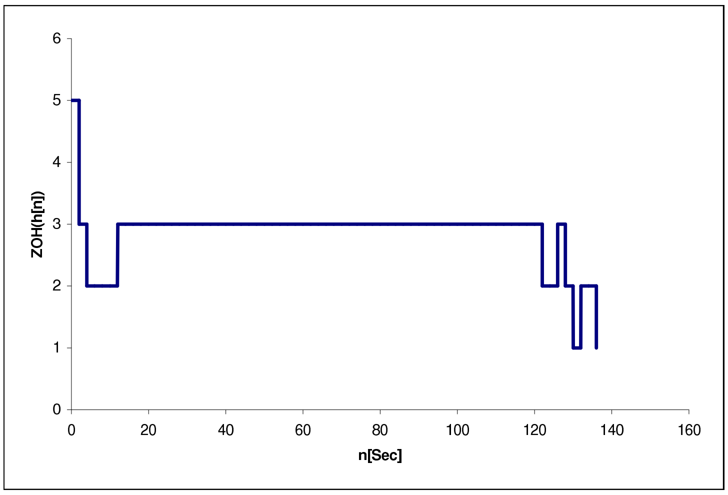


Figure 5.6: Non-monotonic scenario, M-DIESEL.

of the observed agent maintains five different hypotheses, one for every waypoint the observed agent is able to choose. The intention recognition process goal is to determine the waypoint towards which the observed agent runs. The hypotheses number is reduce along time till one correct hypothesis is committed. The ZOH of the *hypotheses number function*, which is yielded by such recognition process triggered with F_t^*, characterizes a non-monotonic signal (Figure 5.6), with $N = 136$ seconds and $H = 9$ changes.

5.2 Single observed agent

In this section we describe the evaluation of the SBR and the M-DIESEL systems for the single agent recognition case. Two agents were situated in suspicious behavior recognition simulator and in VR-Forces simulator. One agent is the executer and the recognizer of the other agent, which is the observed agent. Four experiment data sets for $I = 1$ were generated as described in Section 5.1 and evaluated.

In Subsection 5.2.1 we examine our suggested model for the *Estimated Recognition Load* against the four experiment data sets with fixed sampling interval. In Subsection 5.2.2 we examine our suggested linear model for the *Estimated Recognition Error* against the four experiment data sets with fixed sampling interval. In Subsection 5.2.3 we evaluate the performance of the four scenarios with dynamic sampling interval against those achieved with fixed sampling interval.

5.2.1 *Estimated Recognition Load* model for the fixed sampling interval

The objective of this experiment is to evaluate our *Estimated Recognition Load* model (described in Section 3.2.1). Our hypothesis is that the computational load, consumed by the recognition process in an IIRA, can be reduced as a function of a fixed sampling interval, F_t, according to Equation 3.7. In addition, we show with this experiment an empirical way to estimate the unknown system parameter, r (in Equation 3.7).

In this experiment we compared the *Estimated Recognition Load* values as a function of k, calculated by Equation 3.7 with those measured from the four experiment data sets, by Equation 3.2. Against each experiment data set we present several *Estimated Recognition Load* models related to several values of r and conclude which r matches each scenario and each simulator.

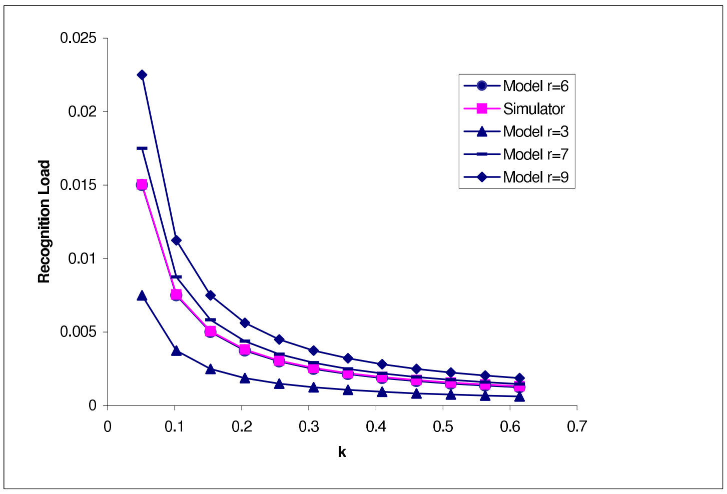


Figure 5.7: Linear scenario, SBR: *Recognition Load* as a function of k.

The non-square points curves, in Figures 5.7–5.10, presents the values of the *Estimated Recognition Load* as a function of k that were calculated by Equation 3.7 with different values of r. The empirical results for the *Recognition Load* as a function of k are presented by the square points curve.

In all four scenarios, the circle curve which presented the error model values, calculated with a specific r, coincides with the square points curve, which presents the empirical results for the *Recognition Load* as a function of k. According to the complete matching between the model values and the experiments results, the unknown system parameter, r, is empirically estimated: $r = 6$ for 'SBR linear scenario', $r = 6$ for 'SBR monotonic non-linear scenario', $r = 2.5$ for 'SBR non-monotonic scenario', and $r = 2.9$ for 'M-DIESEL non-monotonic scenario'.

According to the consist results, for different decreasing signals in different IIRA we conclude that the suggested model (Equation 3.7) is a good estimation for the *Recognition Load* as a function of F_t of a recognition process in an IIRA.

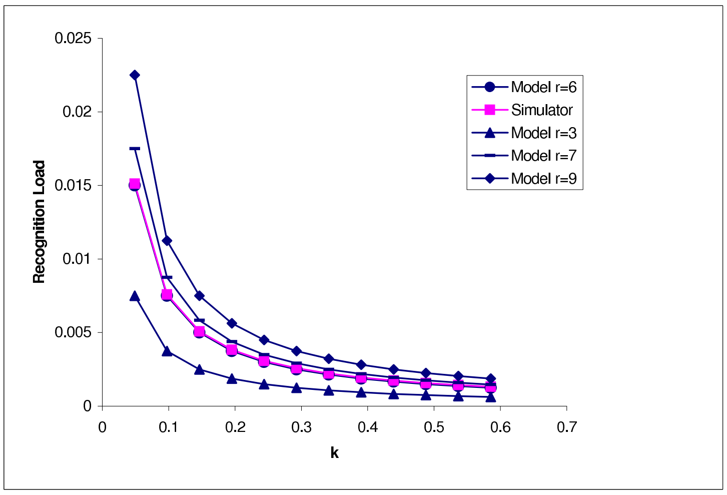


Figure 5.8: Monotonic scenario, SBR: *Recognition Load* as a function of k.

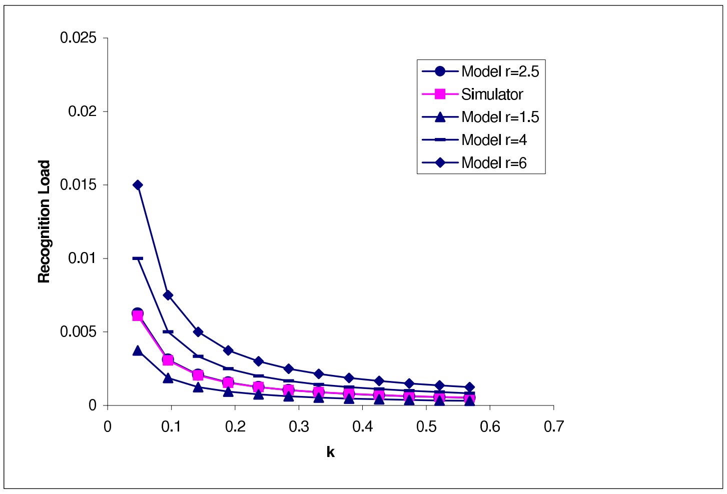


Figure 5.9: Non-monotonic scenario, SBR: *Recognition Load* as a function of k.

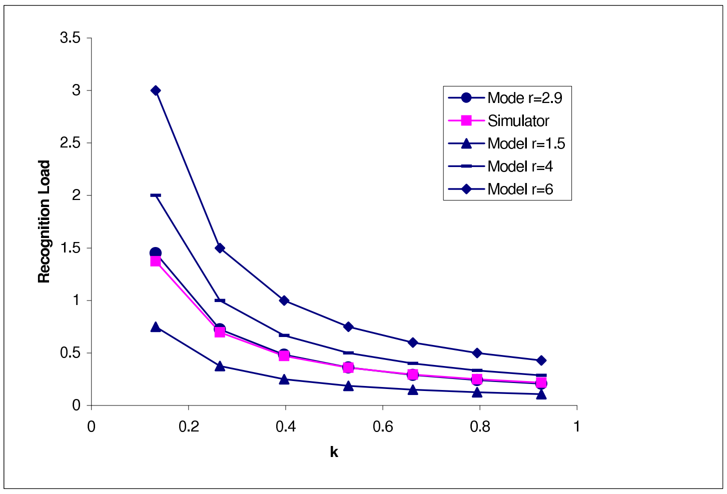


Figure 5.10: Non-monotonic scenario, M-DIESEL: *Recognition Load* as a function of k.

5.2.2 *Estimated Recognition Error* model for the fixed sampling interval

The objective of this experiment is to evaluate our *Estimated Recognition Error* model (described in Section 3.2.2). Our hypothesis is that the linear *Estimated Recognition Error* model as a function of k can be an approximation for the monotonic non-linear, and the non-monotonic decreasing signals (described in Section 3.2.3) of the recognition process in an integrated system.

We compared the *Estimated Recognition Error* values as a function of k calculated by Equation 3.14, with those measured from the four experiment data sets by Equation 3.4. The model error values are presented by the diamond points curve in Figures 5.11 – 5.14, and the results of the measured *Recognition Error* as a function of k, are depicted in these figures by the square points curve. The error bars mark one standard deviation below, and one above, the mean error results.

As it can be seen in Figures 5.11 – 5.14, around $k = 0.4$ a deviation between the model values and the experiments results has occurred, in all scenarios.

In the 'SBR linear scenario', 'SBR monotonic scenario' and 'M-DIESEL non-monotonic scenario', the model values are within one standard deviation of the experiment data sets results and outside one standard deviation of the data set

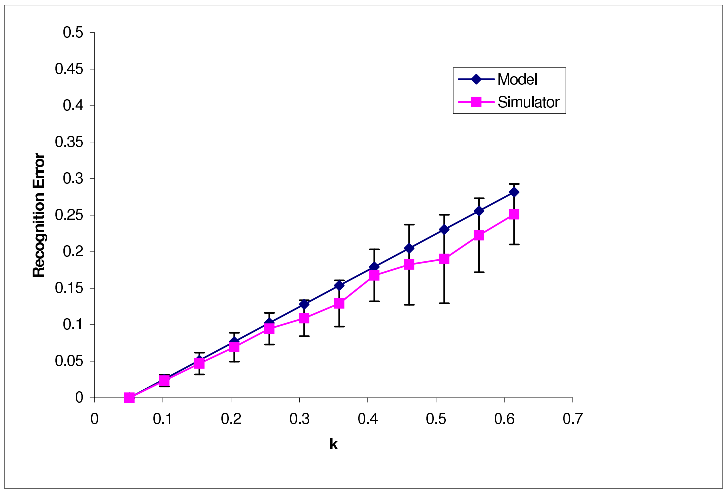


Figure 5.11: Linear scenario, SBR: *Recognition Error* as a function of k.

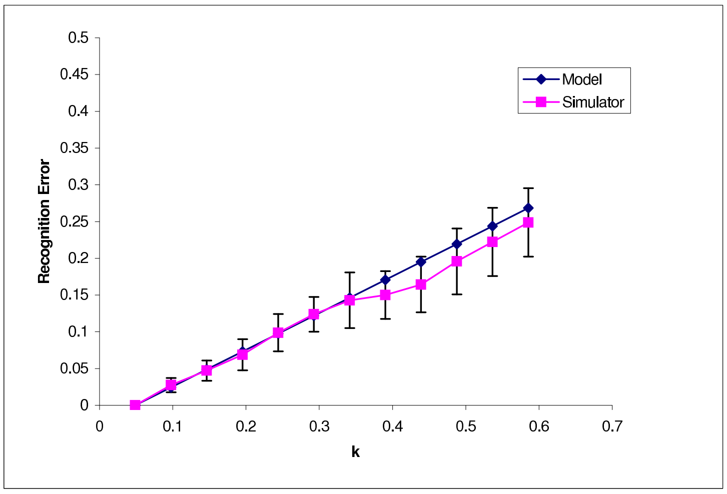


Figure 5.12: Monotonic scenario, SBR: *Recognition Error* as a function of k.

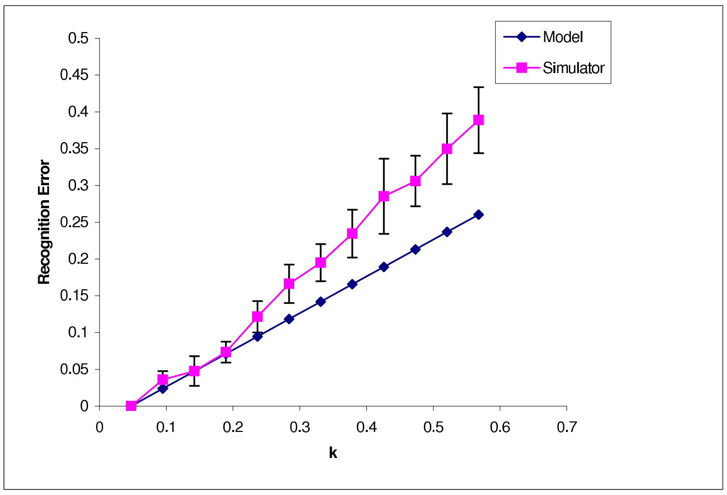


Figure 5.13: Non-monotonic, SBR: *Recognition Error* as a function of k.

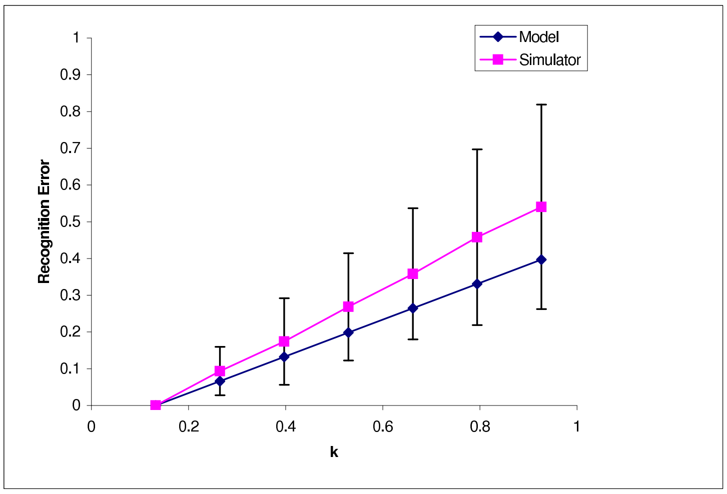


Figure 5.14: Non-monotonic scenario, M-DIESEL: *Recognition Error* as a function of k.

results in the 'SBR non-monotonic scenario'. Therefore, the linear error model (Equation 3.14) can be an approximation for estimation the *Recognition Error* as a function of k in some of the cases. However, the model results may still be useful as a lower bound for the *Recognition Error* of the recognition process in an IIRA.

5.2.3 Dynamic sampling compared with fixed sampling

The objective of this experiment is to evaluate the performance of the dynamic sampling method compared with those of the fixed sampling. Our hypothesis is that the dynamic sampling can reduce the *Recognition Load* while not downgrading the *Recognition Error*.

In evaluating the dynamic sampling performance, we face the technical obstacle that F_t varies with time along the scenario. This fact complicates the comparison with the fixed sampling performance, since their performance cannot be presented as a function of F_t as the fixed sampling performance can. In order to tackle this difficulty, we propose a new graph expressing the *Recognition Load* as a function of the *Recognition Error*. In such a diagram, every heuristic is represented by a single point ($RecognitionError_0$, $RecognitionLoad_0$) depicting the performance of the heuristic.

Definition 5.2.1. *A* family of heuristics *(FoH) is a group of heuristics that differ only in one parameter. A* family of heuristics *may be represented by a curve connecting points associated with different values of the parameter.*

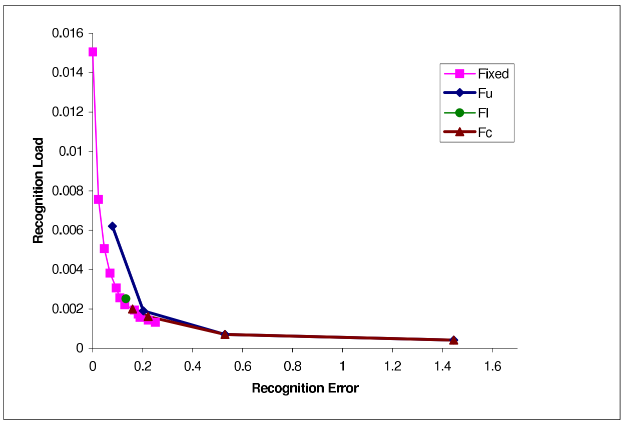
The Uncertainty and the Combination heuristics are expanded to a family of heuristics by varying the H parameter in Equation 3.18. We used the FoH convention to present the fixed sampling performance while the varied parameter is F_t and the dynamic sampling performance while the varied parameter is H.

Additional samples were performed on the original signals and expand the date sets. In the 'SBR linear scenario data set' and 'SBR monotonic non-linear scenario data set' a FoH of the Uncertainty heuristic and of the Combination heuristic were generated with $H =$ 10, 15, 30 and 50. In the 'SBR non-monotonic scenario data set' and 'M-DIESEL non-monotonic scenario data set' a FoH of the Uncertainty heuristic and of the Combination heuristic were generated with $H = 5$, 7, 10, 15, 30 and 50.

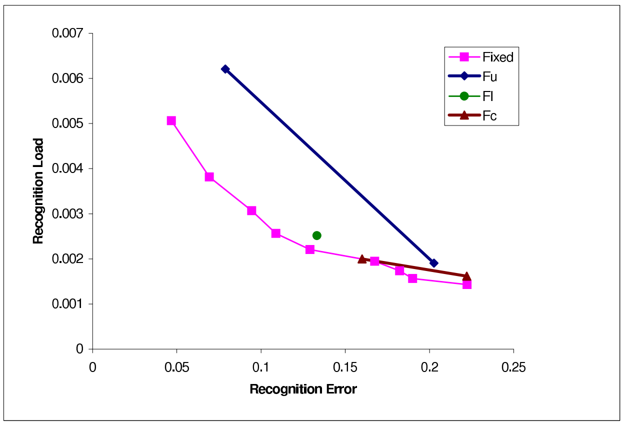
The measured results of the *Recognition Load* and of the *Recognition Error* with fixed sampling interval on the four experiment data sets were taken from Subsections 5.2.1 and 5.2.2. The results of the Fixed FoH are presented on a graph which depicts the *Recognition Load* as a function of *Recognition Error* and are plotted by the square curve (Figures 5.15 – 5.18), each square point presents the performance of a fixed F_t. The results of the *Recognition Load* and of the *Recognition Error* with dynamic sampling interval on the four experiment data sets were measured by Equation 3.2 and Equation 3.4 and are presented on the same graph of *Recognition Load* as a function of *Recognition Error* for comparison with the Fixed FoH results (Figures 5.15 – 5.18). The results of the Uncertainty FoH are presented by the diamond curve in these figures, each diamond point presents the performance of a fixed H. The performance of the Load Heuristic is presented by the circle point, and the performance of the Combination FoH are presented by the triangle curve, each triangle point presents the performance of a fixed H.

As can be seen in Figures 5.15(a), 5.16(a), 5.17(a), and 5.18(a), in all the four scenarios, the square curve, which represents the Fixed FoH, demonstrates the same tendency: as the sampling interval increases, the *Recognition Error* increases. *Recognition Load*, however, decreases asymptotically to a typical load value. We expect this typical load value to be identical to the load on the agent when executing its task without running the recognition process on the observed agent (as is presented in Equation 3.6). In addition, the performance of the Uncertainty and of the Combination FoH (presented by the diamond and triangle points curves) decrease asymptotically to the same load value and converge with the Fixed sampling FoH curve, due to the fact that there is a minimal load on the executer agent and no improvement can be achieved.

Figures 5.15(b), 5.16(b), 5.17(b), and 5.18(b), zoom in and focus on the most relevant performance range, in each of the four scenarios: The heuristics yield no improvement compared with the Fixed sampling FoH, in the 'SBR linear scenario experiment data set' (Figure 5.15(b)). In the 'SBR monotonic non-linear scenario experiment data set', the Combination FoH (F_c) performs better than the Uncertainty FOH (F_u) and the Load heuristic (F_l) compared with the Fixed FoH curve

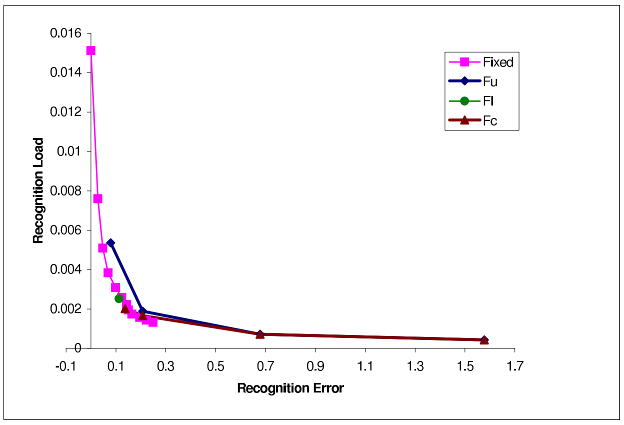


(a) *Recognition Load* as a function of *Recognition Error*.

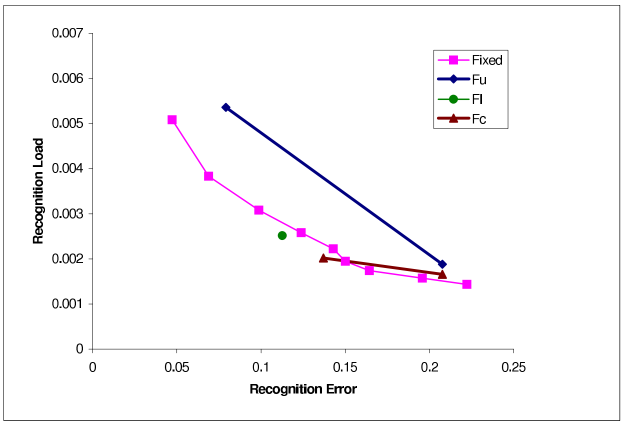


(b) *Recognition Load* as a function of *Recognition Error*, note: this figure is a zooming of 5.15(a).

Figure 5.15: Linear scenario, SBR performance: Dynamic sampling heuristics compared with the Fixed sampling FoH.

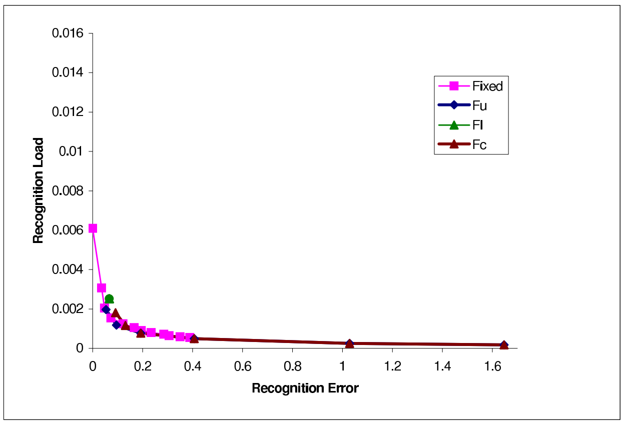


(a) *Recognition Load* as a function of *Recognition Error*.

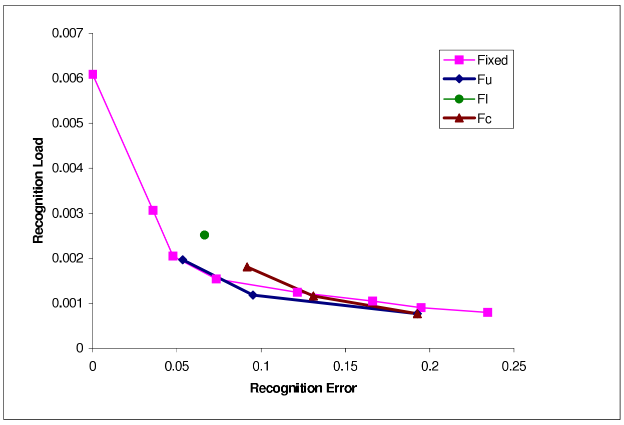


(b) *Recognition Load* as a function of *Recognition Error*, note: this figure is a zooming of 5.16(a).

Figure 5.16: Monotonic scenario, SBR performance: Dynamic sampling heuristics compared with the Fixed sampling FoH.

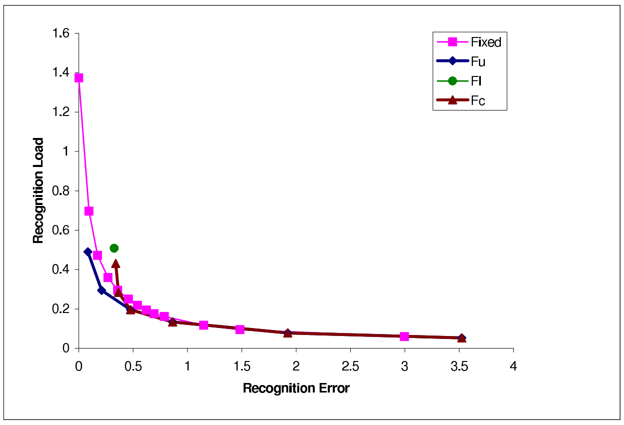


(a) *Recognition Load* as a function of *Recognition Error*.

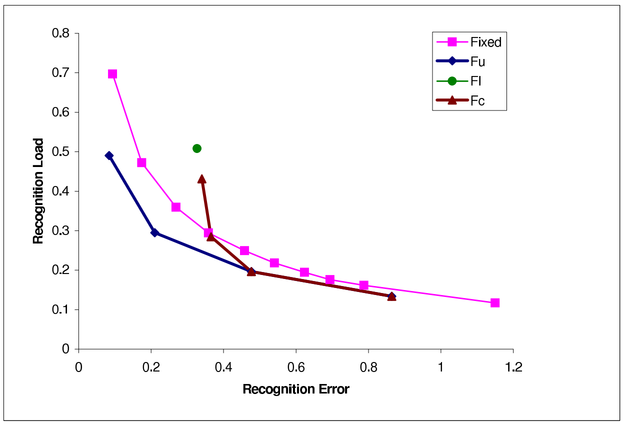


(b) *Recognition Load* as a function of *Recognition Error*, note: this figure is a zooming of 5.17(a).

Figure 5.17: Non-monotonic scenario, SBR performance: Dynamic sampling heuristics compared with the Fixed sampling FoH.



(a) *Recognition Load* as a function of *Recognition Error*.



(b) *Recognition Load* as a function of *Recognition Error*, note: this figure is a zooming of 5.18(a).

Figure 5.18: Non-monotonic scenario, M-DIESEL performance: Dynamic sampling heuristics compared with the Fixed sampling FoH.

	Uncertainty		Load	Combination	
	$H = 5$	$H = 7$		$H = 7$	$H = 10$
SBR Linear					
SBR Monotonic			$5.75E^{-31}$		$1.52E^{-39}$
SBR Non-Monotonic		$8.45E^{-32}$			
M-DIESEL Non-Monotonic	$1.24E^{-42}$	$2.5E^{-16}$		$3.5E^{-7}$	$2.4E^{-36}$

Table 5.1: Two-tailed paired t-Test results between the Fixed FoH and the dynamic heuristics for the single agent recognition case

(Figure 5.16(b)). In the non-monotonic scenario in both systems (Figures 5.17(b) and 5.18(b)) the Uncertainty FoH (F_u) performs better than the Load heuristic (F_l) and the Combination FoH (F_c) compared with the Fixed FoH curve.

A two-tailed t-Test contrasting the results for each heuristic (which shows an improvement), with the *Recognition Load* results for the same *Recognition Error* (those of the Fixed FoH curve), shows that the difference between them is statistically significant. Thus, we support the hypothesis that the dynamic sampling can reduce the *Recognition Load* while still providing the same recognition performance. The results of all two-tailed paired t-Test between the Fixed FoH and the dynamic heuristics are shown in Table 5.1.

5.3 Multiple observed agents

We evaluate our approach for the multi-agent recognition case in two IIMRA, the SBR and the M-DIESEL. Since we expect that the SBR yields the same results for the multi-agent recognition case as the results for the single agent recognition case for the linear scenario and for the monotonic non-linear scenario we did not execute experiments on these scenarios. Therefore, we focus on the non-monotonic decreasing signal scenario. We situated different number of observed agents: $I = 3$, 7, and 10, in the suspicious behavior recognition simulator, and $I = 3$, 5 and 8 in the VR-Forces simulator. Each simulator executed its non-monotonic scenario for 10 times for every I agents. Then we sampled the original signals with several constant values and three dynamic heuristics, and yielded two experiment data set: 'SBR non-monotonic multi scenario experiment data set'and

’M-DIESEL non-monotonic multi scenario experiment data set’.

In subsection 5.3.1 we examine our suggested models for the multi *Estimated Recognition Load* and for the multi *Estimated Recognition Error*. In Subsection 5.3.2 we evaluate the performance of the two IIMRA (SBR and M-DIESEL) with dynamic sampling against those achieved with fixed sampling for the multi-agent recognition case.

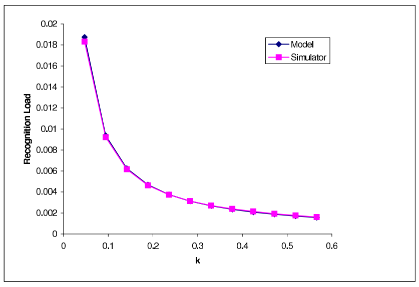
5.3.1 *Estimated Recognition Load* and *Estimated Recognition Error* models for the fixed sampling interval

The objective of this experiment is to evaluate our *Estimated Recognition Load* model for the multi-agent recognition case, as well as our *Estimated Recognition Error* model (described in Section 3.4.2). The first hypothesis is that the computational load, consumed by the recognition processes in an IIMRA, can be reduced as a function of a fixed sampling interval, F_t, according to Equation 3.25. In addition, we support with this experiment for Assumption 3.4.3 that all the observed agents impose the same computational load, r. The second hypothesis is that the *Recognition Error* for the multi-agent recognition case increases linearly as a function of fixed sampling interval, k, according to Equation 3.14.

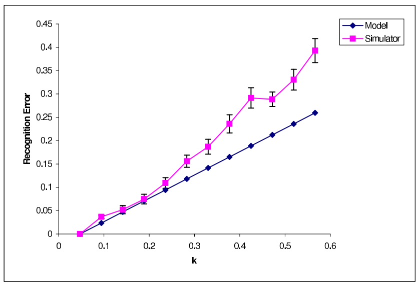
To examine our hypotheses we compared the *Estimated Recognition Load* values as a function of k, calculated by Equation 3.25 as well as the *Estimated Recognition Error* values as a function of k, calculated by Equation 3.26 with those measured by Equation 3.23 and Equation 3.24 from the two experiment data sets for different values of I.

The empirical results (presented by the square points curve) and the the estimated model values (presented by the diamond points curve) for the *Recognition Load*, as a function of k and I, are depicted in sub-figures (a), (c), and (e) in Figures 5.19 and 5.20. The empirical results and the estimated model values for the *Recognition Error* are presented in the same way in sub-figures (b), (d), and (f) in Figures 5.19 and 5.20.

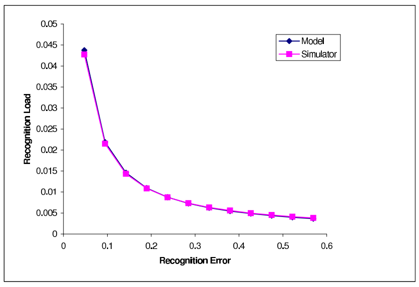
According to the results of the two evaluated IIMRA, we support the hypothesis that our suggested *Estimated Recognition Load* model as a function of k and I (Equation 3.23) is a good approximations for the non-monotonic decreasing sig-



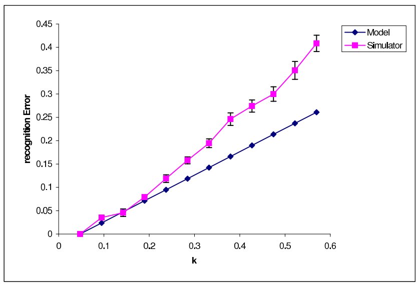
(a) *Recognition Load* as a function of k, $I = 3$, $r = 2.5$.



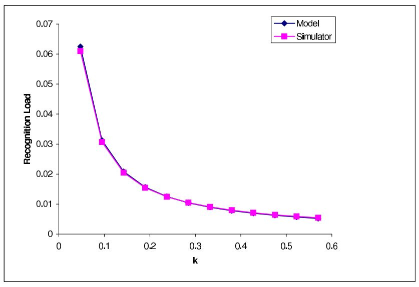
(b) *Recognition Error* as a function of k, $I = 3$.



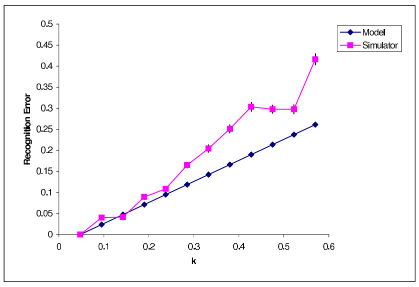
(c) *Recognition Load* as a function of k, $I = 7$, $r = 2.5$.



(d) *Recognition Error* as a function of k, $I = 7$.

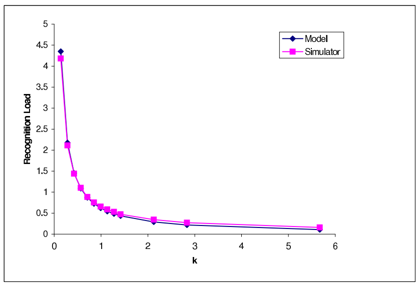


(e) *Recognition Load* as a function of k, $I = 10$, $r = 2.5$.

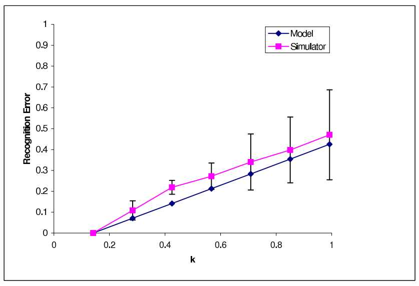


(f) *Recognition Error* as a function of k, $I = 10$.

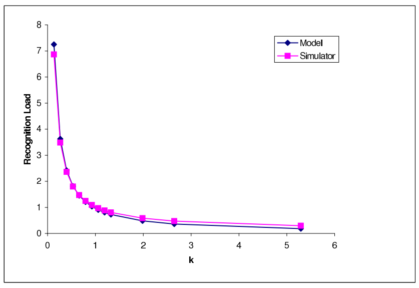
Figure 5.19: SBR: Estimated models compared with experiments. Note that the same r value is used with all I values.



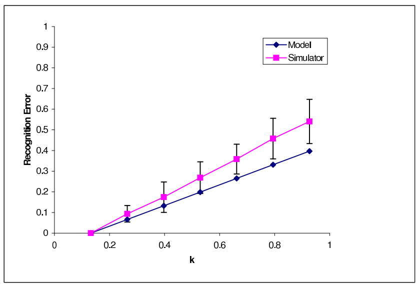
(a) *Recognition Load* as a function of k, $I = 3$, $r = 2.9$.



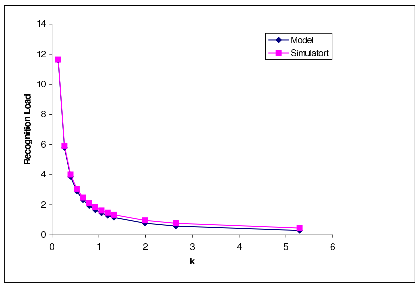
(b) *Recognition Error* as a function of k, $I = 3$.



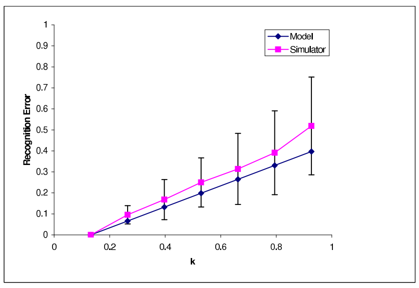
(c) *Recognition Load* as a function of k, $I = 5$, $r = 2.9$.



(d) *Recognition Error* as a function of k, $I = 5$.



(e) *Recognition Load* as a function of k, $I = 8$, $r = 2.9$.



(f) *Recognition Error* as a function of k, $I = 8$.

Figure 5.20: M-DIESEL: Estimated models compared with experiments. Note that the same r value is used with all I values.

nals for the multi-agent recognition case. In addition, according to the matching between the model results while using the same value of r and different values of I, we support our assumption (Assumption 3.4.3) that when the multiple observed agents execute the same scenario, the integrated system consumes the same amount of resources for each of the recognition process.

In the 'M-DIESEL non-monotonic multi scenario experiment data set', the estimated error model values are within one standard deviation of the empirical results and are outside one standard deviation in the 'SBR non-monotonic multi scenario experiment data set' . Like in the single agent recognition case the linear error model for the multi-agent recognition case (Equation 3.24)) can be an approximation for the *Recognition Error* as a function of k only in some of the cases. However, the model results may still be useful as a lower bound for the *Recognition Error* of the recognition processes in an IIMRA.

5.3.2 Dynamic sampling compared with fixed sampling

The objective of this experiment is to evaluate the performance of the dynamic sampling method compared with those of the fixed sampling for the multi-agent recognition case. Our hypothesis is that the dynamic sampling can reduce the *Recognition Load* while not downgrading the *Recognition Error*.

First we examine the performance of using the fixed sampling method for different number of observed agents, I. Then we take one case of I, for each system, and examine the performance of the dynamic sampling compared with those of the fixed sampling.

The results of the fixed sampling interval for the two experiment data sets, were taken from Section 5.3.1, and presented in a graph which depicts the *Recognition Load* as a function of *Recognition Error*. Each point of each curve in this graph presents the performance of a fixed F_t. Each curve in this graph present different I. Figure 5.21 presents the fixed sampling performance of the SBR IIMRA, and Figure 5.22 presents the fixed sampling performance of the M-DIESEL IIMRA.

Each curve related to different I at Figures 5.21 and 5.22 demonstrates the same tendency with no dependency on I: as the sampling interval increases, the

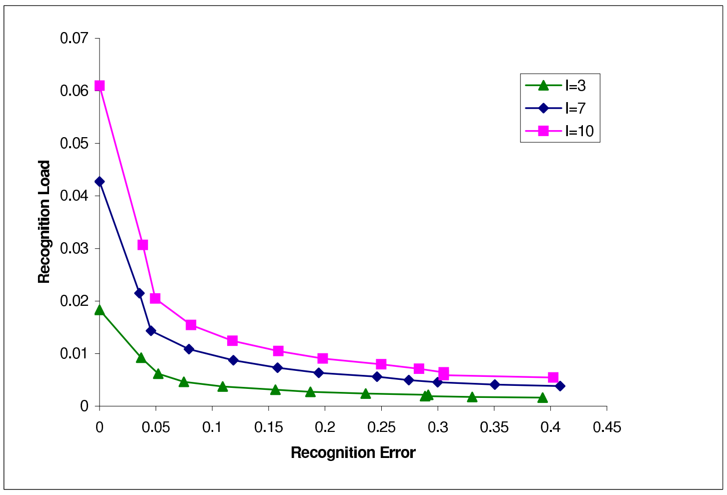


Figure 5.21: SBR: Fixed sampling performance.

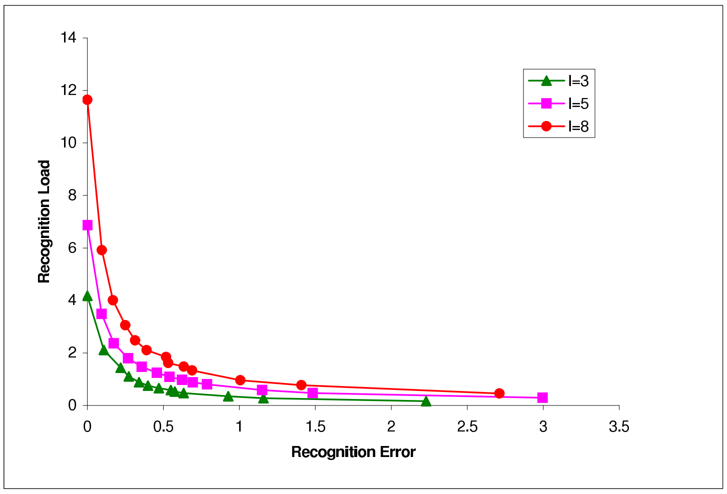


Figure 5.22: M-DIESEL: Fixed sampling performance.

Recognition Error increases. *Recognition Load*, however, decreases asymptotically to the same typical load value. We expect this typical load value to be identical to the load on the agent when executing its task without performing any of the recognition processes on the observed agents.

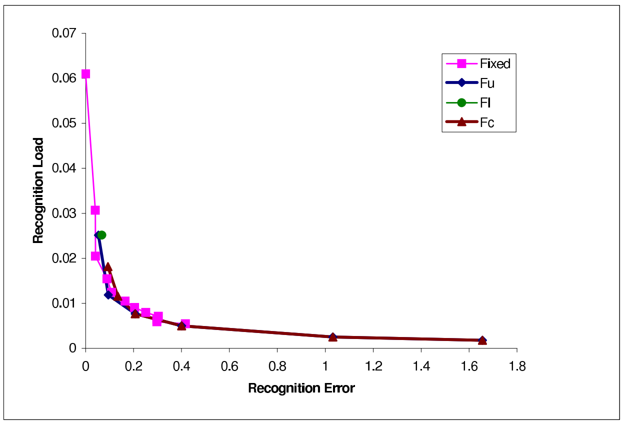
To evaluate the dynamic sampling performance for the multi-agent recognition case, we use the data set generated by the suspicious behavior recognition simulator with 10 observed agents, and the data set generated by the VR-Forces simulator with 5 observed agents.

The empirical results of the *Recognition Load* and of the *Recognition Error* with fixed sampling interval were taken from the experiment described in Subsection 5.3.1. The results of the Fixed FoH are presented on a graph which depicts the *Recognition Load* as a function of *Recognition Error* (Figures 5.23 and 5.24) and are plotted by the square curve. The empirical results of the *Recognition Load* and of the *Recognition Error* with dynamic sampling interval were measured by Equation 3.23 and Equation 3.24 and are presented on the same graphs of *Recognition Load* as a function of *Recognition Error* for comparison with the Fixed FoH results (Figures 5.23 and 5.24). The results of the Uncertainty FoH are presented by the diamond curve in these figures, each diamond point presents the performance of a fixed H. The performance of the Load Heuristic is presented by the circle point, and the performance of the Combination FoH are presented by the triangle curve, each triangle point presents the performance of a fixed H.

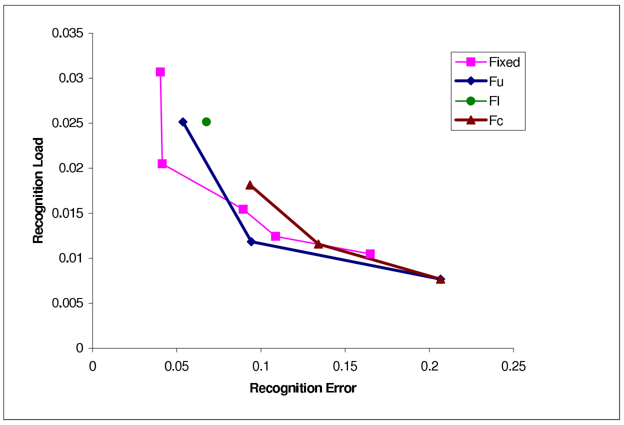
As can be seen in Figures 5.23(a) and 5.24(a) in both systems the performance of the Uncertainty and of the Combination FoH converge to the same performance of the Fixed sampling FoH, due to the fact that there is a minimal load on the executer agent and no improvement can be achieved.

Figures 5.23(b) and 5.24(b), zoom in and focus on the most relevant performance range. The Uncertainty FoH (F_u) performs better than the Load heuristic (F_l) and the Combination FoH (F_c) compared with the Fixed FoH curve in both IIMRA.

A two-tailed t-Test contrasting the results for F_u and F_c (which shows an improvement), with the *Recognition Load* results for the same *Recognition Error* (those of the Fixed FoH curve), shows that the difference between them is statistically significant. Thus, we support the hypothesis that the dynamic sampling can

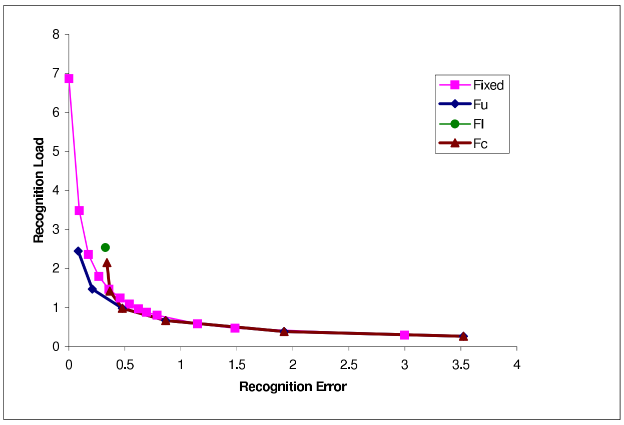


(a) *Recognition Load* as a function of *Recognition Error*.

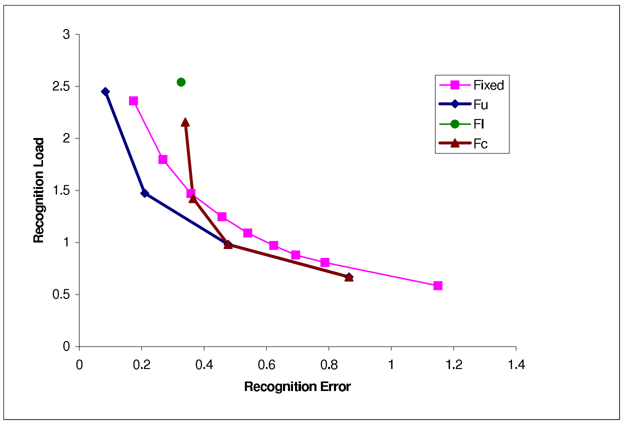


(b) *Recognition Load* as a function of *Recognition Error*, note: this figure is a zooming of 5.23(a).

Figure 5.23: SBR: Dynamic sampling heuristics compared with the Fixed sampling FoH, $I = 10$.



(a) *Recognition Load* as a function of *Recognition Error*.



(b) *Recognition Load* as a function of *Recognition Error*, note: this figure is a zooming of 5.24(a).

Figure 5.24: M-DIESEL: Dynamic sampling heuristics compared with the Fixed sampling FoH, $I = 5$.

	Uncertainty		Load	Combination	
	$H = 5$	$H = 7$		$H = 7$	$H = 10$
SBR Non-Monotonic $I = 10$		$8.12E^{-14}$			
M-DIESEL Non-Monotonic $I = 5$	$5.2E^{-12}$	$4.17E^{-13}$		$2.69E^{-6}$	$1.32E^{-11}$

Table 5.2: Two-tailed paired t-Test results between the Fixed FoH and the dynamic heuristics for the multi-agent recognition case

reduce the *Recognition Load* while still providing the same recognition performance also in the multi-agent recognition case. The results of two-tailed paired t-Test between the Fixed FoH and the dynamic heuristics are summarized in Table 5.2.

5.4 *Mirroring*

To evaluate the efforts which were needed to implement *Mirroring*, the number of rules is compared between a system with *Mirroring* and a system without. Let X be the number of mechanism rules, let Y be the number of application rules and Z be the number of rules which needed to be added in order to adjust the system to a *Mirroring* one. In a *Mirroring* system, the comprehensive number of rules is $X+Y+Z$, where k rules from Z are dedicated to translating the observations and their number is constant. In a system without *Mirroring*, the total number of rules is $2X + 2Y$, since one set of rules is devoted to the executer agent and another set of rules is devoted to the observed agent. Y grows with the system, while it can be easily noticed that the effort with *Mirroring* is constant.

We demonstrate the efforts required to adjust DIESEL to M-DIESEL: DIESEL system consists of $X = 164$ mechanism rules and $Y = 358$ application rules. In order to apply a recognition ability via *Mirroring* $k = 19$ translation rules and $l = 26$ additional mechanism-adjustment rules (such as rules for deciding on when to instantiate the observed agent's state) were required (where $Z = k + l$). We believe that the l adjustment mechanism rules are also required in a system without *Mirroring*. Thus, the k translation rules which are 5.3 percents of the Y

application rules. Since Y is expected to grow with the growing of the application and k is expected to stay constant, the savings are very significant. Compare this with systems with plan recognition and no mirroring, where any additional rules in Y, would necessitate adding appropriate recognition rules.

Chapter 6

Discussion and Conclusions

In this work the *Temporal Monitoring Selectivity* approach is introduced in order to reduce the computational load resulting from integrating the intention recognition ability into an agent system without downgrading the recognition quality. Then, *Mirroring* is introduced as an integration technique.

Two integrated intention recognition systems were used to evaluate our suggested approach: One system is called SBR which is run in the *suspicious behavior recognition* simulator and one is M-DIESEL which is executed in the VR-Forces. In order to apply the sampling method, both systems were updated to trigger each of the recognition processes according to a corresponding sampling interval, F_t. Four experimental data sets were presented and compared:

1. 'SBR linear scenario data set', the scenario is presented in Figure 5.3.

2. 'SBR monotonic non-linear scenario data set', the scenario is presented in Figure 5.4.

3. 'SBR non-monotonic scenario data set', the scenario is presented in Figure 5.5.

4. 'M-DIESEL non-monotonic scenario data set', the scenario is presented in Figure 5.6.

First the experiments results are discussed, followed by discussion the procedure that this research defines for evaluation the integrating a recognition ability into the system. Then, we discuss the opportunities that the *Temporal Monitoring Selectivity* approach raises. At last, we summarize the requirements from a recognition algorithm to be integrated with the suggested *Temporal Monitoring Selectivity* approach.

Recognition Load **Model**

Our main hypothesis is that the computational load, consumed by the recognition process in an integrated system, can be reduced to a function of a fixed sampling interval, F_t, according to Equation 3.7.

The results (Figures 5.7, 5.8, 5.9, and 5.10) for the *Recognition Load* as a function of F_t, on all four experiment data sets, demonstrate the same tendency. The *Recognition Load*, decreases asymptotically to a typical load value, and we expect this typical load value to be identical to the load on the agent when executing its task without running the recognition process on the observed agent (as is presented in Equation 3.6). In all four experiment data sets the results coincide with the model values, thus the *Recognition Load* model, which we suggested (Equation 3.7) is a good approximation for the *Recognition Load* as a function of F_t in an integrated system. Furthermore, this work shows an empirical way to reveal the value of r in the model Equation 3.7, which represents the computational load of the recognition process in the integrated system, and which is depend on the recognition algorithm and on the scenario.

Recognition Error **Model**

In order to apply an analytic model for the *Recognition Error* as a function of F_t for the recognition process in an integrated system, we discuss the characteristics of a typical signal describing the recognition process. In this work the linear decreasing signal is used according to the assumption that when dealing with few hypotheses, most of the signals can be approximated, to a degree, by a combination of three typical characteristics: a steady state signal, a linear decreasing signal and a linear increasing signal.

Our hypothesis is that the linear *Recognition Error* model as a function of k, according to Equation 3.14, is a good approximation for the non-linear decreasing signals of the recognition process for the single agent recognition case in an integrated system.

The results (Figures 5.11, 5.12, 5.13, and 5.14) for the *Recognition Error* as a function of k, in all four experiment data sets, demonstrate the same tendency. As the sampling interval increases, the *Recognition Error* increases. The *Recognition Error* model values are within the variance of the three scenarios (out of four) empirical results, pointing out that the linear model is a practical approximation for the non-linear decreasing signal. In addition, in the scenario in which the model values are out of the variance of the experimental results, the model can be used as a lower bound for the *Recognition Error* of the recognition process in an integrated system. Nevertheless, the recognition error model is not as accurate as the load model, and farther work is required on developing it.

Dynamic Sampling compared with Fixed Sampling

In this work a number of general heuristics are suggested: The Fixed FoH, the Uncertainty FoH, the Load heuristic and the Combination FoH. Their performances are compared by a graph of *Recognition Load* as a function of *Recognition Error*.

Our hypothesis is that the dynamic sampling (Uncertainty, Load and Combination FoH) can reduce the *Recognition Load* while not downgrading the *Recognition Error* compared to the fixed sampling (Fixed FoH).

According to the results (Figures 5.15, 5.16, 5.17, and 5.18), the Fixed FoH curve demonstrates the tendency: as the sampling interval increases, the *Recognition Error* increases. *Recognition Load*, however, decreases asymptotically to a typical load value. Also, the performances of the Uncertainty and of the Combination FoH decrease asymptotically to the same typical load value as the Fixed FoH and converge to the same performance of the Fixed FoH, due to the fact that there is a minimal load on the executer agent and no improvement can be achieved.

To summarize the performances of the suggested heuristics:

1. For the linear decreasing signals the dynamic heuristics achieved no improvement compared with the Fixed FoH.

2. For the monotonic decreasing signals the Combination FoH achieved better performance.

3. For the non-monotonic decreasing signals (in different two systems) the Uncertainty FoH achieved better performance.

Using a FoH is a practical tool to derive the suitable sampling heuristic depends on the performance's demands of the integrated system. For example, in the non-monotonic decreasing scenario on the VR-Forces simulator, if the requirement of the system is not to exceed *Recognition Error* of 0.08, then, the system might use the Uncertainty heuristic with $H = 5$. If the requirement is not to consume more than 0.29 *Recognition Load*, then, the system might use the Uncertainty heuristic with $H = 7$.

Expanding to Multiple recognition processes
This research shows that the sampling approach reduces the computational load of multi-agent intention recognition processes. By applying different dynamic F_t to each of the intention recognition processes, better performance is obtained.

The recurrent and consistent behavior of the two simulators for the non-monotonic signal (Figures 5.21 and 5.22), i.e. *Recognition Load* and *Recognition Error* that were demonstrated in this research, strongly points out that a baseline of *Recognition Load* as a function of the number of observed agents, I, as well as a function of fixed F_t can be modeled by Equation 3.25. In addition, we base the assumption that all the observed agents impose the same computational load, r, as it demonstrated in Figures 5.19 and 5.20.

Figures 5.23 and 5.24 demonstrate the performances of the Fixed FoH, the Uncertainty FoH, the Load heuristic and the Combination FoH for the multi case for the non-monotonic decreasing signal on SBR and M-DIESEL simulators. When using the dynamic sampling, each of the recognition process of the i-th observed agent is triggered according to its dynamic F_{ti}. We demonstrate that with the Uncertainty FoH the performance of the entire system is improved. The tendency of the performance of the dynamic FoH are much the same as in the single case since the scenarios of the multiple observed agents are the same.

Evaluation of a recognition process in an integrated system
In addition to the suggested sampling method, this work describes in details a

benchmark to evaluate the performance of a recognition process which is integrated in a system. The benchmark is:

1. Define a formal and common language. In this work the $h[n]$, $h[n]^*$, $L[j]$,N, H, ZOH($h[n]$), F_t, F_t^* were defined.

2. Define the system parameters to be evaluated and the measurement method. In this work the *Recognition Load* and the *Recognition Error* were chosen to be the system parameters to be evaluated. The *Recognition Load* was measured according to Equation 3.2 and the *Recognition Error* was measured according to Equation 3.4.

3. Base the chosen system parameters on models. Once a baseline is defined, the system performance can be evaluated, avoiding the experiments that should be done to achieve parameters measurements. The baseline can also be applied to estimate the optimal recognition process depending on system requirements. In this work we show that the *Recognition Load* model (Equation 3.7) and the *Recognition Error* model (Equation 3.14) are good approximations for the system performance with Fixed sampling FoH.

4. Determine a protocol to compare the performances of different heuristics for the recognition process. In this work we present and compare the results of the fixed sampling FOH and of the dynamic sampling heuristics by the performance graph - the *Recognition Load* as a function of the *Recognition Error*.

This benchmark for system evaluation is also practical for comparison between heuristics in order to chose which heuristic is better to use for satisfying predefined system performance requirements. For example, in the case of the 'SBR monotonic non-linear scenario' if the requirement of the system is not to exceed *Recognition Error* of 0.12, then, the system might chose the Load heuristic although it improves load reduction less than the Uncertainty FoH compared with the Fixed FoH but with less recognition quality. If the requirement is not to exceed *Recognition Error* of 0.14, then the Uncertainty FoH with $H = 10$ is better than the Load heuristic (see Table 5.2).

The sampling method as a basis for applying general heuristics
The sampling method, triggering each of the recognition processes according to a controlled dynamic F_t, enables using a wide range of heuristics. These heuristics provide opportunities to deal with dynamic system constraints, like new knowledge about the observed agent or new requirement of the executer agent, the decision of whom and when perform the recognition process. We believe that, base on our research, other ways of dynamically changing the F_t parameter, can be explored for the single recognition case and for the multi recognition case. Several suggested heuristics are:

1. In social comparison system, the focus of attention, i.e., the level of importance of each recognized agent, changes in time. Thus, observation target is changed. Our method can easily accomplish the heuristic depend on the focus of attention as: $F_{ti}[n] = (A - FOA[n](i))F_t^*$, where A is a parameter of the heuristic, $FOA[n](i)$ is the value of the focus of attention of the i-th observed agent at time n. The F_t of the i-th agent is changed according to its attention, the sampling frequency should increase upon increasing of the FOA.

2. In a system in which the recognition process lasts for a long time and the intention of the observed agent is varied frequently, a heuristics depends on the history of changes in its intentions, can be defined as: $F_t = (C - \sum_{i=n-T}^{i=n} c[i])F_t^*$, where C is the heuristic parameter, T is the memory interval and $c[i]$ is the number of changes in the observed agent intentions from the last time it was observed.

3. A heuristic that changes the F_t parameter as a function of the recognized agent's intention, informed by the contents of the hypotheses list which is returned from the recognition process. For example, if one of the hypotheses is *Attack*, meaning the observed agent is potentially recognized as an attacker of the executer, the executer can increase the recognition process frequency, in order to guard from it's steps, or decrease the recognition process frequency, in order to save computational resources for its own execution activities. The decision what to do, according to the observed agent's intention, depends on the system application.

4. A Time-Division Multiplexing (TDM) heuristic in which the recognizer divides the time into several recurrent timeslots with fixed length, one for each recognition process. The intention recognition process of agent 1 runs at timeslot 1, the intention recognition process for agent 2 runs at timeslot 2, etc. One TDM frame consists of one timeslot per agent. After the last agent the cycle starts all over again with a new frame, starting with the second sample of the recognition algorithm. Another heuristic can be based on the TDM heuristic where the monitoring order within each TDM frame will be determined by the agent's priority.

5. A heuristic which is based on clustering similar agents and monitoring only one representative agent from each cluster.

6. A heuristic that applies different values of fixed sampling interval to each of the i-th observed agent according to predefined preference and interlaces it with Time-Division Multiplexing in order to spread the load among the time-slots.

7. Another direction is to allow each F_{ti} to be dynamically changed according to a different heuristic.

Requirements from a recognition algorithm for using sampling method

In order to be integrated in a system which use sampling method, the recognition algorithm might need to meet the following requirements:

1. The recognition algorithm which is run in the integrated system is complete. In other words, at least one of the hypotheses the recognition algorithm returns is the correct behavior leading to a successful recognition. Preferably, it should be minimally complete, but this is not strictly required.

2. The recognition algorithm's computational load is affected by the number of the hypotheses that it maintains. Therefore, the *Recognition Load* can be measured by a function of the size of the hypotheses list which is returned by the recognition algorithm.

3. Our sampling approach relies on the ability of the recognition algorithm to freeze itself and then resume, while taking into account the fact that it missed observations (e.g., as was done by Kaminka et al. [32], Avrahami-Zilberbrand et al. [9, 7]).

4. Each of the maintained hypotheses is considered to consume the same amount of computational resources as the others, independently to the way in which the hypotheses are generated.

5. The recognition algorithm is executed only at the time points when the integrated system activates it. Thus, we use accumulated value over time, to measure the computational resources consumed by the recognition algorithm up to now, in order to express the *Recognition Load* criterion.

6. Our recognition process has no previous knowledge about the observed agent intentions, such as when it might change its goals. In other words, it has no access to an oracle (otherwise, why recognition at all?).

In Section 3.3 we discussed the advantage of an algorithm which supplies a ranked list of hypotheses to be integrated in a system in the light of the computational load. When hypotheses list is yielded with different probabilities, the uncertainty is decreased by the information that the least probable state, can be removed from the possible set of future hypotheses list. Therefore, when using different probabilities, the entropy of a process is smaller, relatively to the case of equal probabilities. For example, if the recognition algorithm returns 3 hypotheses, with probability $1/3$, then the entropy is: 1.58. When a recognition algorithm returns 3 hypotheses with the probabilities: $1/2$, $3/8$ and $1/8$, the entropy is: 1.395, less than the first one.

The property of yielding probabilities with the hypotheses list, can be a significant advantage for a recognition algorithm to be preferred by an integrated system. Examples for a recognition algorithm which supplies different probabilities are Geib and Goldman [22], and Avrahami-Zilberbrand [8], AHMEM [11], etc.

Mirroring

Integrating intention recognition via *Mirroring* seems to be the right technique according to its achievements: simultaneous performing, multi-intention recognitions, good software engineering, and improving performance by applying *Temporal Monitoring Selectivity* approach. We believe that the *Mirroring* technique and the way the brain recognizes behavior are very related.

Bibliography

[1] A. Aldroubi and K. Gröchenig. Nonuniform sampling and reconstruction in shift-invariant spaces. *Society for Industrial and Applied Mathematics*, 43:585–620, 2001.

[2] J. F. Allen. Recognizing intentions from natural language utterances. *Artificial Intelligence*, 15(3), 1980.

[3] D. Assaf. A dynamic sampling approach for detecting a change in distribution. *The Annals of Statistics*, 16:236–253, 1988.

[4] D. Assaf, M. Pollak, Y. Ritov, and B. Yakir. Detecting a change of a normal mean by dynamic sampling with probability bound on false alarm. *The Annals of Statistics*, 21:1155–1165, 1993.

[5] D. Assaf and Y. Ritov. Dynamic sampling procedures for detecting a change in the drift of brownian motion: A non-bayesian model. *The Annals of Statistics*, 17:793–800, 1989.

[6] D. Assaf and Y. Ritov. Dynamic sampling applied to problems in optimal control. *Journal of Optimization Theory and Applications*, 95(3):565–580, 1997.

[7] D. Avrahami-Zilberbrand and G. A. Kaminka. Fast and complete symbolic plan recognition. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-05)*, 2005.

[8] D. Avrahami-Zilberbrand and G. A. Kaminka. Hybrid symbolic-probabilistic plan recognizer: Initial steps. In *Proceedings of the AAAI Workshop on Modeling Others from Observations (MOO-06)*, 2006.

[9] D. Avrahami-Zilberbrand, G. A. Kaminka, and H. Zarosim. Fast and complete plan recognition: Allowing for duration, interleaved execution, and lossy observations. In *Proceedings of the IJCAI Workshop on Modeling Others from Observations (MOO-05)*, 2005.

[10] T. Balch, Z. Khan, and M. Veloso. Automatically tracking and analyzing the behavior of live insect colonies. In J. P. Müller, E. Andre, S. Sen, and C. Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 521–528, Montreal, Canada, 2001. ACM Press.

[11] H. Bui. A general model for online probabilistic plan recognition. In *International joint conference on Artificial Intelligence*, 2003.

[12] S. Carberry. *Plan Recognition in Natural Language Dialogue*. MIT Press, Cambridge, MA, USA, 1990.

[13] S. Carrbery. Techniques for plan recognition. *User Modeling and User-Adapted Interaction*, 11:31–48, 2001.

[14] J. Catlett. Peepholing: Choosing attributes efficiently for megainduction. In *Proceedings of the Ninth International Workshop on Machine Learning*, pages 49–54, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[15] H. Chalupsky. SIMBA: Belief ascription by way of simulative reasoning. Technical Report 96-18, 31 1996.

[16] E. Charniak and R. P. Goldman. A Bayesian model of plan recognition. *Artificial Intelligence*, 64(1):53–79, Nov. 1993.

[17] P. R. Cohen, C. R. Perrault, and J. F. Allen. Beyond question answering. In W. G. Lehnert and M. H. Ringle, editors, *Strategies for Natural Language Processing*, pages 245–274. Erlbaum, Hillsdale, NJ, 1982.

[18] L. Festinger. A theory of social comparison processes. *Human Relations*, 7:117–140, 1954.

[19] L. Fogassi, P. F. Ferrari, B. Gesierich, S. Rozzi, F. Chersi, and G. Rizzolatti. Parietal lobe: From action organization to intention understanding. *Science*, 308(5722):662–667, 2005.

[20] N. Fridman and G. A. Kaminka. Towards a cognitive model of crowd behavior based on social comparison theory. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence (AAAI-07)*, 2007.

[21] C. W. Geib and R. P. Goldman. Plan recognition in intrusion detection systems. In *In DARPA Information Survivability Conference and Exposition (DISCEX)*, June 2001.

[22] C. W. Geib and R. P. Goldman. Probabilistic plan recognition for hostile agents. In *Proceedings of the Fourteenth International Florida Artificial Intelligence Research Society Conference*, pages 580–584. AAAI Press, 2001.

[23] C. W. Geib and M. Steedman. On natural language processing and plan recognition. In M. M. Veloso, editor, *International joint conference on Artificial Intelligence*, pages 1612–1617.

[24] M. A. Girshick and H. Rubin. A bayes approach to a quality control model. *The Annals of Mathematical Statistics*, 23:114–125, 1952.

[25] B. J. Grosz and S. Kraus. Collaborative plans for complex group actions. *Artificial Intelligence*, 86:269–358, 1996.

[26] M. J. Huber and E. H. Durfee. Deciding when to commit to action during observation-based coordination. In *Proceedings of the First International Conference on Multiagent Systems (ICMAS-95)*, pages 163–170, 1995.

[27] G. H. John and P. Langley. Static versus dynamic sampling for data mining. In E. Simoudis, J. Han, and U. M. Fayyad, editors, *The 2nd International Conference on Knowledge Discovery and Data Mining, KDD*, pages 367–370. AAAI/MIT Press, 2–4 1996.

[28] G. A. Kaminka and M. Bowling. Towards robust teams with many agents. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, 2002.

[29] G. A. Kaminka and I. Frenkel. Flexible teamwork in behavior-based robots. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, 2005.

[30] G. A. Kaminka and I. Frenkel. Integration of coordination mechanisms in the BITE multi-robot architecture. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA-07)*, 2007.

[31] G. A. Kaminka and N. Fridman. Social comparison in crowds: A short report. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-07)*, 2007.

[32] G. A. Kaminka, D. V. Pynadath, and M. Tambe. Monitoring teams by overhearing: A multi-agent plan recognition approach. *Journal of Artificial Intelligence Research*, 17, 2002.

[33] G. A. Kaminka and M. Tambe. Social comparison for failure detection and recovery. In *Intelligent Agents IV: Agents, Theories, Architectures and Languages (ATAL-97)*, number 1365 in Lecture Notes in Artificial Intelligence, pages 127–141. Springer Verlag, 1998.

[34] G. A. Kaminka and M. Tambe. Robust multi-agent teams via socially-attentive monitoring. *Journal of Artificial Intelligence Research*, 12:105–147, 2000.

[35] G. A. Kaminka, M. Tambe, and C. M. Hopper. The role of agent modeling in agent robustness. In *AI meets the real world: Lessons learned (AIMTRW-98)*, 1998.

[36] G. A. Kaminka, A. Yakir, D. Erusalimchik, and N. Cohen-Nov. Towards collaborative task and team maintenance. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-07)*, 2007.

[37] H. A. Kautz. A formal theory of plan recognition and its implementation. In J. F. Allen, H. A. Kautz, R. Pelavin, and J. Tenenberg, editors, *Reasoning*

About Plans, pages 69–125. Morgan Kaufmann Publishers, San Mateo (CA), USA, 1991.

[38] H. A. Kautz and J. F. Allen. Generalized plan recognition. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pages 32–37. AAAI press, 1986.

[39] J. E. Laird. It knows what you're going to do: adding anticipation to a quakebot. In *Proceedings of the Fifth International Conference on Autonomous Agents*, Montreal, Canada. ACM Press.

[40] P. Langley and S. Sage. Induction of selective bayesian classifiers. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 399–406. Seattle, WA: Morgan Kaufmann, 1994.

[41] A. W. Moore and M. S. Lee. Efficient algorithms for minimizing cross validation error. In *International Conference on Machine Learning*, pages 190–198, 1994.

[42] A. Newell. *Unified Theories of Cognition.* Harvard University Press, Cambridge, Massachusetts, 1990.

[43] E. S. Page. A test for a change in parameter occuring at an unknown point. *Biometrika*, 42:523–526, 1955.

[44] E. S. Page. On problems in which a change of parameters occurs at an unknown point. *Biometrika*, 44:248–252, 1957.

[45] M. Pollak. Optimal detection of a change in distribution. *The Annals of Statistics*, 13:206–227, 1985.

[46] M. Pollak. Average run lengths of an optimal method of detecting a change in distribution. *The Annals of Statistics*, 15:749–779, 1987.

[47] J. G. Proakis and D. G. Manolakis. *Digital Signal Processing Principles, Algorithms, and Applications.* Prentice-Hall International, 1996.

[48] D. V. Pynadath and M. P. Wellman. Probabilistic state-dependent grammars for plan recognition. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI-2000)*, pages 507–514, 2000.

[49] T. Raines, M. Tambe, and S. Marsella. Towards automated team analysis: A machine learning approach. In *Third international RoboCup competitions and workshop*, 1999.

[50] A. S. Rao. Means-end plan recognition – towards a theory of reactive recognition. In *Proceedings of the International Conference on Knowledge Representation and Reasoning (KR-94)*, pages 497–508, 1994.

[51] G. Rizzolatti. The mirror neuron system and its function in humans. *Anatomy and Embryology*, 210(5–6):419–421, 2005.

[52] S. Saria and S. Mahadevan. Probabilistic plan recognition in multiagent systems. In *In Proceedings of International Conference on Artificial Intelligence and Planning Systems.*, 2004.

[53] C. Schmidt, N. Sridhan, and J. Goodson. The plan recognition problem: an intersection of psychology and artificial intelligence. *Artificial Intelligent*, 11:45–83, 1978.

[54] A. Sen and M. S. Srivastava. On test for detecting change in mean. *The Annals of Statistics*, 3:98–108, 1975.

[55] A. N. Shiryayev. *Optimal Stopping Rules*. Springer, New York, 1978.

[56] Soar. http://sitemaker.umich.edu/soar/home/, 2006.

[57] M. S. Srivastava and Y. Wu. Dynamic sampling plan in shiryayev-roberts procedures for detecting a change in the drift of brownian motion. *The Annals of Statistics*, 22:805–823, 1994.

[58] C. Stauffer, W. Eric, and L. Grimson. Learning patterns of activity using real-time tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):747–757, 2000.

[59] M. Tambe. Tracking dynamic team activity. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, August 1996.

[60] M. Tambe and P. S. Rosenbloom. RESC: An approach to agent tracking in a real-time, dynamic environment. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-95)*, August 1995.

[61] M. Technologies. Vr-forces. http://www.mak.com/vrforces.htm, 2006.

[62] A. Yakir and G. A. Kaminka. An integrated development environment and architecture for Soar-based agents. In *Innovative Applications of Artificial Intelligence (IAAI-07)*, 2007.

[63] X. Zhang, P. Willett, and Y. Bar-Shalom. Uniform versus nonuniform sampling when tracking in clutter. *IEEE Transactions on Aerospace and Electronic Systems*, 42(2), 2006.

[64] I. Zuckerman, S. Kraus, J. S. Rosenschein, and G. A. Kaminka. An adversarial environment model for bounded rational agents in zero-sum interactions. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-07)*, 2007.

Appendix A

Soar code examples for *Mirroring* implementation

Here are some examples of rules, which are written in Soar. Soar works by testing the *if* parts of rules. These *if* parts are called conditions and appear before the arrow symbol in the rule sentence. If all of the conditions of a rule are true in the current situation, the *then* parts, or actions, of the rule are executed, which usually involves making changes to working memory, which holds all of the dynamic data structures in a Soar program. The *then* parts appear after the arrow symbol in the rule sentence. For introduction to Soar, see [56]

The rule of building a new *state* on the top state, for a new observed agent is:

```
sp {mirroring*initialize*state
   :o-support
   (state <s> ^superstate nil)
   (<s> ^mode RECOGNITION)
   (<s> ^state <state>)
   (<state> ^executer yes
            ^inputs <m-in>)
   (<m-in> ^objects <m-objs>
           ^self.time <t>)
   (<m-objs> ^ent <m-ent>)
   (<m-ent> ^id <id>
            ^health <ht>
            ^movement <mov>
            ^posture <posture>)
  {(<m-ent> ^position <pos>)
   (<pos> ^coordinate world
          ^location <loc>
          ^heading <hd>
          ^pitch <pc>)
   (<loc> ^x <x>
          ^y <y>
          ^z <z>)}
  {(<m-ent> ^position <pos2>)
   (<pos2> ^coordinate self
           ^azimuth <az>
           ^elevation <el>)}
 -{(<s> ^state <any>)
   (<any> ^executer no)
   (<any> ^inputs.self.id <id>)}
-->
   (<s> ^state <state1>)
   (<state1> ^executer no
             ^type state
             ^inputs <o-in>
             ^intention <int>)
   (<int> ^count 0)
   (<o-in> ^objects <any>
           ^self <o-self>
           ^freeze <frz>)
   (<o-self> ^id <id>
             ^time <t>)
   (<o-self> ^position <o-pos>
             ^health <ht>
```

```
              ^movement <mov>
              ^posture <posture>
              ^speed normal
              ^stamina excellent)
   (<o-pos> ^location <o-loc>
            ^azimuth <az>
            ^elevation <el>
            ^pitch <pc>
            ^heading <hd>)
   (<o-loc> ^x <x>
            ^y <y>
            ^z <z>)
   (<frz> ^timeout 80)
   (<o-in> ^update_input yes)
}
```

The rule is executed just if the executer in RECOGNITION mode and a new observed agent, which has not have its own *state* yet, is identified. Then a new *state* structure is created with a variable $state1.executer = no$. $state1$, is a structure that reflects the perception of the observed agent, thus, the sub-structure $state1.input.self$ hold the knowledge about its own id, position, orientation, health, speed, stamina and so on.

When a new *state* is created, its input layer need to be updated, therefore in the above rule the $state1.update\_input$ set to true. As a results, all the translation rules are executed. The translation rules interpret the input layer data of the executer's *state* from its perception into the the observed agent's perception. An example for such a rule is:

```
sp {mirroring*translate*inputs*self*location*x
   :o-support
   (state <s> ^state <state>
              ^state <state1>)
   (<state> ^executer yes)
   (<state> ^inputs.objects <objs>)
  {(<objs> ^ent <ent>)
   (<ent> ^id <id>)
   (<ent> ^position <pos>)
   (<pos> ^coordinate world)
   (<pos> ^location <loc>)
   (<loc> ^x <x>)}
   (<state1> ^executer no)
   (<state1> ^inputs <o-in>)
  {(<o-in> ^self <o-self>)
   (<o-self> ^id <id>)
   (<o-self> ^position <o-pos>)
   (<o-pos> ^location <o-loc> )
   (<o-loc> ^x <o-x> {<> <x>})}
   (<o-in> ^update-input yes)
-->
   (<o-loc> ^x <o-x> -)
   (<o-loc> ^x <x> +)
}
```

This rule updates the x location of the observed agent, by reads it from the *entity* sub-structure in the *state* structure of the executer and write it in the sub-structure *self* in the *state* of the observed agent.

All the application rules are written with no consideration who's the *state* belongs to. For example, the next rule is the precondition rule of the behavior *lay-down*. Its occurrence is a precondition for this behavior to become active in the agent recipe. In case it becomes an active behavior on the observed agent's recipe, it considered as a potential hypothesis for the observed's intention.

```
sp {ops*lay-down*precondition
    (state <s> ^state <state>)
    (<state> ^recipe <recipe>
             ^events <events>
             ^inputs <inputs>)
    (<recipe> ^rsc <rsc>)
```

```
   (<rsc> ^sc <sc>)
   (<sc> ^name lay-down)
   (<inputs> ^self.posture <posture> <> Lie )
-->
   (<sc> ^precondition <precondition>)
   (<precondition> ^value true )
   (<precondition> ^dparams nil)
   (<precondition> ^type success )
}
```

In order to operate the simulator, according to the executer behavior, the agent writes a command on the output layer. An example to command the simulator to *lay-down* is:

```
sp {ops*lay-down*apply
   :o-support
   (state <s> ^state <state>)
   (<state> ^recipe <recipe>
            ^events <events>
            ^outputs <outputs>)
   (<recipe> ^rsc <rsc>)
   (<rsc> ^sc <sc>)
   (<sc> ^name <cname> lay-down
         ^active yes
         ^dparams <dparams>)
-->
   (<outputs> ^set-posture <kneel>)
   (<kneel> ^command-id 19)
   (<kneel> ^posture Lie )
   (<kneel> ^move-type stopped)
}
```

This rules is an example for an applicable rule which is not distinguish who's agent this *state* belongs to. This rule writes the command-id 19 on the *output* layer the simulator reacts accordingly and the agent lays down. How can we be sure that only command from the executer state*state* will be written to the simulator?

An *output* layer is built as a bridge layer on the top of the output-link layer, which is a shared memory between the agent program and the simulator program. When the observed agent's *state* is built, it created without an *output* sub-structure, thus, the above rule never be executed for the observed agent, according to the condition of the existence of the *output* sub-structure. An *output* layer is created just for the executer by this rule:

```
sp {output-link*generate
   (state <s> ^state <state>)
   (<state> ^executer yes
            ^io.output-link <il>)
-->
   (<state> ^outputs <il>)
}
```