TÉCNICO LISBOA

Universidade de Lisboa

Instituto Superior Técnico

Decision-Making under Uncertainty for Real Robot Teams

João Vicente Teixeira de Sousa Messias

Orientador: Doutor Pedro Manuel Urbano de Almeida Lima

Co-Orientador: Doutor Matthijs Theodor Jan Spaan

Tese especialmente elaborada para obtenção do Grau de Doutor em Engenharia Electrotecnica e de Computadores

Tese Provisória

Setembro de 2013

Título: Tomada de Decisão sob Incerteza para Equipas de Robôs Reais

Nome: João Vicente Teixeira de Sousa Messias

Doutoramento em: Engenharia Electrotécnica e de Computadores

Orientador: Prof. Doutor Pedro M. U. A. Lima

Co-orientador: Doutor Matthijs T. J. Spaan

Resumo:

Este trabalho está focado na aplicação de métodos de Teoria de Decisão (TD) em cenários de Robótica Cooperativa. São abordados problemas teóricos e práticos envolvidos na modelação de processos de tomada de decisão sob incerteza para agentes físicos.

A família existente de métodos de TD é revista, e as respectivas limitações, relativas à modelação de sistemas multi-robô, são identificadas. São propostas novas metodologias, e é investigada a aplicação de técnicas actuais, para superar essas limitações.

Como abordagem ao problema da minimização de comunicação multi-agente, apresenta-se um método novo para obter políticas de comunicação para Processos de Decisão de Markov Multi-agente Parcialmente Observáveis (MPOMDPs), que pode ser usado antes da execução do sistema.

Demonstra-se como a tomada de decisão multi-robô pode ser descrita como um processo controlado por eventos, discutindo as vantagens dessa interpretação. Investiga-se a aplicação de Processos de Decisão Semi-Markov Generalizados (GSMDPs) em equipas de robôs reais.

Aborda-se também o problema de observabilidade parcial para sistemas multiagente controlados por eventos. Introduz-se a classe de MPOMDPs Controladas por Eventos, apresentam-se resultados teóricos e empíricos da mesma, e consideram-se domínios Semi-Markov.

Por fim, descreve-se a implementação dos métodos propostos num sistema real de vigilância multi-robô, e documenta-se o software desenvolvido como parte deste trabalho.

Palavras-chave: Planeamento sob Incerteza; Processos de Decisão de Markov; Sistemas Multiagente; Sistemas de Eventos Discretos; Robótica Cooperativa; Redes de Robôs; Comunicação Multi-Robô; Observabilidade Parcial; Vigilância Multiagente; Robôs Futebolistas.

Title: Decision-Making under Uncertainty for Real Robot Teams

Abstract:

This work focuses on the application of Decision-Theoretic (DT) frameworks to scenarios in Cooperative Robotics. We address both theoretical and practical issues involved in modeling decision-making under uncertainty for physical agents.

We review the family of different existing DT approaches, and identify their limitations regarding the modeling of multi-robot systems. We then propose novel methodologies, and investigate the applicability of current methods and frameworks, to address those limitations.

To address the problem of minimizing multiagent communication, we present a novel method to determine efficient communication policies for Multiagent Partially Observable Markov Decision Processes (MPOMDPs), that operates prior to system execution.

We show how multi-robot decision-making can be described as an event-driven process, and discuss the practical advantages of that interpretation. We investigate the practical application of the Generalized Semi-Markov Decision Process (GSMDP) framework to a team of real robots.

We also address the problem of partial observability for event-driven multiagent systems. We introduce the Event-Driven MPOMDP framework, provide both theoretical and empirical results related to its application, and extend it to Semi-Markov domains.

Finally, we describe the implementation of our methods to a real multi-robot surveillance system, and document the software tools that were developed as part of this work.

Key-words: Planning under Uncertainty; Markov Decision Processes; Multiagent Systems; Discrete Event Systems; Cooperative Robotics; Networked Robot Systems; Multi-Robot Communication; Partial Observability; Multiagent Surveillance; Soccer Robots.

Acknowledgements

Funding Acknowledgments

This work was funded by Fundação para a Ciência e a Tecnologia (FCT), through the PhD Student Scholarship SFRH/BD/44661/2008, through ISR/IST pluriannual funding (PIDDAC program funds), reference PEst-OE/EEI/LA0009/2013, and by a research grant (Bolsa de Investigação para Mestre) at INESC-ID in the MAIS+S project, part of the Carnegie Mellon - Portugal Program (reference CMU-PT/SIA/0023/2009).

Personal Acknowledgments

Four years ago, I was on the threshold of a great change in my life. I knew it, as I was embarking to my first RoboCup experience, but I could only imagine what was still to come. Looking back on this journey now, on all that I've seen, and all that I've learned, I can say that it has far exceeded my imagination.

I owe this invaluable opportunity to Professor Pedro Lima, and I am sincerely thankful for it. I hope that the contributions of my work, and what comes after it, can ever live up to all that I've been given.

I must also extend my special thanks to my co-advisor Matthijs Spaan, for his guidance and ideas; his patience in helping out with our papers and experiments, even in the long hours before the tough deadlines; and all his help and companionship during my short adventure in the Netherlands. This work wouldn't have been possible without his contribution.

In a thesis about robot teams, I wonder if I can thank a team of robots. The SocRob MSL robots and I went through so much, that at times I felt that I knew the quirks and the "personality" of each of them. I've learned virtually all of what I know of Robotics and Artificial Intelligence while working on those robots in some way. But, of course, I wasn't alone, and the most valuable members of the team were not playing

in the soccer field, but worrying on the sidelines. Thanks to João Reis, Aamir Ahmad, Miguel Serafim and all the older (and newer!) members of the SocRob team. There's a bond between us that comes from the special kind of panic that only a RoboCup (and Robótica) competition can instill. We might not have scored many goals, but in all that we have learned to do with the little that we had, we have won the greatest prize of all.

I'd also like to thank all the members of the MAIS+S project, especially José Carlos Castillo and Stefan Witwicki, for their help. We did a lot of work in a very short time, and a large part of this thesis depended on it.

To all of my family, and in particular my mother, and my father, I leave my special thanks, for supporting my life in its most chaotic moments. I hope you can be proud of this work, which is also yours.

And, finally, to my better half, the love that I've found, and that grew, during all these years. Thanks Ana. We're on the threshold of another great change, and we'll be crossing it together.

Contents

List of Figures

LIST OF FIGURES

LIST OF FIGURES

List of Tables

Chapter 1

Introduction

1.1 Motivation: Planning Under Uncertainty in the Real World

Planning and decision-making are fundamental processes in many human activities. The field of Operations Research (OR) provides the mathematical basis to quantitatively evaluate decisions, allowing the best plans to be identified, in terms of their perceived utility or efficiency in accomplishing given goals. Many of the techniques developed in this field are therefore applicable to the domain of Artificial Intelligence (AI), in that they may provide decision-making capabilities to autonomous agents. One of the areas of research common to both AI and OR is that of Decision Theory (DT), in which the problem of decision-making under uncertainty is included. Although DT has a wide scope that is not limited to decision making in probabilistic settings, in this work, we will refer to DT in that context.

DT techniques address the problem of decision-making in environments in which it is necessary to take into account uncertainty in the actions and/or observations of an agent, as is the case in many real world scenarios. A Markov Decision Process (MDP) is a widely known and well-studied mathematical framework to model problems where the outcome of an agent's actions is probabilistic, but full knowledge of the state of the agent is assumed (Bellman, 1957a). For this class of problems, several algorithms exist that provide optimal and approximate solutions to MDPs in reasonable time.

When the knowledge of the agent is insufficient to directly determine its state, as it happens with a mobile robot with noisy sensors, the uncertainty of its observations

must also be considered. Such problems are within the domain of application of the Partially Observable Markov Decision Process (POMDP) framework. Efficient planning algorithms for POMDPs have been subject to active research for over four decades (Sondik, 1971). While solving a POMDP optimally in its general case is a difficult problem, several algorithms exist to compute approximate solutions to those models, which allows them to be scaled up to the point that they can be applied to the control of robotic agents (Kurniawati et al., 2008; Pineau et al., 2003; Spaan and Vlassis, 2005).

In certain applications, however, a single agent model is not enough to model the full scale of the problem. Such is the case, for example, in cooperative robotics, where multiple agents must work together to achieve a common goal. For that reason, there has been an increasing interest in the topic of multiagent systems, both from the perspective of Robotics (Asama, 2009; Lima and Custodio, 2005) as well as that of AI (Shoham and Leyton-Brown, 2009). Various generalizations of the MDP framework to the problem of multiagent decision-making under uncertainty have also been proposed, for example: Decentralized MDPs (Dec-MDPs) and Decentralized Partially Observable MDPs (Dec-POMDPs) (Bernstein et al., 2002) which assume cooperative agents, but are intractable to solve optimally without communication (NEXP-Complete for optimal finite-horizon solutions)[1]; Network Distributed POMDPs (ND-POMDPs) (Nair et al., 2005) which are scalable to many agents, but assume transition and observation independence; Interactive POMDPs, which include models of each agent in the state space (Gmytrasiewicz and Doshi, 2005); and the simpler, fully centralized Multiagent MDPs (MMDPs) / POMDPs (MPOMDPs), in which free communication between agents is assumed (Boutilier, 1996; Pynadath and Tambe, 2002), but where the planning problem is of equivalent complexity to that of solving a single-agent problem defined over the whole team (Pynadath and Tambe, 2002).

Despite the proliferation of the theoretical advances in multiagent MDPs and related models, there have been very few reported applications of these methods to the decision-making of teams of robots. Notable exceptions do exist, for fully observable settings Bowling and Veloso (2003); Matarić (1997), and also for partially observable domains Capitán et al. (2013); Emery-Montemerlo et al. (2005).

[1]Recent approximate solution methods for Dec-POMDPs include (Kumar and Zilberstein, 2010; Oliehoek et al., 2008a, 2013; Pajarinen and Peltonen, 2011).

Nevertheless, this apparent lack of applications of established DT methods to multi-robot systems is not only endemic of MDP-based approaches; it is a commonplace problem in the field of AI. Although research on multi-robot systems has been active for at least the past two decades, it has mostly focused on the problems of multi-robot cooperative perception and navigation (Asama, 2009). In contrast, there has been relatively little exploration of the theoretical aspects of decision-making that are characteristic of multi-robot systems, from an AI perspective. The problem of multi-robot decision-making has been typically interpreted as a problem of task allocation (Gerkey and Mataric, 2003; Lemaire et al., 2004), or behavior scheduling and coordination (Dias et al., 2006), and addressed through formalisms that incorporate few of the insights of state-of-the-art AI methodologies.

The fact that most multi-robot systems are inherently partially observable (at least from the perspective of each individual agent), coupled with the so-called "curse of dimensionality", are the most frequently evoked explanation for this difficulty of approaching those domains from the perspective of AI (and of DT, in particular). Solving a general, unstructured multiagent version of a POMDP, for example, is exponentially harder than solving its single-agent instantiations. Another important issue is that many of these methods operate over symbolic representations of the multiagent system (such as abstract *states*, *actions*, and *observations*), and the problems involved in defining these abstract representations, and in mapping between them and the quantities involved in physical multiagent systems, are often overlooked.

This work addresses the problems of modeling and implementing multi-robot decision-making through techniques grounded on the theory of MDPs. By doing so, it will attempt to bridge the apparent gap between theory and practice in this field of study, and draw further interest into the real-world application of decision-theoretic frameworks. Although the applications that will act as case studies concern networked robot systems, other scenarios involving multiple physical agents may provide suitable domains for the application of the techniques which are discussed here.

1.2 Related Approaches to Multiagent (and Multi-Robot) Decision-Making

As a long-standing line of research in Artificial Intelligence and Operations Research, the problem of multiagent decision-making has been approached in several different ways.

Game Theoretic (GT) approaches are very closely related to the DT methodologies that are discussed in this work. GT methods can model not only cooperative, but also competitive agent problems, but they are typically concerned with one-shot (static) decision-making (Shoham and Leyton-Brown, 2009). Nevertheless, DT systems can be viewed as sequences of stochastic games, and the synergy between these two formalisms has been recently explored in the context of multiagent planning (Emery-Montemerlo et al., 2005; Hansen et al., 2004; Oliehoek et al., 2008a; Spaan and Melo, 2008).

Another popular approach to the problem of multiagent planning is through Distributed Contraint Optimization (DCOP) techniques (Maheswaran et al., 2004). Although DCOP methods are particularly efficient when taking into account scheduling constraints or interdependencies between the actions of various agents, they do not consider stochasticity, either in the environment itself or in the perception of each agent.

Some of the successful modeling formalisms for single and multi-robot decision-making belong to the theory of Discrete Event Systems (DES) (Cassandras and Lafortune, 1999; Costelha and Lima, 2007; Damas and Lima, 2004; Quottrup et al., 2004). DES models, such as those provided by Finite State Automata (FSAs) or Petri Nets (PNs), are specifically suited to model *asynchronous*, or *event-driven* systems. As we will see in Chapters 5 and 6, multi-robot decision making is often included in that class of systems. However, the qualitative view of DES formalisms is typically not applicable to the problem of obtaining an optimal decision-making plan, *per se*. Rather, it is useful in analyzing the properties of a given plan and in providing theoretical guarantees of its functionality (for example, by verifying if the system will not enter *deadlock* states from which it cannot recover). At most, the methods for *Supervisory Control* of DES theory can be used to drive an event-driven system to a desired set of states. In this work, we will explore the synergy between the qualities of DES formalisms for the high-level description of robot behaviors and tasks, and the theoretical support for planning under uncertainty of DT frameworks.

The *implementation* of multiagent systems has itself been subject to an appreciable amount of research. Various software architectures have been developed specifically for this purpose, for example, JADE (Bellifemine et al., 2007) and INGENIAS (Pavón and Gómez-Sanz, 2003). Such architectures streamline the process of deploying control solutions for multiagent systems, not only by providing a framework that transparently handles inter-agent communication, but also by providing the designer with programming languages, or interfaces, with which to specify the behavior of each agent. However, the type of plans that are produced by DT methods cannot be directly implemented through this schema. DT plans are meant to account for all possible contingencies that may arise during execution, and, as such, for realistically sized problems, they cannot be manually specified.

The reason why we propose to use DT to model multi-robot systems is that, from the available formalisms for multiagent decision-making, it is the most versatile, in that it can model stochasticity in sensing and actuation; it can account for the effects of continuous time; and it has extensive theoretical support both for planning and also for machine learning methods, in multiagent settings. Due to these qualities, DT methods have a strong potential for application in cooperative robotics scenarios.

1.3 Objectives

The work presented in this thesis follows a broad, but well-defined purpose: **to identify and address some of the problems that render existing Decision-Theoretic approaches impractical for teams of robots**.

More concretely, the objectives of this work are:

- The identification and / or development of a cohesive set of DT frameworks and methods that are applicable, in practice, to cooperative robotics problems, and which explicitly address the perceived limitations of currently existing approaches;
- A clear procedure to systematically deploy those methodologies in real robotics applications, and the development of the necessary tools to support that process;
- The demonstration of the proposed methods in scenarios involving real teams of robots.

1.4 Thesis Outline and Contributions

We will now overview the the organization of this document, and provide a brief summary of the contributions that are made in each chapter:

- Chapter 2 introduces the necessary theoretical background on Markov Decisions Processes and associated models, which will act as a basis for the remainder of the work;

- In Chapter 3, we review relevant past work in this field, with particular emphasis on real-world applications of decision-theoretic frameworks. We then present a thorough "walkthrough" of the modeling process of a cooperative robotics task through DT frameworks. We explore the limitations of these frameworks, and identify the main issues to be addressed when considering real teams of robots. While in this context, we introduce and test a case study in cooperative robotics in a realistically simulated environment;

- The issue of efficient communication, which is one of the obstacles involved in the application of DT methods to teams of robots, is studied in Chapter 4. There, we present a method to obtain offline communication policies in partially observable multiagent problems, and show that we are able to significantly reduce the amount of communication that is necessary for the coordination of a team of agents under those conditions;

- In Chapter 5, we approach the problem of multi-robot decision-making as an event-driven process. We study the applicability of the *Generalized Semi-Markov Decision Process* framework to this problem, and highlight its respective advantages and limitations with respect to standard multiagent MDPs. We present the first documented application, to our knowledge, of that framework to the control of a team of real robots;

- In Chapter 6, we focus on the problem of partial observability in event-driven multiagent systems. We introduce the Event-Driven MPOMDP framework as a means of modeling such systems. We show that our proposed framework retains theoretical properties that are essential for its applicability. In particular, we prove that Value Functions for Event-Driven MPOMDPs are Piecewise Linear

Convex (PWLC) functions, which allows them to be solved efficiently through existing POMDP solution algorithms, with minor modifications; and we derive a procedure that allows agents to update their information regarding the environment, in the presence of possible false negative observations. We discuss the practical application of our framework, and present empirical results in simulated environments that showcase its advantages over synchronous alternatives. Furthermore, we also extend the framework to Semi-Markovian domains, resulting in a framework that is specifically suited to model decision-making for teams of robots;

- In Chapter 7, we demonstrate the application of our Event-Driven MPOMDP framework, to solve a problem of task assignment in a real networked robot system. We will discuss the steps involved in the deployment of our methods to that scenario, and describe the software tools that were developed for that purpose, as part of this work;

- Finally, in Chapter 8, we conclude this thesis and present potential directions for future research.

1.5 Publications

The following chapters of this thesis are based on international publications:

Chapter 4:

J. Messias, M. T. J. Spaan, and P. U. Lima. Efficient offline communication policies for factored multiagent POMDPs. In *Proceedings of the 25th Annual Conference on Neural Information Processing Systems (NIPS-11)*, pages 1917–1925, 2011.

Chapter 5:

J. Messias, M. T. J. Spaan, and P. U. Lima. GSMDPs for multi-robot sequential decision-making. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI-13)*, 2013.

Chapter 6:

J. Messias, M. T. J. Spaan, and P. U. Lima. Multiagent POMDPs with asynchronous execution. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-13) - Extended Abstract*, pages 1273-1274, 2013.

J. Messias, M. T. J. Spaan, and P. U. Lima. Asynchronous Execution in Multiagent POMDPs: Reasoning over Partially-Observable Events. *Artificial Intelligence (Special Issue on AI and Robotics)* **Submitted. Pending Review**.

Chapter 2

Background

The work developed in this thesis is based on the theory of Markov Decision Processes (MDPs) (Bellman, 1957a). This well-studied mathematical framework provides a wide array of tools for the formulation, and solution, of decision-making problems in stochastic environments. In their original form, MDPs can model single agent decision-making processes[1], where full knowledge of the state of the associated environment is assumed, and where the system is thought to evolve in discrete time. To address these limitations, various important extensions to the MDP framework have been introduced, which, as a whole, allow a broad class of realistic multiagent problems to be modeled. A given task may be categorized as belonging to one of these model classes depending on its particular requirements, and the different simplifying assumptions which are made on the nature of the problem.

In Figure 2.1, we show the relationship between several relevant DT frameworks that extend MDPs according to each of three fundamental properties that will later be exploited in this work, for our purposes of modeling multi-robot systems: the ability to model time-driven dynamics; partial observability; and multiagent decision-making.

In this chapter, and in order to make the necessary definitions clear, we briefly review the MDP framework, and the associated extensions that are most relevant for this work.

The notation used in this chapter is mostly consistent with that of basic probability theory (Jazwinski, 1970). The sets used in the various modeling frameworks are assumed to be representable as countable unions of singleton elements, unless otherwise

[1] Or *centralized* multiagent decision-making processes, in which agents can communicate perfectly.

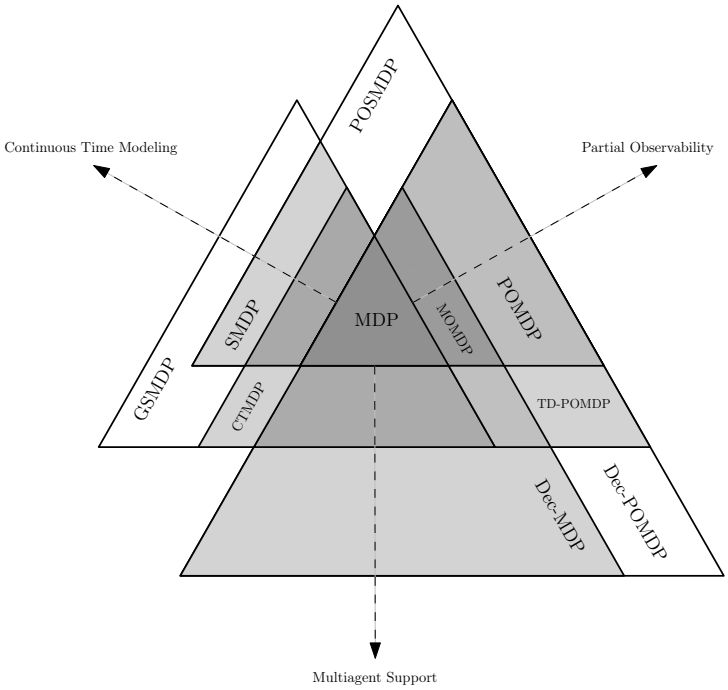


Figure 2.1: An overview of the topological relationships between relevant MDP-based model classes, according to their descriptive capabilities.

stated, and to be equipped with the appropriate discrete topology. For any set X over which a probability measure is required, an adequate probability space $\langle X, \Sigma, \Pr \rangle$ is assumed to exist in which X is the default space of possible outcomes, Σ is the Borel algebra generated by the discrete topology on X, and $\Pr(\mathrm{A})$ is the probability measure of event $A \in \Sigma$. Under this terminology, $\Xi(X)$ refers to the family of all possible probability distributions over X. The power set of X will be represented as $PS(X)$, and its cardinality as $|X|$. The cartesian product of sets X and Y will be shown as $X \times Y$, and as $\prod_{i=1}^{n} X_i$ for the n-ary case.

Additionally, throughout this work, we will use integer subscripts in order to index elements within ordered sets, and also to refer to instantiations of variables at discrete steps of a decision-making process. If both of these indices are simultaneously necessary in context, then the innermost subscript will refer to the element index, while the outermost subscript will refer to its "step" index. That is, if $\mathbf{x} = \langle x_1, x_2 \rangle$, then $x_{1,n}$ will refer to the value of x_1 at step n.

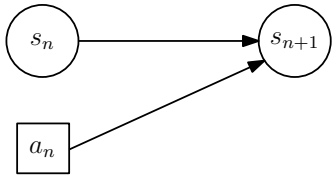


Figure 2.2: A 2-DBN representing an MDP.

2.1 Markov Decision Processes

The MDP framework is particularly suited to modeling problems where the outcome of the actions of an agent is probabilistic, but the state of the process can be known, without uncertainty, at any given time. MDPs were introduced in the context of Operations Research (Bellman, 1957a), and have since found their way into diverse practical single-agent applications (further discussed in Section 3.1). The framework forms a basis for many reinforcement learning techniques (Kaelbling et al., 1996; Sutton and Barto, 1998). Formally, an MDP is defined as follows:

Definition 2.1.1. *A Markov Decision Process (MDP) is a tuple* $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$*, where:*

$\mathcal{S}$ *is the* state space*, a discrete set of possibilities for the state* s *of the process;*

$\mathcal{A}$ *is the* action space*, the set of actions that the agent can perform;*

$T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ *is a* transition function*, such that* $T(s, a, s') = \Pr(s' \mid s, a)$*. It models the transition dynamics associated with the actions of the agent;*

$R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ *represents the "reward" that an agent receives for performing action* a *in state* s*, representing either its utility or cost.*

The reward function is sometimes represented as $R(s, a, s')$, making it depend also on the resulting state s', but an equivalent formulation $R(s, a)$ is always possible.

Given this definition, and in looser terms, a Markov Decision Process represents a system in which, at each discrete step of its execution, an action a is taken by an agent, which causes the system's state to change stochastically from s to s', according to $T(s, a, s')$. This behavior can be graphically represented as a two-slice Dynamic Bayesian Network (2-DBN), which captures the probabilistic dependency between variables at different steps in the process, as shown in Figure 2.2. For an introduction to Dynamic Bayesian Networks, the reader is referred to Murphy (2002). In such a

system, the *Markov property* is held, i.e., the distribution over the future state of the system depends only upon the present state and action, at any given decision step. Equivalently:

$$\Pr(s_{n+1} \mid s_n, a_n, s_{n-1}, a_{n-1}, \ldots, s_0, a_0) = \Pr(s_{n+1}|s_n, a_n) \quad , \tag{2.1}$$

Although less common, there are also variations of this type of model which consider continuous state and/or action spaces (Bertsekas and Shreve, 1978; Puterman, 1994).

2.1.1 Planning and Learning for MDPs

The objective of any decision-making agent in an MDP-based model is to perform a sequence of actions which attempts to maximize the accumulated reward over a given number of steps. This limit on the number of steps is referred to as the *horizon* for the decision-making problem, h. In the case of a standard MDP, this typically amounts to the problem of obtaining a set of decision rules $\delta(s) : \mathcal{S} \rightarrow \mathcal{A}$, or in other words, a map of system states to reward-maximizing actions. The set $\pi(s) = \{\delta_1(s), \delta_2(s), \ldots, \delta_h(s)\}$ of decision rules for every time step constitutes an h-horizon *policy* for the agent. If h is infinite, then $\pi(s) = \delta_i(s) \, \forall i = 1, \ldots, \infty$. Such a policy is said to be *stationary*, or time-invariant.

Policies can also be stochastic, if decision rules instead map to probability distributions over actions, *i.e.* $\delta(s) : \mathcal{S} \rightarrow \Xi(\mathcal{A})$. In this case, an agent draws an action from $\delta(s)$ at each step. Although stochastic policies are of instrumental importance in some domains (such as in *reinforcement learning* settings, which will be revisited later), it can be shown that, within the set of stochastic policies for an MDP, there is always at least one deterministic policy that provides maximum expected reward (Puterman, 1994). Most methods presented in this chapter assume policies to be deterministic, since this is the prevalent case in existing MDP theory, and since it simplifies the theoretical treatment of these models.

There is a wide amount of well-established literature on the problem of obtaining optimal, as well as approximate, policies for MDPs (Feinberg and Shwartz, 2002; Puterman, 1994). The most well-known of these will be outlined in this section, which will provide an insight into the necessary differences, and eventual similarities, between solu-

tion methods for MDPs and those of its associated extensions which will be introduced throughout this chapter.

Since the future states of an MDP cannot be predicted with full certainty, a common measure of the quality of a solution, which most solvers try to optimize, is the *expected* discounted reward over future states:

$$E_\pi \left\{ \sum_{n=0}^{h-1} \gamma^n R(s_n, \delta_n(s_n)) \right\} \quad , \tag{2.2}$$

where $E_\pi \{\cdot\}$ is the expectation operator, considering that the agent is following policy π. The discount factor $\gamma \in [0, 1)$ allows the series to converge in the case of an infinite-horizon problem, and is usually 1 for a finite-horizon case. The above quantity is referred to as the h-horizon *value* of a given state according to π, $V^\pi_{h-1}(s)$. The complete mapping $V^\pi_{h-1} : \mathcal{S} \to \mathbb{R}$ constitutes a *value function* for the problem. If $h = \infty$, we will omit the horizon subscript.

The most ubiquitous solution methods for discrete MDPs belong to the class of Dynamic Programming (DP) algorithms (Bellman, 1957b). This is the case of Value Iteration algorithm, which recursively updates the value function through the *Bellman backup* equation:

$$V^\pi_n(s) = R(s, \delta_n(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \delta_n(s), s') V^\pi_{n+1}(s') \tag{2.3}$$

For infinite-horizon problems, this can be slightly modified to provide the optimal value function $V^*(s)$:

$$V^*(s) = \max_{a \in \mathcal{A}} \left\{ R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V^*(s') \right\} \tag{2.4}$$

Equation (2.4) is guaranteed to converge to a single invariant point, which corresponds to the optimal value function for the problem (Puterman, 1994). For finite horizon problems, only h backups are needed to obtain the optimal value function, but each intermediate result must be stored. If h is infinite, its value function is stationary, so only the final result is necessary. Optimality in this context means that, starting from state s, there exists no sequence of actions of length k with expected reward higher than $V^*_{h-k}(s)$. The optimal policy $\pi^*(s)$ can be easily extracted from the problem's optimal

value function, by simply choosing, in each state, the action associated with the highest expected value:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^*(s') \right\} \tag{2.5}$$

A different approach to this problem, known as policy iteration, instead operates in the space of possible policies. It alternates between the calculation of the expected value associated with a given (possibly non-optimal) policy, and the improvement of that policy by changing its first action, when possible (Puterman, 1994). This method is also guaranteed to converge towards the optimal policy.

Even though the state of the system is available to the agent at run-time, if the system dynamics are known (*i.e.* the matrix T), then the reward $R(s,a)$ received for a given state-action pair does not need to be observed *during execution*, since there is enough information in these models to calculate the expected reward for a given policy. The solution to the MDP is then said to be obtained "offline", symbolizing that policy execution is not required in order to solve the decision-making problem. This is also ubiquitously referred to as the *planning* problem for decision-theoretic models.

In contrast, an agent can also improve its policy during execution, if the collected reward as it interacts with the environment is also observed at run-time, or "online", either immediately associated with a given state and action, or after the episodic execution of a given policy. In the associated literature, this is known as the *learning* problem for decision-theoretic agents. A common way of addressing this problem is through the use of temporal difference learning (or TD-learning) methods and its closely related adaptations, which estimate the value function and improve the associated policy accordingly, using the immediately collected reward as an input. The most basic form of temporal difference learning is implemented through the so-called TD(0) equation:

$$V^\pi(s) = (1 - \alpha_k)\, V^\pi(s) + \alpha_k \left(R(s,a) + \gamma V^\pi(s') \right) \quad , \tag{2.6}$$

where $\alpha_k \in (0,1)$ is the learning rate or step-size after $k \in \mathbb{N}_0$ iterations, which in practice acts an exponential smoothing over estimated values, and π is a given *behavior* policy. These methods rely on the property that $R(s,a) + \gamma V^\pi(s')$ approaches $V^\pi(s)$ as the value function converges towards its invariant point. They are in fact instantiations

of a more general *stochastic approximation* algorithm (Robbins and Monro, 1951). For these methods to *learn* V^π, sufficient exploration of state-action pairs is assumed (i.e. each pair must be visited a number of times). To this end, reinforcement learning methods enforce some level of stochasticity in the resulting agent policy (*i.e.* π becomes a stochastic policy), which leads the agent to select random actions if the state space must be further explored. To ensure convergence, the learning rate α_k should also satisfy that $\sum_{k=0}^{\infty} \alpha_k = \infty$ and $\sum_{k=0}^{\infty} \alpha_k^2 < \infty$.

TD-learning methods can be classified as being *on-policy*, if they learn the value of the behavior policy (as above), or as *off-policy* methods, in which the estimated value converges, instead, to that of the optimal policy V^*. A very common off-policy reinforcement learning method based on temporal differences is Q-learning (Watkins, 1989), which updates the optimal value of state-action pairs (s, a) as follows:

$$Q(s,a) = (1-\alpha)\, Q(s,a) + \alpha \left(R(s,a) + \gamma \max_{a' \in \mathcal{A}} Q(s',a') \right) \quad . \tag{2.7}$$

Common on-policy algorithms include TD(λ), SARSA, and actor-critic methods (Kaelbling et al., 1996; Sutton, 1984; Sutton and Barto, 1998).

A note on terminology: throughout this work, we will often focus on the problem of modeling real systems through MDP-based frameworks, but we will intentionally leave our considerations unbound to any specific solution methodology (*i.e.* planning or learning) whenever that distinction is not relevant. To this end, and to be able to characterize a given DT model with respect to the practicality of its solution, we define the term *operational complexity* to mean the computational complexity of a planning algorithm, or the sample complexity of a learning algorithm, over a given MDP-based model.

2.2 Extensions for Continuous-Time Problems

The concept of time in an MDP is abstract from the perspective of the planning problem. In an MDP, a decision is taken whenever an action is considered to have finished, or equivalently when a state transition is known to have occurred. The "time" index in much of the associated theory more accurately represents the number of sequential decision "epochs", or steps, which have elapsed, or the remaining number of such decisions left to take. It should then be made clear that decisions are not actually bound

to any specific time instant, or interval, associated with the execution of the underlying system, and rather with state transitions which are triggered by actions.

In many real-world scenarios, the physical time that elapses in the system is relevant to the decision-making process. This is the case, for example, of queueing and maintenance systems (Puterman, 1994), control of epidemic processes and population modeling (Guo and Hernández-Lerma, 2009), dynamic power management systems (Rong and Pedram, 2003), and also in robotics problems (Mahadevan and Khaleeli, 1999), which are of special interest to this work.

2.2.1 Semi-Markov Decision Processes

A Semi-Markov Decision Process (SMDP) (de Cani, 1964; Howard, 1963; Jewell, 1963) extends the basic MDP framework by allowing the underlying system dynamics to also depend on the time since the last decision was carried out. Since the model dynamics will then require memory of the instant in wich a particular state was entered, the system does not strictly maintain the Markov property within each state. However, after a state transition takes place, all memory is discarded, and the future state of the system can still be predicted based only on present information. This is why such a system is said to be *Semi*-Markov.

In an SMDP, decisions are still taken in sequential steps, but the particular instant at which each step occur is also relevant to the decision-making process, and so its stochasticity is modeled accordingly. A SMDP can describe, for every state and action, and possibly depending on the resulting future state, a probability density function over the specific time instant of the next decision.

State transitions are a necessary, but not sufficient, condition for a decision to be taken by an SMDP agent (Puterman, 1994). The system state can in fact change multiple times between decisions. This work will restrict its focus to SMDPs in which decisions are taken upon the occurrence of any state transition, which will allow a later correspondence between decision steps and *events*.

For SMDPs in these conditions, and considering infinite-horizon problems, the expected discounted reward becomes:

$$V^{\pi}(s_0) = E_{\pi}\left\{\sum_{n=0}^{\infty} e^{-\lambda \mathcal{T}_n}\left(R(s_n, \delta_n(s_n)) + \int_{\mathcal{T}_n}^{\mathcal{T}_{n+1}} C(s_n, \delta(s_n)) e^{-\lambda(t-\mathcal{T}_n)} dt\right)\right\} \quad , \quad (2.8)$$

where $\mathcal{T}_n \in \mathbb{R}_0^+$ is the instant at which step n occurs, with respect to the time at which the first action was taken (naturally $\mathcal{T}_0 = 0$); $C(s_n, a_n)$ is a *cumulative* reward rate that is accrued throughout each decision step; and the discount factor, λ, which is here limited to the interval $(0, \infty)$, serves the same purpose as γ in a discounted MDP.

The formal definition of an SMDP is as follows:

Definition 2.2.1.
A Semi-Markov Decision Process (SMDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, T, \mathcal{F}, R, C \rangle$*, where:*

$\mathcal{S}, \mathcal{A}, T, R$ *have the same definition as in a Markov Decision Process (see Definition 2.1.1);*

$\mathcal{F} = \left\{ f^a_{s,s'} : \mathbb{R}_0^+ \to \mathbb{R} \mid \int_\alpha^\beta f^a_{s,s'}(t)dt \;=\; \Pr(\alpha \leq t \leq \beta \mid s', s, a)\,,\ \forall s, s' \in \mathcal{S},\ a \in \mathcal{A} \right\}$ *is the* time model. *Each* $f^a_{s,s'}$ *is a probability density function over the remaining time until the next decision step, given that transition* $\langle s, s' \rangle$ *occurs, and action* a *is applied*[1]*.* $F^a_{s,s'}(\tau) \;=\; \Pr(t \leq \tau \mid s', s, a)$ *represents the respective cumulative distribution function for each of these transition-action tuples. Each* $f^a_{s,s'}$ *is assumed to admit a Laplace transform;*

$C : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ *is the cumulative reward rate which, in addition to the* instantaneous *reward model* R*, allows the modeling of an utility or cost associated with the sojourn time at* s *while executing* a*.*

With this definition, the total reward for a state-action pair in an SMDP is given by:

$$U(s,a) = R(s,a) + E_\tau \left\{ \int_0^\tau e^{-\lambda t} C(s,a) dt \right\} \tag{2.9}$$

$$= R(s,a) + \sum_{s' \in \mathcal{S}} \int_0^\infty p(\tau, s' \mid s, a) \left(\int_0^\tau e^{-\lambda t} C(s,a) dt \right) d\tau \quad , \tag{2.10}$$

where $p(\tau, s' \mid s, a)$ represents the mixed continuous-discrete joint distribution over resulting states and decision instants. The last term in (2.9) corresponds to the expectation of the reward accumulated through C, over the time τ between decision instants, and discounted over time by λ. Through a simple application of Bayes' Theorem:

$$p(\tau, s' \mid s, a) = p(\tau \mid s', s, a) \Pr(s' \mid s, a) = f^a_{s,s'}(\tau) T(s, a, s') \tag{2.11}$$

[1] Equivalently, this is the time that is needed for the system to move between s and s' while executing a. Note that this a relative measure between successive steps — it is *not* the total time elapsed in the system. Moreover, each $f^a_{s,s'}$ is assumed to be invariant with respect to decision steps.

This implies that, in an SMDP, $T(s, a, s')$ represents the state transition probabilities only at decision instants, and is not influenced by time. It can be thought of as describing an "embedded" discrete-time MDP at these instants (Puterman, 1994). The total reward can then be simplified as:

$$U(s,a) = R(s,a) + \sum_{s' \in \mathcal{S}} T(s,a,s') \int_0^\infty \left(f^a_{s,s'}(\tau) \int_0^\tau e^{-\lambda t} C(s,a) dt \right) d\tau \tag{2.12}$$

$$= R(s,a) + \frac{1}{\lambda} C(s,a) \sum_{s' \in \mathcal{S}} T(s,a,s') \left(1 - \int_0^\infty f^a_{s,s'}(\tau) e^{-\lambda \tau} d\tau \right) \tag{2.13}$$

$$= R(s,a) + \frac{1}{\lambda} C(s,a) \sum_{s' \in \mathcal{S}} T(s,a,s') \left(1 - \mathcal{L}\left\{ f^a_{s,s'}(\tau) \right\} \right) \quad , \tag{2.14}$$

where $\mathcal{L}\{f(t)\}$ is the Laplace transform of $f(t)$, as used in (de Cani, 1964; Jewell, 1963)[1].

The value of executing a stationary policy in an SMDP is then, for each state:

$$V^\pi(s) = U(s, \pi(s)) + \sum_{s' \in \mathcal{S}} V^\pi(s') \int_0^\infty p(t, s' \mid s, a) e^{-\lambda t} dt \tag{2.15}$$

$$= U(s, \pi(s)) + \sum_{s' \in \mathcal{S}} T(s,a,s') V^\pi(s') \int_0^\infty f^a_{s,s'}(t) e^{-\lambda t} dt \tag{2.16}$$

$$= U(s, \pi(s)) + \sum_{s' \in \mathcal{S}} \mathcal{L}\{f^a_{s,s'}(t)\} T(s,a,s') V^\pi(s') \tag{2.17}$$

Equation (2.15) allows an SMDP to be viewed as an equivalent discrete-time MDP with state-action dependent discount rates $\gamma(s,a) = \mathcal{L}\{f^a_{s,s'}(t)\}$ (Bradtke and Duff, 1995; Puterman, 1994). This, in turn, forms a very positive practical result for SMDPs, since virtually all solution algorithms for discrete MDPs can also be applied to this class of models. Well-established reinforcement learning algorithms, such as TD(λ) and Q-learning, have been shown to work with SMDPs in (Bradtke and Duff, 1995). Note that, even though our ultimate goal, when planning over an SMDP, is to reduce it to a discrete MDP that is equivalent for decision-making purposes, it is not possible to describe that equivalent MDP directly, in the general case, sidestepping the need to explicitly model time. The values of $\gamma(s,a)$ are not known *a priori*, and they cannot

[1] We note that the Laplace-Stieltjes transform of the cumulative distribution function $F^a_{s,s'}$ is most commonly used for this simplification, due to its relationship with probability generator functions in continuous-time Markov chains. We will not make use of this relationship, and opt to use the standard Laplace transform for clarity.

be arbitrarily defined. Modeling the temporal distributions $f^a_{s,s'}(t)$, according to the actual underlying physical system, is a fundamental step of this process. It is through the knowledge of these distributions that an SMDP agent can evaluate sequences of actions not only with respect to their total reward, but also considering the time that is needed for their execution.

2.2.2 Continuous-Time Markov Decision Processes

Another class of models closely related to SMDPs is that of Continuous-Time Markov Decision Processes (CTMDPs). CTMDPs are defined by the same constructs as a SMDP, and can in fact be viewed as special cases of the latter class, in that they assume that all distributions described in the time model $f^a_{s,s'}$ (refer to Definition 2.2.1) are exponential in nature. Due to the memoryless property of these distributions, in this case, the system remains fully Markovian at any given point in time. In contrast, SMDPs are only Markovian at decision instants.

CTMDPs are suitable, for example, for arrival-rate problems in queueing systems, and population modeling with limited resources (Guo and Hernández-Lerma, 2009), in which all events are known to be generated by underlying Poisson processes. Any CTMDP can be transformed into an equivalent discrete-time MDP through a process of *uniformization* (Puterman, 1994), which first provides an equivalent, oversampled continuous time model with equal decay rates for every transition, and then calculates an appropriate discount factor that accounts for the time in each transition. As in the case of SMDPs, this allows CTMDPs to be solved through virtually any MDP solution algorithm.

2.2.3 Generalized Semi-Markov Decision Processes

One limitation of SMDPs is that, even though the time between decision steps may be governed by non-Markovian distributions, upon reaching a new state, all information regarding the past execution of the system is assumed to be unnecessary in order to predict its future states. The lower limit of the integral over time in (2.15) is always 0; otherwise, the process would not retain the Markovian property at decision instants. In some systems, however, it may be necessary to maintain a non-Markovian temporal model *across* decision steps, which in turn may require knowledge of the past execution of the system. The following example illustrates this problem:

Example 2.2.1. The problem of persistently enabled events: *Suppose that a robot operates in an environment with two rooms, represented by states L and R, and is able to move from one room to the other with fixed probability, according to a single action a. The robot can run out of battery, at which point it will enter the absorbing state N. The state space and possible transitions of this problem are shown in Figure 2.3. In that context, we will here refer to "events" as labels for the state transitions $\langle s, s' \rangle$. Three events can happen in this system: staying in the same room, e_{stay}; moving to the other room, e_{move}; and running out of battery, $e_{battery}$.*

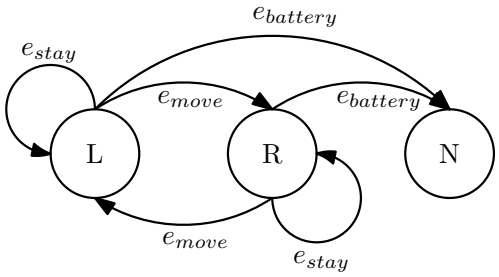


Figure 2.3: A simple scenario which showcases the limitations of SMDPs, and illustrates our definition of "events". Events, which are abstractions to transitions between states, are represented by arrows and labeled $\{e_{stay}, e_{move}, e_{battery}\}$.

Assume that the battery life of the robot is a random variable T_B governed by an arbitrary positive probability distribution, $g(t_B)$. For example, Weibull distributions are commonly used for this purpose. In an SMDP model of this system, this would mean that $f^a_{\cdot,N} = g(t_B)$. Note that T_B is measured with respect to a fixed clock, which is external to the decision-making process. In other words, before each decision, the time until the battery of the robot runs out is decremented by the sojourn time at the previous state.

Suppose now that the robot experiences, with a fixed period Δ, either e_{stay} or e_{move}. Naturally, the robot will function while $T_B - k\Delta > 0$, for $k \in \mathbb{N}$. At each step, the expected remaining battery life is $E\{T_B \,|\, T_B > k\Delta\} = \int_{k\Delta}^{\infty} p(t_B \,|\, t_B > k\Delta) t_B dt_B = \int_{k\Delta}^{\infty} h(t_B) t_B dt_B$ where

$$h(\tau) = \begin{cases} \frac{g(\tau)}{\int_{k\Delta}^{\infty} g(\tau) d\tau} & \text{if} \tau > k\Delta \\ 0 & \text{otherwise} \end{cases}$$

The flaw of this model is that, at each iteration of (2.15)*, all of the information regarding the past of an SMDP is lost, so it is impossible to describe $p(t_B \,|\, t_B > k\Delta)$ at step k, unless $g(t_B)$ is exponential (*i.e. *the process is actually a CTMDP), and therefore*

memoryless; or unless k is included in the system state. That is to say, the perceived probability of running out of battery in states L, R is the same regardless of how many decision steps have elapsed. This would erroneously represent that the expected lifetime of the battery of robot would remain constant throughout execution. The event $e_{battery}$ is said to be persistently enabled*, in that it remains enabled despite the triggering of other events.*

In practice, this problem of modeling "persistently enabled events" appears in the context of modeling multiagent system, or systems with otherwise *asynchronous* dynamics. We will revisit this problem in Chapter 5, in the context of multi-robot systems.

Before proceeding, we will present the formal definition for what constitutes an "event", which will be used throughout this work:

Definition 2.2.2. *Let $\mathcal{E}$ be a countable set. $\mathcal{E}$ is a set of* events *over a state space $\mathcal{S}$ if there is a surjective mapping $\Phi : \mathcal{S} \times \mathcal{S} \to \mathcal{E}$ such that, for all pairs of state transitions $\langle u, u' \rangle, \langle v, v' \rangle \in \mathcal{S} \times \mathcal{S}$:*

- *If $\Phi(u, u') = \Phi(v, v')$, then $T(u, a, u') = T(v, a, v')$ and $f^a_{u,u'} = f^a_{v,v'}$, $\forall a \in \mathcal{A}$ (events abstract transitions with the same stochastic properties);*
- *$\Phi(u, u') \neq \Phi(v, v')$ if $u' \neq v'$ (events univocally identify transitions from each state).*

Furthermore, we will formally define the *enabled*, *disabled*, and *persistently enabled* properties for events:

Definition 2.2.3. *Let $\mathcal{E}$ be a set of events over a state space of an SMDP $\langle \mathcal{S}, \mathcal{A}, T, \mathcal{F}, R, C \rangle$. An event $e \in \mathcal{E}$ is said to be* enabled *at $\langle s, a \rangle$ if there exists $s' \in \mathcal{S}$ such that $e = \Phi(s, s')$ and $T(s, a, s') > 0$.*

The set of all enabled events in these conditions will be represented as $E(s, a)$. Any event $e' \in \mathcal{E}$ such that $e' \notin E(s, a)$ is disabled *at $\langle s, a \rangle$.*

Definition 2.2.4. *Let $\mathcal{E}$ be a set of events over a state space of an SMDP $\langle \mathcal{S}, \mathcal{A}, T, \mathcal{F}, R, C \rangle$. An event $e \in \mathcal{E}$ is* persistently enabled *from step n to $n + 1$ if $e \in E(s_n, a_n)$ and $e \in E(s_{n+1}, a_{n+1})$, but $\Phi(s_n, s_{n+1}) \neq e$ (e did not trigger at step n).*

We are now in a position to define Generalized Semi-Markov Decision Processes (GSMDPs), which were first introduced by Younes and Simmons (2004), in an effort to address the aforementioned limitations of SMDPs. GSMDPs are an extension of the earlier Generalized Semi-Markov Process (GSMP) formalism, introduced in Glynn (1989)

to analyze Discrete Event Systems (DES). This establishes a close parallelism between SMDPs and areas of the theory of Discrete-Event Systems (Cassandras and Lafortune, 1999) which explicitly model time, such as Stochastic Timed Automata (STA) (Neto, 2010) and Generalized Stochastic Petri Nets (GSPNs) (Murata, 1989). In GSMDPs (and GSMPs), the occurrence of an event in the system does not necessarily erase all information of past events, actions, and triggering times. This makes it possible to infer the time which has elapsed since each event became enabled, even if different events have been meanwhile triggered. This information, in turn, allows for the accommodation of general non-Markovian temporal distributions over state transitions, across multiple decision instants.

Definition 2.2.5. *A Generalized Semi-Markov Decision Process (GSMDP) is a tuple* $\langle \mathcal{S}, \mathcal{A}, T, \mathcal{F}, R, C, \mathcal{E} \rangle$ *where:*

$\mathcal{S}, \mathcal{A}, R, C$ *have the same definition as in a Semi-Markov Decision Process (see Definition 2.2.1);*

$\mathcal{E}$ *is a set of events over* $\mathcal{S}$*;*

$T : \mathcal{S} \times \mathcal{A} \times \mathcal{E} \to [0, 1]$ *represents event probabilities,* $T(s, a, e) = \Pr(e \,|\, s, a)$*, for* $s \in \mathcal{S}, a \in \mathcal{A}, e \in \mathcal{E}$*;*

$\mathcal{F}$ *holds the same meaning as in an SMDP, but it is instead mapped by events, and* **it can depend on the execution history of the system since an event became enabled**. *Formally, let* $t_k \in \mathbb{R}_0^+$ *represent the elapsed time between steps* $k-1$ *and* k*, for* $k > 0$ *(with* $t_0 = 0$*); then,* $\vec{\epsilon}_{0:n} = \{\langle s_0, a_0, t_0 \rangle, \langle s_1, a_1, t_1 \rangle, \ldots \langle s_{n-1}, a_{n-1}, t_{n-1} \rangle\}$ *represents the* execution history *of the system between decision steps* 0 *and* n*. If* e *was enabled at step* $m < n$*, and remained enabled for all steps* $\{m+1 \ldots, n-1\}$*, then* $f_e^a(t \,|\, \vec{\epsilon}_{m:n})$ *is such that* $\int_\alpha^\beta f_e^a(t \,|\, \vec{\epsilon}_{m:n}) dt = \Pr(\alpha \leq t \leq \beta \,|\, a, e, \vec{\epsilon}_{m:n})$*. In other words,* $f_e^a(t \,|\, \vec{\epsilon}_{m:n})$ *represents the probability density over the remaining time until the next decision step, given that the next event that will trigger is* e*, and that it has been enabled since step* m*.*

It should be noted that our definition of "events", and consequently, of GSMDPs, is slightly different than that which is used by Younes and Simmons (2004). In Chapter 5, we will compare both definitions, and show that GSMDP models can be converted between them; consequently, the validity of the methodologies which we will discuss in this work, with respect to the GSMDP framework, is not affected by this dissimilarity.

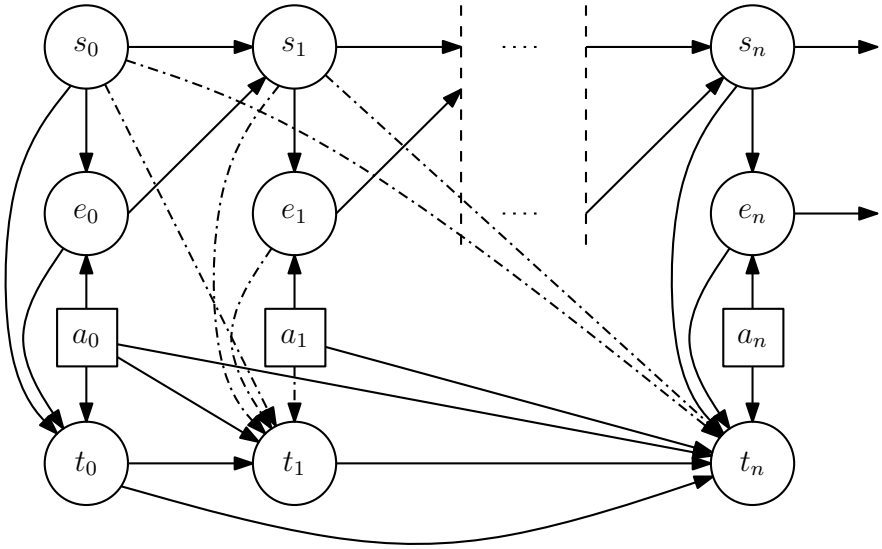


Figure 2.4: A DBN representing a GSMDP. Since the system is non-Markovian, it cannot be described through a 2-DBN. Some connections are dash-dotted for better visibility.

It follows from Definition 2.2.5 that, in a GSMDP, the actions of the agent may also depend on execution histories, instead of simply on states. Let t_i represent, as above, the relative time between decisions $i-1$ and i. Then, $\mathcal{T}_n = \sum_{i=0}^{n} t_i$ represents the total time elapsed in the system until step n. Ultimately, the goal of the planning problem is to optimize the value function over the complete execution history:

$$V^{\pi}(\epsilon) = \sum_{n=0}^{\infty} e^{-\lambda \mathcal{T}_n} \left(R(s_n, \pi(\vec{\epsilon}_{0:n})) + \int_0^{t_n} e^{-\lambda t} C(s_n, \pi(\vec{\epsilon}_{0:n}))dt \right) \tag{2.18}$$

The maximization of (2.18) is a difficult problem, since the underlying dynamics are non-Markovian, and typical DP approaches cannot be directly applied in such a case. Younes and Simmons (2004) proposed a method to solve a GSMDP by first approximating each non-exponential component of $\mathcal{F}$ with a *Phase-Type* distribution. Phase-Type distributions describe the time that is needed for a continuous-time Markov chain (that is, a CTMDP without decision inputs) to evolve from an initial state to an absorbing state. It is well-known that *Phase-Type* distributions can be used to approximate any general positive probability distribution. Using this technique implies an expansion of the state space of the problem (via the introduction of *virtual* Makov chains in the system to emulate a given non-Markovian transition), but allows the resulting approximate model to be representable by a CTMDP. Consequently, and through uni-

formization, typical discrete-time MDP methods can be applied. We will review this method in greater detail in Chapter 5, in the context of multi-robot decision-making. Alternatively, Rachelson et al. (2008) have recently proposed an approximate reinforcement learning algorithm for GSMDPs based on policy iteration.

2.3 Extensions for Partially Observable Domains

In many practical scenarios to which MDPs can be applied, it is not unrealistic to assume that the state of the environment can indeed be known by the agent. Such is the case of many scenarios in Operations Research, where the state can comprise the number of people in a queue, or the phase of a manufacturing process, for example; and also for virtual scenarios (Little, 1955; Stidham and Weber, 1993; Tesauro, 1992). However, whenever a physical agent is required to actively sense its environment, through whichever means are available to it, this assumption may quickly be violated, since typically only an *estimate* of the actual state of the system can be obtained. In other scenarios, even if the agent can sense its surroundings with great accuracy, it may still only be observing *part* of the complete state of the system. The term *partial observability* refers to both of these situations, and establishes a contrast with the *fully observable* scenarios so far described, in which the state is directly accessible at any time. The field of Robotics provides evident examples of scenarios with partially observable environments (Mahadevan and Khaleeli, 1999; Miller et al., 2009; Nourbakhsh et al., 1995; Spaan and Vlassis, 2004; Sridharan et al., 2010). POMDPs can also be viewed as a controllable form of Hidden Markov Models (HMMs), which are widely used in the field of Bayesian Inference (Murphy, 2002).

2.3.1 Partially Observable Markov Decision Processes

The framework of Partially Observable Markov Decision Processes (POMDPs) constitutes a conceptually straightforward extension of MDPs to situations in which the agent is not only allowed to *act* upon the environment, but is also able to *observe* it at each step of the process. These observations are generated by the environment itself, through an associated stochastic model, as a response to a given state-action pair.

Definition 2.3.1. *A Partially Observable Markov Decision Process (POMDP) is a tuple* $\langle \mathcal{S}, \mathcal{A}, T, \mathcal{O}, O, R \rangle$*, where:*

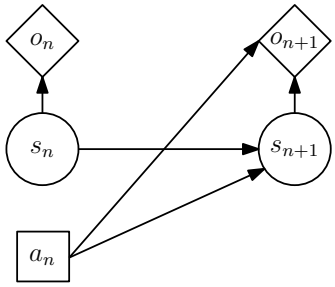


Figure 2.5: A 2-DBN representing a POMDP.

$\mathcal{S}, \mathcal{A}, T, R$ have the same definition as in a Markov Decision Process (see Definition 2.1.1);

- *$\mathcal{O}$ is a set of observations o which can be generated by the environment;*
- *$O : \mathcal{O} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ is the* observation function*, such that $O(o, s', a) = \Pr(o \mid s', a)$ for $o \in \mathcal{O}$, $s' \in \mathcal{S}$, $a \in \mathcal{A}$;*

The central difference in the theoretical treatment of POMDPs, with respect to MDPs, is that, since the state of the system is no longer directly accessible, POMDPs must instead operate over elements of $\Xi(\mathcal{S})$. These elements are often referred to as *belief states*, and represented as $b \in \mathcal{B}$, where $\mathcal{B} \equiv \Xi(\mathcal{S})$ and is likewise termed as *belief space*. If the true state of the system at step n is x, then $b_n(s) = \Pr(s = x \mid b_0, a_0, o_0, \ldots, a_n, o_n)$, where b_0 is an initial belief state.

2.3.1.1 Planning Algorithms for POMDPs

As described through (2.3), a value function of an MDP can be compactly represented as a finite-dimensional quantity (specifically, a vector in $\mathbb{R}^{|\mathcal{S}|}$ representing the values of the $|\mathcal{S}|$ states of the problem). In the POMDP case, since there are infinitely many elements in $\mathcal{B}$, this is no longer possible, and it is therefore necessary to obtain an alternative representation for the expected total reward associated with any policy $\pi : \mathcal{B} \to \mathcal{A}$, over a decision-making horizon, h. This expectation, taken over the h steps of the process, takes the form:

$$E_\pi \left\{ \sum_{n=0}^{h-1} \gamma^n \sum_{s \in \mathcal{S}} b_n(s) R(s, \delta_n(b_n)) \right\} \quad . \tag{2.19}$$

In turn, the Bellman equation (2.3) applied to finite-horizon POMDPs becomes:

$$V_n^*(b) = \max_{a \in \mathcal{A}} \left\{ \sum_{s \in \mathcal{S}} b(s) \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \sum_{o \in \mathcal{O}} T(s,a,s') O(o,s',a) V_{n+1}^*(b^{a,o}) \right) \right\} , \tag{2.20}$$

where $b^{a,o}$ is the *updated belief*, such that, $\forall s \in \mathcal{S}$:

$$b^{a,o}(s') = \frac{O(o,s,a) \sum\limits_{s' \in \mathcal{S}} b(s) T(s,a,s')}{\sum\limits_{u' \in \mathcal{S}} O(o,u',a) \sum\limits_{u \in \mathcal{S}} b(u) T(u,a,u')} \quad . \tag{2.21}$$

This function intuitively describes the process of incorporating information from the actions and observations of an agent into its state estimate.

Equation (2.20) enables the application of the Value Iteration algorithm to POMDP settings. It can provide an optimal (or in the infinite-horizon case, an approximately optimal) policy for the agent for any given belief state, but it does not address the problem of how to actually represent the respective solution as a finite-sized structure, so that it can be accessed at run-time. It is well known, however, that the function V_n^* in (2.20) is a piecewise-linear convex function (Cheng, 1988; Sondik, 1971), and, as such, admits a compact representation as:

$$V_n^*(b) = \max_{\alpha \in \Gamma_n} \left\{ \sum_{s \in \mathcal{S}} b(s) \alpha(s) \right\} \quad , \tag{2.22}$$

where each α is a $|\mathcal{S}|$-dimensional vector, often simply referred to as an α-vector, and Γ_n is the set of all such vectors at step n. Furthermore, a mapping $\nu : \Gamma_n \to \mathcal{A}$ exists which associates each α-vector with an action at that step. Intuitively, an α-vector encodes, as a linear function of the belief state of the system, the expected reward of taking action $\nu(\alpha)$ and then following the optimal policy for the remaining steps. Using representation (2.22) allows a value function to be represented at each step by simply storing the sets Γ_n. The respective policy can easily be extracted:

$$\delta_n(b) = \nu \left(\arg\max_{\alpha \in \Gamma_n} \left\{ \sum_{s \in \mathcal{S}} b(s) \alpha(s) \right\} \right) \tag{2.23}$$

An important property of the value function when represented in this way, which will later be used in this work (see Chapter 4), is that, due to its convexity, each α-vector

defines a convex region of $\mathcal{B}$ over which it is maximal. For a finite number of alpha vectors (this number is bounded above by $|\mathcal{A}|^{\frac{|O|^{h+1}-1}{|O|-1}}$(Cassandra, 1998a)), all regions generated by Γ_n are convex polytopes. The region associated with a given $\alpha \in \Gamma_n$ is defined by the following constraints:

$$\sum_{s \in \mathcal{S}} b(s) \left(\alpha'(s) - \alpha(s)\right) \leq 0 \quad , \forall \alpha' \in \Gamma_n \,|\, \alpha' \neq \alpha \tag{2.24}$$

$$\sum_{s \in \mathcal{S}} b(s) = 1 \tag{2.25}$$

$$b(s) \geq 0 \quad , \forall s \in \mathcal{S} \tag{2.26}$$

These regions are referred to as the *linear support* of each $\alpha \in \Gamma_n$, or $\mathbb{L}(\alpha)$ (Cheng, 1988). This concept is exemplified in Figure 2.6. The linear support of a given α is non-empty if and only if there is some point $b \in \mathcal{B}$ for which $\delta_n(b) \equiv \nu(\alpha)$, according to (2.23)[1]. All vectors which do not verify this condition may be discarded, since they do not contribute any information towards the policy of the agent (Cassandra, 1998a).

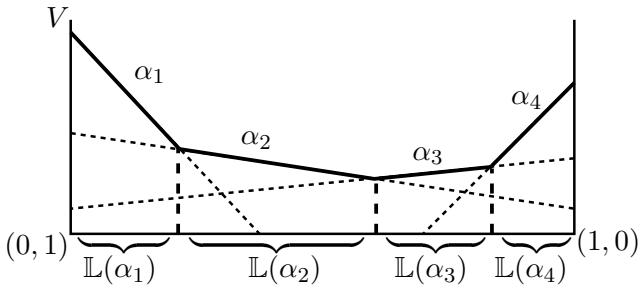


Figure 2.6: The linear supports of a value function as a convex partition of $\mathcal{B}$.

Attempting to optimally solve a finite-horizon POMDP is a PSPACE-hard problem (Papadimitriou and Tsitsiklis, 1987). Classical POMDP algorithms attempt to determine the minimal set of α-vectors which make up the optimal value function, either by enumerating every such possible vector and pruning the result (Monahan, 1982), or by exploring the structure of the induced linear supports over the belief space (Cassandra, 1998a; Cheng, 1988). Many state-of-the-art POMDP solution algorithms forgo strict optimality in favor of scalability. A particularly successful class of algorithms is that of *point-based* solvers. These solvers optimize expected reward over a finite set of

[1]Note that the extreme case in which the linear support for $\alpha \in \Gamma_n$ is a singleton set is irrelevant, since this implies that there exists $\alpha' \in \Gamma_n \backslash \alpha$ such that $b \cdot \alpha = b \cdot \alpha'$ for the only point $b \in \mathbb{L}(\alpha)$

samples taken from the belief space. This approach takes advantage of the fact that, from a given initial belief, there is only a finite set of possible (or "reachable") belief states, given all possible observations, and the actions selected by the agent at those points. Therefore, calculating the optimal value function over the complete belief space often becomes unnecessary. The set of maximal α-vectors, over all belief state samples, constitutes an approximation to the optimal value function. Examples of this type of algorithm include PBVI (Pineau et al., 2003), SARSOP (Kurniawati et al., 2008) and PERSEUS (Spaan and Vlassis, 2005).

2.3.1.2 Continuous-Domain POMDPs

When modeling physical problems with naturally continuous variables, one possible approach is to consider continuous DT models for the environment from the onset, sidestepping the need to perform any discretization of those variables.

This poses a complex theoretical obstacle, however, since belief distributions become infinite-dimensional structures. In order to maintain analytical tractability, a possible solution is to represent underlying belief distributions in a parametric form, either by assuming system linearity (Porta et al., 2006), by approximating non-linear models through switching-mode linear components (Brunskill et al., 2010), or by approximating the true distribution within a family of known closed-form alternatives (Zhou et al., 2008). Another popular approach is to use particle-filter techniques to approximate non-parametric distributions (Porta et al., 2006; Thrun, 2000). It should also be noted that most of these solutions focus on continuous-state POMDPs, and few can also handle continuous action and observation spaces (Porta et al., 2006).

2.3.2 Partially Observable Semi-Markov Decision Processes

The considerations which have been previously made regarding the limitations of the MDP framework when dealing with time-dependent problems (Section 2.2), are also applicable to POMDPs. Likewise, the extensions which have been formulated in order to overcome these shortcomings, for the fully observable case (CTMDPs, SMDPs and GSMDPs), provide a basis for the definition of analogous counterparts for partially observable domains. One such readily defined extension, introduced in (White, 1976), and used in the context of mobile robot navigation (Mahadevan and Khaleeli, 1999), is the class of Partially Observable Semi-Markov Decision Processes (POSMDPs). For

concreteness, a POSMDP incorporates a time model $\mathcal{F}$, and cumulative reward structure C (see Definition 2.2.1) into a commonly defined POMDP (see Definition 2.3.1).

A POSMDP can be reduced to its *embedded* POMDP if it is controlled only at transition instants (White, 1976), as described in Section 2.2.1 for the SMDP case. This makes synchronous POSMDPs solvable through ordinary POMDP algorithms (Mahadevan and Khaleeli, 1999), and implies that all considerations made with respect to partial observability are equivalent between the two frameworks. However, POSMDPs are also unable to model event concurrency, making them unsuitable for general multiagent problems, in their original form.

2.4 Extensions for Multiagent Decision-Making Problems

While the various frameworks so far described already allow a broad class of realistic problems to be modeled, all of them assume that there is a single agent, or a system which can be viewed as a single agent, involved in the decision-making process. However, there is a clear interest in being able to apply the aforementioned theory to decision-making problems in multiagent systems. Multiagent DT methods are of particular relevance to the field of Cooperative Robotics, which provides the main case-studies for this work, and in which various practical applications of MDP-based models have been reported (Capitán et al., 2013; Matarić, 1997; Miller et al., 2009; Oliehoek and Visser, 2006). Other applications of multiagent decision-making include sensor networks (Nair et al., 2005), and network routing problems in distributed systems (Pajarinen and Peltonen, 2011; Tao et al., 2001).

Adding multiple agents adds another layer of complexity to the decision-making problem, since, in the general case, agents will then have to reason not only over their own actions and observations, but also over those of other agents, taking into account possible interdependencies.

2.4.1 Decentralized Partially Observable Markov Decision Processes

Decentralized POMDPs (Dec-POMDPs) constitute the most general MDP-based framework with respect to the scope of multiagent scenarios it can model (refer to Figure 2.1). Dec-POMDPs were introduced in Bernstein et al. (2002) as a straightforward extension

of POMDPs to scenarios where more than one agent may be acting upon the environment, and observing some of its features with uncertainty.

In Dec-POMDPs and their respective multiagent-oriented subclasses, it is often necessary to index model components (*e.g.* actions, observations) with respect to an individual agent, and also with respect to the corresponding step in the decision-making process. In the following discussion, the notation $z_{i,n}$ will be used to refer to a generic element z of agent i at step n. For simplicity, the step index will be omitted if it is clear or unspecified in a given context.

Definition 2.4.1. *A Decentralized Partially Observable Markov Decision Process (Dec-POMDP) is a tuple* $\langle d, \mathcal{S}, \mathcal{A}, T, \mathcal{O}, O, R \rangle$*, where:*

d *is the number of agents;*

$\mathcal{A} = \prod_{i=1}^{d} \mathcal{A}_i$ *is a set of* joint *actions.* A_i *contains the individual actions of agent* i*. Each joint action* $\mathbf{a} \in \mathcal{A}$ *is a tuple of individual actions* $\langle a_1, a_2, \ldots, a_d \rangle$*, where* $a_i \in \mathcal{A}_i$*.*

$\mathcal{O} = \prod_{i=1}^{d} \mathcal{O}_i$ *is a set of joint observations* $\mathbf{o}$*.* O_i *contains the individual observations of agent* i*. As with actions,* $\mathcal{O} := \{\langle o_1, o_2, \ldots, o_d \rangle \mid o_i \in \mathcal{O}_i\}$*;*

$\mathcal{S}, T, O, R$ *have the same definition as in a Partially Observable Markov Decision Process (see Definition 2.3.1), and are mapped, where appropriate, by joint actions and joint observations;*

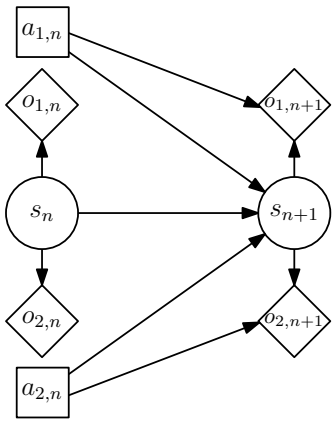


Figure 2.7: A 2-DBN representing an example of a two-agent ($d = 2$) Dec-POMDP.

In this case, the goal of the decision-making problem is to maximize the reward gathered by the team (since R maps from joint actions and states), so it is assumed that

all agents are cooperating towards a team-driven behavior. However, it is very important to note that, in a general Dec-POMDP, observations are not necessarily shared amongst the team: in general, each agent i receives only its respective observation o_i (see Figure 2.7). This raises an important issue regarding inter-agent coordination, since agents will have to take actions which jointly maximize collective reward, based only on locally available information.

The amount of local information which is available to an agent is one of the fundamental features which are characteristic to each sub-class of this framework. As a relevant example, if each agent is allowed to observe part of its environment, but the composition of all individual observations allows the joint state of the system to be univocally identified, such that there is a surjective mapping $G : \mathcal{O} \rightarrow \mathcal{S}$, then the system is said to be *jointly fully observable*, also known as a Dec-MDP (Bernstein et al., 2002). Note that for each agent, the problem may still be partially observable from a local perspective. The presence of communication constitutes another characteristic feature which further refines each of these model classes (see Section 2.4.2). However, in the most general setting, nothing is assumed regarding any of these qualities.

If communication is unavailable, this makes it impossible to maintain a *joint* belief over the global state of the process, since, through (2.21), the observations of all agents would have to be shared at each time-step. Without such a sufficient statistic to exploit the Markovian nature of the process, agents are forced to reason over temporal sequences (*histories*) of actions and observations, each of which might have led the system to a particular state. Joint policies, in this context, then become maps of action-observation histories to actions. Furthermore, since, for each agent, both the actions and observations of all other agents remain unknown, the number of all *possible* such histories increases exponentially with the number of decision steps (in particular, there are $(|\mathcal{A}||\mathcal{O}|)^n$ histories at step n).

Although optimal Q-value functions have been shown to exist for general Dec-POMDPs by Oliehoek et al. (2008a), which allow a particular action-observation history to be valued with respect to the optimal joint policy, the same authors prove that obtaining an optimal policy based on such a function through DP, as in the case of (PO)MDPs, is intractable, except for very small-scale problems. Optimally solving a non-communicative Dec-POMDP is a provably NEXP-complete problem (Bernstein

et al., 2002). However, optimal algorithms for finite-horizon Dec-POMDPs exist, and act as a benchmark for small-scale problems (Hansen et al., 2004; Spaan et al., 2011).

In order to circumvent the inherent computational intractability of finding the optimal solution, common Dec-POMDP algorithms approximate the optimal result either through point-based DP (Seuken and Zilberstein, 2007), a search in the space of possible joint policies (Szer and Charpillet, 2005), or by representing the problem as one of Expectation-Maximization, over the parameters of Finite-State Controllers (FSCs) which encode the joint policy (Kumar and Zilberstein, 2010; Pajarinen and Peltonen, 2011).

Outside of the MDP framework, Dec-POMDPs can be viewed as a subset of a yet more general mathematical framework developed in the scope of game theory, that of Partially Observable Stochastic Games (POSGs)(Oliehoek, 2010). In that case, the reward structure is generalized so that agents may be pursuing individually-defined rewards and goals, as opposed to being restricted to cooperative behavior. This allows *adversarial* decision-making problems to be modeled, which fall outside of the scope of this work. However, some state-of-the-art solution methods for general Dec-POMDPs draw upon game-theoretical concepts, such as representing the decision-making process at any given time as a one-step Bayesian Game (Oliehoek et al., 2008a).

2.4.2 Modeling Communication

In practical multiagent applications, it is often reasonable to assume that some form of communication can be established between different agents. This is particularly valid in the domain of cooperative robotics.

The availability of communication raises three additional decision-making problems:

- What to communicate;
- When to communicate;
- Who to communicate with.

In order to address these issues, various extensions to the Dec-POMDP framework were proposed, which introduce additional structures to explicitly model communication decisions and costs, such as the COM-MTDP framework (Pynadath and Tambe, 2002).

However, obtaining a solution to these types of models remains NEXP-complete problem, since, even if agents are able to maintain a joint belief state through communication, overcoming the need to reason over complete histories of actions and observations, the number of possible communication decisions to be evaluated would once again increase the complexity of deciding the joint policy (now both an *action policy* and a *communication policy* would be required) by an exponential factor. In the worst case, messages can be too costly for any communication decision to be taken, and so the problem is once again reduced to a non-communicative scenario.

Goldman and Zilberstein (2004) showed that exchanging observations between agents, at each time-step, is at least as good as any other inter-agent communication language. This fact allows problems in which free communication is assumed, and observations are shared between agents, to be viewed as centralized versions of their respective model classes. For example, under these circumstances, Dec-MDPs reduce to centralized MDPs (this had already been presented by Boutilier (1996)), also known as *Multiagent* MDPs (MMDPs). In contrast to the NEXP-complete complexity of solving a general Dec-MDP, finding the solution to an MMDP a is P-complete problem (Papadimitriou and Tsitsiklis, 1987), since it is the same problem as that of solving a single agent MDP, with as many individual actions as there are joint actions in the Dec-MDP case.

This difference in computational complexity prompts a different approach to the problem of modeling communication: if the agents are assumed to communicate freely with each other, and exchange observations at each time step, the centralized model may then be solved at a lower computational cost. Finally, communication usage can minimized *a posteriori*, by analyzing the resulting joint policy. This approach was taken by Roth et al. (2007) for Dec-MDPs. However, when dealing with teams of robotic agents, it is not often reasonable to assume that the state of the system is jointly-fully observable. It is then more natural to apply this approach to the general class of Dec-POMDPs, that is, to consider that through free communication, these models can be regarded as centralized Multiagent POMDPs (MPOMDPs). By doing so, solving the model becomes a PSPACE-complete problem. This is the approach taken in Roth et al. (2005b), and also in this work (see Section 3.2.7.1). The main disadvantage of using model centralization is that *implicit* communication is disregarded (Goldman and Zilberstein, 2004). This is the information which can be extracted from the dependencies in the actions and observations of different agents. In turn, this

implies that the resulting communication policies may be of inferior quality than if an explicit approach to communication is used instead. Another relevant issue is that fully centralized models cannot easily describe communication failures or delays. Spaan et al. (2008) describe a method to reason over stochastically delayed communication (with up to one step of delay) in this setting, using Bayesian Games to solve the respective planning problem.

When considering a centralized, team-wise model, it is advantageous to maintain it in a factored format (discussed in the following section). Otherwise the dimensions of the model components may grow exponentially in the number of agents.

2.4.3 Factored Models

All MDP-based frameworks described so far suffer from what has come to be known as the "curse of dimensionality" - that is to say, the required space and time for the solution of these models grows exponentially with the cardinality of the model components (Bellman, 1957b). Besides its effect on computational complexity, this exponential nature of the total number of elements in each component of a stochastic model implies that, for planning approaches, even the process of filling in their values quickly becomes impractical if these conditional probability distributions (CPDs) are represented in a tabular form (T and O have, respectively, $(|S|^2|A|)$ and $(|S|^2|A||O|)$ elements), and if this process cannot be automated.

This greatly impacts the scalability of such models, and, as such, has led to the development of alternative representations which aim to mitigate this effect. One such alternative is to obtain a *factored* representation of the models, in which each of the components of the system may be described as a combination of characteristic factors, which are typically exponentially smaller in size (Boutilier and Poole, 1996; Hansen and Feng, 2000; Poupart, 2005).

Factored models exploit the structure of a given problem to achieve representational simplicity in two different ways: first, by considering that each state/action/observation factor can be conditionally dependent on only a subset of other factors, which greatly reduces the number of variables involved in the definition of each CPD; secondly, through the use of decision diagrams or trees to represent the structure of each CPD, further simplifying their description.

In this sense, the Dec-POMDP framework, as described in Definition 2.4.1, already possesses naturally factored action and observation spaces (an *agent-wise* factorization). A factored Dec-POMDP may further extend this property to the state of the system, by introducing a set of k state factors $\mathcal{X} = \{\mathcal{X}_i : i \in \{1, \ldots, k\}\}$, so that, in turn, $\mathcal{S} = \prod_{i=1}^{k} \mathcal{X}_i$. Each $\mathcal{X}_i$ is arbitrarily sized.

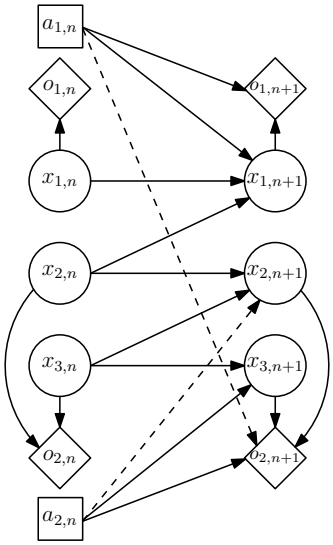


Figure 2.8: A 2-DBN representing an example of a factored two-agent Dec-POMDP. Note that the number of state factors (3) and the number of agents (2) is not necessarily the same.

Although the Dec-POMDP example is here used, factorization can be applied to any of the previously described frameworks, without any loss of generality. A graphical structure such as the DBN presented in Figure 2.8 accompanies most factored representations, which allows the conditional dependencies between the various model components to be made clear. It also enables the compact representation of the transition and observation models of the system as sets of local Conditional Probability Tables (CPTs) (Poupart, 2005), or as Algebraic Decision Diagrams (Hoey et al., 1999), for further savings in the amount of space required to contain this information. Several of the currently fastest, most scalable solution algorithms for (PO)MDPs use these properties, such as SPUDD (Hoey et al., 1999) and Symbolic Perseus (Poupart, 2005).

The notions of conditional independence expressed by factored models are of particular relevance in multiagent scenarios. General Dec-POMDPs do not assume any particular structure in the interaction between agents, but if a factored representation

is possible, it will often highlight the independency between the effects of the actions of certain agents, or groups of agents, at a given step. This originates the concept of *locality of interaction* in multiagent problems (Oliehoek et al., 2008b), which refers to the fact that agents will typically influence others which are acting in the same "neighborhood"[1] but may act independently from distant or unrelated agents, and so will tend to form *local* groups of cooperating agents, according to the structure of a particular scenario. The reward model can also be factored, in this case, by considering the joint reward gathered by the team, at a given step, as the summation of local reward terms associated with different clusters of state factors and agents. This additive property extends to the value function, and therefore the maximization of each of the involved terms becomes a simpler problem involving an exponentially smaller number of variables. These considerations may greatly simplify the problem of calculating a joint policy in cases where interactions between agents are sparse.

It is important to note, however, that the type of inter-agent (in)dependency so far described is typically only valid instantaneously, or for very short horizons - as the system evolves, its factors will rapidly correlate, unless they constitute strictly independent subprocesses over all steps(Boyen and Koller, 1998). If strict independence is indeed assumed, then the model can be classified as being *Transition-Observation Independent* (TOI). A TOI subclass of Dec-POMDPs is that of Network-Distributed POMDPs (ND-POMDPs) (Nair et al., 2005). In such a model, the decision problem is only coupled through the factored reward model (in fact, if the reward model could also be strictly decoupled across all agents, a TOI scenario could instead be modeled as separate decision problems from the onset). This assumption allows for greatly improved scalability, but in practice it often severely limits the modeling capabilities of the framework. A more general approach is taken by Transition-Decoupled POMDPs (TD-POMDPs)(Witwicki, 2012), which assumes that the problem can be decoupled for a particular agent if the policies of other agents are assumed fixed. This allows the non-concurrent influence of mutually shared state factors by different agents.

Another relevant property of factored models is that, in the general case, the dependencies between the actions of different agents may change depending on certain

[1]"Neighboring" agents are commonly thought to be those with concurrent influence arcs in the 2-DBN representing the problem, although this notion can be extended for any number of steps. It often correlates with, but does not necessarily imply, a spatial relationship between agents.

variable assignments (here state factors, actions), within a given decision step. If so, the variables are said to exhibit *context-specific* independence (Boutilier et al., 1996), which can be exploited to further reduce the amount of space required for the associated stochastic models.

Chapter 3

On the Practical Implementation of MDPs and Related Models

Although MDPs and their generalizations have been shown to form a powerful and versatile decision-theoretic framework, which can model, and solve, a wide array of AI applications, the practical implementation of these frameworks in real-world scenarios is not as widespread as the existing amount of underlying theoretical development could imply. A common criticism against the use of MDP-based extensions lies in the inherent computational complexity of obtaining their respective solutions. Although this is not so significant for classical discrete MDPs, which have been shown to be solvable in polynomial time (Papadimitriou and Tsitsiklis, 1987), it is particularly valid for their partially observable and decentralized extensions, which range from PSPACE-complete (finite-horizon POMDPs) to being undecidable (infinite-horizon (Dec-)POMDPs) in terms of complexity (Bernstein et al., 2002; Papadimitriou and Tsitsiklis, 1987)). This ultimately impairs the scalability of these models, unless significant structure is present, and it can be exploited through techniques such as factorization (see Section 2.4.3).

This chapter will present a brief review of existing practical implementations of MDP-based frameworks (Section 3.1). Section 3.2 explores the current practical limitations of POMDPs, for teams of agents in realistic environments. A small-scale multiagent scenario, which was developed in the scope of this work as a means of exposing these limitations, is also introduced and evaluated.

3.1 A Review of MDP-Based Applications

The importance of documenting real-world applications of the MDP framework had already been recognized as early as the work of White (1985), leading to subsequent surveys on the topic by that author (White, 1988, 1993). An important distinction is made in those works, between accounted implementations of MDPs based on real-world data, which act as a proof-of-concept for the underlying theory, stimulating further research, but that have gone no further in practice; and those which have provided decision-making policies that have actually been implemented in industry, finance, or other activities outside of the research environment, or have at least influenced relevant decisions made therein. In the latter sense, and despite an early, *apparent* lack of applications of the MDP framework, there was ample evidence of its versatility through the diversity of areas in which its practical use, over time, had indeed been reported. Notable examples include:

- The operation of hydroelectric power systems (Lindqvist, 1962; Little, 1955; Wang and Adams, 1986). The objective of these applications is to control the amount of water contained in a reservoir, which acts as a source to hydroelectric generators, such as to optimize the economic efficiency of the plant. The state of the system represents the level of water in the reservoir, and has an associated (economic) cost. The inflow to the reservoir is uncertain, since it originates in a natural water course. This domain of application for MDPs (and closely related *stochastic programming* techniques) remains active to this date (Fleten and Kristoffersen, 2008; Pritchard et al., 2005);

- Financial applications, such as managing investments while maintaining an expected balance, taking into account uncertain expenses and returns (White and Norman, 1965), or uncertain time to the next investment opportunity (Derman et al., 1975). The reader is referred to the work of Bäuerle and Rieder (2011) for recent developments in this field;

- Inventory management (Chao and Manne, 1983; Fabian et al., 1959), a classic problem in Operations Research. The level of resources in stock is modeled as (part of) the state of the system, subject to changes through demand, and decisions correspond to restocking operations associated with possibly uncertain costs;

- Maintenance and repair scheduling (Dreyfus, 1957; Duncan and Scholnick, 1973), where the age and condition of the respective equipments is taken as the state, subject to random failures, and decisions correspond to repair operations, inspections, or the purchase of components;

- Queueing management (Ignall and Kolesar, 1974; Low, 1974). In these problems, the number of elements in a queue, which has an uncertain arrival rate, is typically modeled as the state of the system. Each element is assigned a utility for the service, as well as a potential waiting cost. Decisions can consist of accepting or rejecting new elements, or changing the utility of each element, thereby affecting the arrival rate. This is a particularly appropriate field for the use of the CTMDP and SMDP frameworks;

- Agriculture (Kristensen, 1994; Onstad and Rabbinge, 1985), where the population level of a species can be modeled in the presence of limited resources or epidemic events. Decisions can include disease treatment options, or the introduction of new individuals.

The reader is referred to Feinberg and Shwartz (2002); White (1993) for more thorough surveys of real and potential MDP applications.

Another particularly relevant area of real-world probabilistic decision-making, in which there is growing interest in the theory of Markov processes, is that of medical prognosis and treatment (Ahn and Hornberger, 1996; Hauskrecht and Fraser, 2000; Shechter et al., 2008). However, this field most often uses Markov models to perform *a posteriori* analysis and inference over real data, and there are still few reported implementations of decision-making policies obtained through the solution of MDPs, as part of the clinical process (Díez et al., 2011; Schaefer et al., 2005; Sonnenberg and Beck, 1993).

Due to pioneering works such as those of Witten (1977), Sutton (1984), and Watkins (1989), MDPs became known as a fundamental framework for the study of Reinforcement Learning (RL), triggering rapid development in that sub-field of AI. The increased availability of computational power, together with the advances in dynamic programming, enabled the development of a wide number of successful game-playing agents, for scenarios such as backgammon (Tesauro, 1992), Tetris (Tsitsiklis and Van Roy, 1996),

and chess (Thrun, 1995), among others (Kaelbling et al., 1996). MDP-based RL has also become, since then, an increasingly popular option to address complex robotics applications (Kober et al., 2008; Mahadevan and Connell, 1992; Mataric, 1994; Ng et al., 2006; Riedmiller et al., 2009), due to their performance and simplicity of implementation when compared to model-based approaches.

In many real-world scenarios, the assumption of full observability is often impractical, and standard MDP models cannot provide accurate representations of the underlying systems. The POMDP framework provides the necessary tools to model such partially observable problems, and so extends the domain of practical applications which can be formally addressed through decision-theoretic approaches. Early documented applications of POMDPs to real-world settings were mainly related to inspection and maintenance problems, similar to those described for the MDP case, but in which the condition of an equipment can only indirectly known through examination, and so the true state of the system remains unknown (Eckles, 1968; Ellis et al., 1995; Ohnishi et al., 1986). Since then, and accompanying the development of this framework, POMDPs have been shown to be suitable models for a wide range of applications (Cassandra, 1998b), including:

- Population modeling and control, when considering an exact enumeration of individuals to be infeasible, such as in fisheries (Lane, 1989);
- Corporate decision-making, for problems such as determining whether or not to perform an internal investigation of a given department, the status of which is only known indirectly through periodic reports (Kaplan, 1969);
- Financial applications such as dynamic price allocation with unobservable demand (Aviv and Pazgal, 2005) or stock portfolio management (Wang and Cao, 2011), considering that stock values are volatile and subject to random changes;
- Spoken dialog management (Williams and Young, 2007; Young et al., 2010), considering that speech recognition is fallible, and so the "state" of a dialog can only be observed with uncertainty;
- Medical decision-making (Hauskrecht and Fraser, 2000), extending similar MDP applications to the more realistic setting in which the presence of an underlying

disease can only be tested through investigative actions. Real-world applications of continuous-time POSMDPs in this domain have been reported by White et al. (1982);

- Robotics problems, in which POMDPs often provide more realistic models than their fully observable counterparts, since the sensing capabilities of the agents can then be taken into account in the decision-making process. POMDPs have been applied to mobile robot navigation (Koenig and Simmons, 1998; Simmons and Koenig, 1995; Spaan and Vlassis, 2004; Thrun, 2000), object grasping (Grupen and Coelho, 2002; Hsiao et al., 2007), target tracking, (Hsu et al., 2008), control of unmanned aerial vehicles (Miller et al., 2009), and diverse other robotics applications (Paquet et al., 2005; Pineau and Thrun, 2002). The POSMDP framework has also been briefly studied in these scenarios (Lim et al., 2004; Mahadevan and Khaleeli, 1999);

However, of the reported *potential* applications of the POMDP framework, only a small subset is known to have produced results affecting lasting, real-world decision-making problems (of the above, Eckles (1968); Ellis et al. (1995); Ohnishi et al. (1986); White et al. (1982)). It is fair to conclude that in real-world applications, system engineers tend to compromise between model realism and computational/representational complexity, opting for simpler, if not inaccurate, models such as MDPs in order to avoid the inherent intractability of large-scale partially observable problems.

Although RL techniques have also been extended for POMDPs (Jaakkola et al., 1995; Kaelbling et al., 1996), the necessary increase in the sample complexity of approximating optimal policies through learning has stunted the application of these techniques beyond those of academic examples. Recently, Vlassis et al. 2009 has reported the successful application of a learning method for POMDPs, based on the EM algorithm, to the control of a (single) robotic platform.

Research on multiagent decision making has been increasingly active over the last decade. MDP extensions for multiagent systems have been shown to be applicable, in simulated environments, to the control of general distributed systems involving virtual agents (Becker et al., 2004; Oliehoek and Visser, 2006; Pajarinen and Peltonen, 2011; Paquet et al., 2005; Tao et al., 2001), specific game-playing scenarios (Wu and Chen,

2008), and sensor networks (Nair et al., 2005)[1]. However, we note that these application examples concern *virtual* agents. In contrast with the appreciable success of MDP theory on real single agent domains, there have been comparatively very few reported applications of any of its derived frameworks to the control of real multiagent systems. In the field of cooperative robotics, there are few, but noteworthy applications of multiagent MDP theory (Bowling and Veloso, 2003; Capitán et al., 2013; Emery-Montemerlo et al., 2005; Matarić, 1997). These works have regarded cooperative robotics as an appropriate case study for DT methods, but have not explicitly addressed the practical problems involved in the application of those methods to physical agents, and ultimately addressed those problems in an *ad-hoc*, unstructured manner. Providing an in-depth look at the properties and requirements of multi-robot systems, from the perspective of decision-making, constitutes one of the objectives of this thesis, since it may promote wider acceptance of MDP theory to applications in cooperative robotics.

3.2 POMDPs for Real Teams of Robots

In this section, we will describe the steps involved in the process of modeling a real team of robots as a POMDP. We will expose the most typical difficulties involved in this procedure, and review the different techniques which have been proposed, in the associated literature, to address some of those problems. We will also underline those problems that remain largely unaddressed, and which will form a basis for the novel work presented in the remaining chapters of this thesis.

Alongside this step-by-step exposition, we will also put these modeling methods to practice in a cooperative robotics scenario, which will act as a running example across the chapter. The case-study for this example is robotic soccer, a widely known environment for cooperative robotics (Riedmiller et al., 2009; Spaan and Groen, 2003). At the end of this section, our case-study is then tested in a realistic simulator, which takes into account the physical properties of the agents.

[1]Note that, since the decision-theoretic frameworks which are here being considered assume a cooperative nature between agents, adversarial situations, such as common financial applications, are not here considered (these scenarios fall typically in the domain of *stochastic game theory*, although some MDP extensions have been shown to accommodate adversarial representations (Ng et al., 2010))

3.2.1 A Case Study in Robotic Soccer: Overview

Robotic soccer is a particularly challenging environment for applied AI, in that it is highly reactive and uncertain, but at the same time requires thorough cooperation between agents.

A common task in robotic soccer is to have various robotic soccer players cooperating in order to take the ball towards their opponent's goal. Since this is a conceptual exercise, the analysis will here be limited to a scenario with two cooperating robots, although the same considerations could be made for a scenario with a larger team of robots (assuming that at least one computationally tractable model for the problem exists).

Figure 3.1: A typical in-game situation in RoboCup Middle-Size League Robotic Soccer, showing cooperation between two robots.

A snapshot showing an example of cooperation between two robots in a real in-game setting is shown in Figure 3.1. In this two-agent case, one of the players should carry the ball forward, and the other should position itself so that it may receive a pass from its partner, if necessary. The robots may choose to pass the ball in order to avoid imminent obstacles, since it is difficult to avoid obstacles while carrying the ball. The robot that carries the ball at any given time will be referred to in this case study as the "Attacker", and its partner the "Supporter". Whenever a pass occurs, the roles of the robots should switch, meaning that an Attacker becomes a Supporter and vice-versa. The Attacker should then kick the ball to the goal as soon as it detects an opportunity to score. The initial position of the robots and of the ball in their field of play is unknown, and so is their role. The team should then cooperatively decide which robot should carry the ball. During the course of their task, the robots may encounter obstacles that they should

be able to avoid, although the position of these obstacles is not known beforehand. The robots possess sensors to detect their own location, with uncertainty, the position of the ball, and any surrounding obstacles.

3.2.2 Identifying an Appropriate DT Framework

The first step in modeling any system as a decision-theoretic problem is to select the appropriate framework, from among the many existing alternatives (some of which have been described in Chapter 2), which can account for its requirements.

The following queries act as a guideline to identify potential DT frameworks for a given scenario, according to each of the fundamental qualities of a generic decision-theoretic model, presented in Figure 2.1:

1. Is the state of the system fully observable?
2. Are there multiple agents involved, and is free communication reasonable?
3. Is the influence of continuous time relevant for the decision-making process?

These queries allow a problem designer to rule out those DT frameworks that should *not* be used in a particular situation, if they assume system properties that are not verifiable in practice. In most cases, however, there can still be multiple frameworks, differing in their generality, which can be used to model a given system. These frameworks can still carry underlying assumptions that are acceptable in practice, making them easier to model and solve (or train), but may come at the expense of system performance during execution.

The problem designer is then faced with a trade-off between *expected* system performance, and the operational complexity of the solution to a model. To make matters worse, when the modeling process is complete, this selection typically becomes definitive, or at least very difficult to revert – as we will still argue to a further extent, the amount of work which is invested into the deployment of a particular DT framework in practice is hard to reuse; if any of the assumptions regarding the properties of a real system were to change, there is no guarantee that any of the prior modeling work would carry over to a different DT framework.

A prime example of a real-world application that can be modeled through different DT frameworks is that of medical decision aid: several reported solutions assume the

states of the system, directly related to the health of a patient, to be fully-observable through laboratorial procedures or other forms of analysis (Schaefer et al., 2005); others take into account the limited reliability of diagnostic procedures, treating the underlying causes of symptoms as partially-observable (Hauskrecht and Fraser, 2000; White et al., 1982). Conceptually, the latter approach is more realistic; however, since fully observable models are easier to solve, they also allow for more complex state descriptions (e.g. variables describing the physical condition of the patient), so their performance is not necessarily worse. Ultimately, the best approach in practice is the one with the highest rate of correct diagnostics.

In multiagent scenarios, which are main the focus of this work, this trade-off between computational complexity and model realism also hinges on the availability of communication between agents: as it was pointed out in Section 2.4.1, scenarios in which communication policies must be determined alongside action policies, or in which no communication is possible, are subject to an exponential increase in computational complexity when trying to obtain an approximately optimal policy. This problem, in itself, constitutes one of the major obstacles to the practical implementation of decentralized decision-theoretic models in real scenarios. This issue is here given additional relevance, since it forms one of the topics in which this work will provide a novel solution.

Remark 3.2.1. *Multiagent decision-theoretic frameworks which explicitly model communication, or which consider non-communicative agents, are difficult to scale up to realistically-sized problems.*

Applying these aforementioned considerations to the presented robotic soccer case study, it is clear that the specified environment is partially observable. Modeling this property is unavoidable, since the observations of each agent are uncertain, and even through their union the full state of the system cannot be known. Our selection is then limited to the decentralized extensions of POMDPs. Of these, the simplest is the MPOMDP class (which, as was described in 2.4.2, is equivalent to a team-wise single agent POMDP) although it assumes free communication between agents. Since this is not limiting assumption in this particular situation, and it allows for a richer state and action space description, this will be the chosen framework to model the problem.

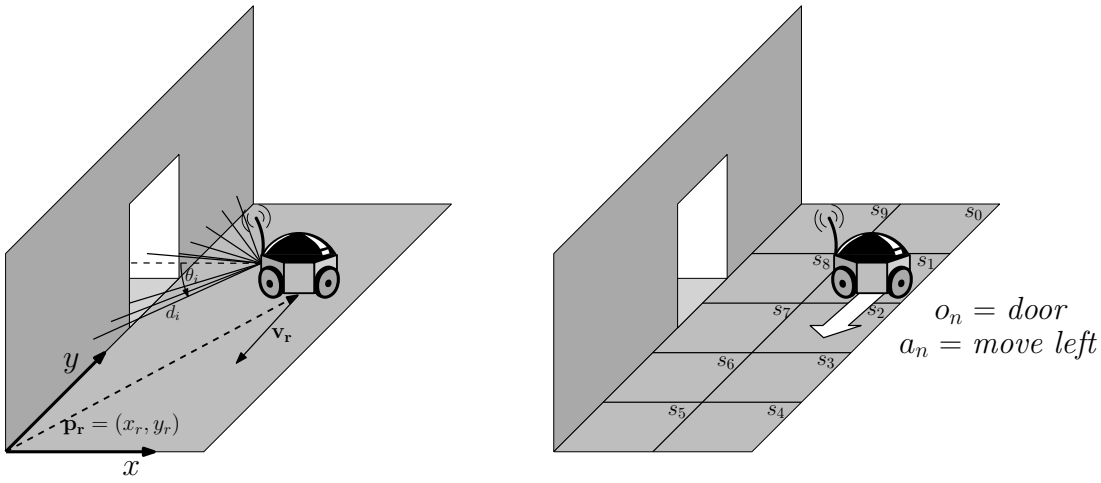


Figure 3.2: The contrast between the physical operation of a mobile robot and its decision-theoretic interpretation. a) For a typical mobile robot, the state (its position), controls (velocity) and sensor data (here depicted as a set of range-and-bearing readings) are all continuous and multidimensional; b) Most tractable DT models assume a discrete, symbolic representation for states, actions, and observations. Mapping such symbolic representations to/from the actual physical system is a non-trivial problem.

3.2.3 Modeling States, Actions, and Observations

As physical systems, an exact description of the dynamics of a team of robots while interacting with their environment would necessarily be defined over continuous-domain variables. As we have reviewed in Chapter 2, while DT frameworks allow for continuous state, action, and/or observation spaces, there have been few reported applications of such models in multiagent settings, due to the analytical complexity of their description, and consequently, of their operational complexity.

It then becomes necessary, for practical multiagent systems, to approximate physical properties, control inputs, and available sensorial data through a set of suitably defined discrete states, actions, and possibly observations. A depiction of this problem is shown in Figure 3.2.

Modeling continuous physical variables into a discrete structure, however, is an issue worth careful consideration. On one hand, the cardinality of the state, action, and observation sets should be evidently kept to a minimum, since this will reduce the overall operational complexity of the model; on the other, reducing the number of

possible values for these variables may aversely affect the performance of the agents, since the system dynamics become less descriptive.

Let us first consider the problem of discretizing the system state. Principled methodologies to determine suitable discretizations of a generic continuous state space, given a training set of interactions of physical agents with their environment, include vector quantization and related clustering techniques (Fernández and Parker, 2001), neural networks (Miller et al., 1990) and decision / regression trees (Uther and Veloso, 1998). Despite being widely used in the context of robotics, there are notable drawbacks to these techniques: they assume that the physical state of the system can be known with certainty, at least for the purpose of collecting the training data; and they assume that the action space of the model is already discrete and known *a priori*. When either of these assumptions is not valid, the physical state space is discretized using prior knowledge about the system, typically by considering each state to be an assignment of discrete system features, as in tile coding (Sutton and Barto, 1998), or more generally through factored representations, as described in Section 2.4.3 (Oliehoek et al., 2008b; Poupart, 2005).

As for observations, similar arguments can be made. The model may consider characteristic features of the environment which are observable in a given state (such as landmarks, or characteristic patterns in sensor data, etc.), and map them to discrete, symbolic values. The problem of classifying observations from sensorial data can be approached with essentially the same methodologies as the problem of state discretization. The simplest methodology is to manually design a set of decision rules that classify raw sensor data into the desired set of features, as in the work of Nourbakhsh et al. (1995). Other reported applications use neural network training methods to the raw sensor data, as described by (Mahadevan and Khaleeli, 1999), or clustering techniques (Duda et al., 2001) to produce $|\mathcal{O}|$ class representatives of the input data, subsequently matching each new data point to its nearest representative vector. It is also possible to consider DT models with continuous observation spaces (Hoey and Poupart, 2005). However, planning methods for such models have not been shown to be scalable to multiagent problems.

An often overlooked difficulty in extracting symbolic observation data from sensorial inputs is that the decision-making loop of a robotic agent is rarely operating at the same frequency as its sensors. It is common, furthermore, for a robotic platform to have

multiple sensors producing different sets of data at different frequencies, and accepting control inputs at yet another rate. It becomes necessary, then, to decide how much data to use, when classifying it into symbolic observations, within a decision episode. One simple heuristic is to perform classification over every incoming data set, as soon as it arrives, and use the last returned label at the instant of each decision step as its observation. This is essentially a *zero-order hold* of the output of the classifier. Another approach, which is more robust to noise but more complex to implement, is to model the classification process as an HMM which is synchronous with a stream of sensor data, but not with the decision-making loop of the agent. At each decision step, the most likely label (the most likely state in the HMM) at that time can be used as the observation of that step.

It may also be assumed that sensor processing modules, operating in the robot in real time independently from its decision-making, are able to produce not only an estimate of the physical state of the robot, but also an associated belief distribution over its configuration space. For example, this is typically the output of a dedicated localization algorithm. If this belief distribution over the continuous state can be marginalized over a discretized state structure, for example over a set of topological locations, then the resulting "discrete" belief can be used directly for decision-making purposes. This bypasses the need to classify observations at run-time, although they must still be considered in the DT model.

Actions, in turn, are discretized by considering them as abstract processes which control the autonomous agent(s) through a sequence of continuous inputs. These inputs, for a robotic agent, typically correspond to actuator speeds or torques. In that case, an action can involve the closed loop control of an agent at a lower level of abstraction (*e.g.* a navigation action of a mobile robot to a given pose), or it can be a direct, open-loop assignment of real-valued control inputs (*e.g.* motion in a given direction at a constant speed). In other cases, actions can already have an inherently discrete, symbolic form to begin with. In human-robot interaction, for example, each step of a communication episode, involving the transmission of text or sound to the user, can be considered as an action.

Having covered the typical discretization methodologies for states, actions, and observations, it is important to note that there is an implicit mutual dependency between all model dimensions. The adequate level of abstraction for actions and observations,

for example, is dependent upon the particular form of the state space which is chosen for the decision-making model. Of course, the same argument is valid for the definition of the state space itself, if there is prior knowledge regarding the actions and observations that should be in the model — none of these free variables can be discretized without considering the correlation, at a semantic level, of the resulting discretized components.

As an example, Figure 3.3 depicts the interdependency between symbolic states and actions, by comparing two different approaches to the problem of discretizing the configuration space of each robot in our conceptual robotic soccer environment. Finer, grid-like based discretizations, which can accurately describe the position of an agent in the physical world (Figure 3.3a) are typically useful for problems of navigation under uncertainty, and require an appropriate discretization of low-level motion controls (such as moving in a given direction for a fixed amount of time/space). However, in partially observable domains, the fan-out of possible belief states at the problem horizon h grows exponentially (there are at most $(|\mathcal{A}||\mathcal{O}|)^h$ possible belief states at step h), which can make these approaches quickly intractable. An alternative is to consider a state space description such as the one depicted in Figure 3.3b, in which states are more closely related with logical predicates (such as if the robot is near the goal, or in its own side of the field), but carry less information about the particular configuration of the agent in physical space. This is useful for high-level decision making, since it allows for descriptions of more complex actions, such as pushing a ball towards the goal, which assume a greater level of autonomy for each robot. This *topological* description of the system state is also characteristic of Discrete-Event Systems approaches (Neto et al., 2004).

Following these considerations, the problem of modeling the environment through a discrete DT framework can be viewed as one of selecting the "level" of abstraction over which the decision-making should be carried out. This, of course, is not only a problem dependent, often subjective decision in itself, but one which is left up to the problem designer. The designer then requires in-depth knowledge of the system *and also* of the selected decision-theoretic framework itself, in order to predict the performance of the controlled system when its respective DT policy is deployed in the real world. This may limit the acceptance of MDP-based frameworks in real-world activities where expert knowledge of these matters is not readily available.

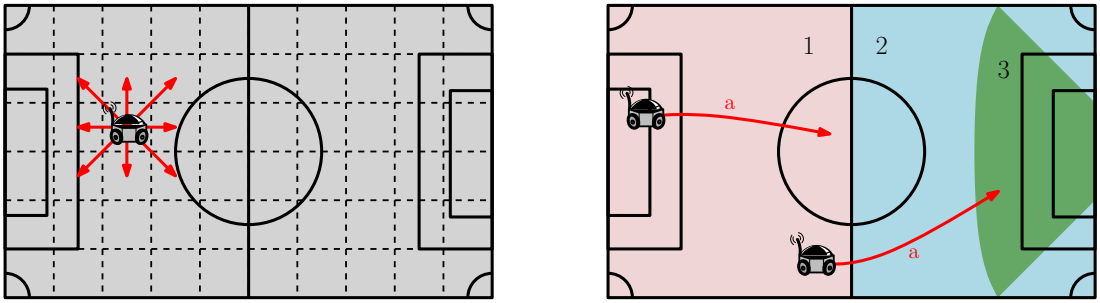


Figure 3.3: Two possible state discretizations for a robotic soccer environment: (a) A grid-like metric discretization, typical for navigation problems. The actions of a robot may move it to adjacent cells; (b) A discretization based on logical *predicates*, useful for high-level decision-making. Here, states take a more logical interpretation, such as "being in either side of the field" (states 1 and 2), or "being near the goal" (state 3). Actions may be seen as complex *behaviors* for a robot. The represented action attempts to move the robot closer to the goal. However, the Markovian property of the system may be broken.

Remark 3.2.2. *For physical systems, the number of states and actions (and possibly observations) of an associated discrete DT model are correlated free parameters, and the common criteria for their selection involve a trade-off between operational complexity and predicted system performance. Discrete models constitute potentially lossy approximations of continuous domains.*

A popular approach in AI to model complex systems is to deal with simultaneous decision-making at various levels of abstraction, modeling the problem hierarchically. This approach has been successfully explored both for fully observable MDPs (Parr, 1998; Sutton et al., 1999) and for POMDPs (Pineau and Thrun, 2002; Theocharous and Mahadevan, 2002; Theocharous et al., 2005). These approaches typically consider semi-Markov high-level decision theoretic models in which the notion of state and action is abstracted, and each such pair may encompass the execution of a lower-level task, itself modeled as a potentially fully-Markovian problem. This allows, in some cases, for a representation which is closer to decision-making processes carried out in realistic applications, which may, at different times, fall into different levels of abstraction. It can also provide very advantageous savings in terms of computational complexity for planning in large-scale problems; and in terms of the amount of experience required in reinforcement learning applications, since solutions for a particular low-level task can

potentially be applied in similar domains which share the same dynamics, through the use of techniques such as *reuse learning* (Theocharous and Mahadevan, 2002).

Returning to the robotic soccer case study, and recalling that the focus of this example is on conceptual clarity, a straightforward approach is taken regarding environment modeling. The problem is considered to be one of high-level decision-making, in accordance with the description presented in Section 3.2.1, where it is assumed that the robots are equipped with basic *behaviors*, and that the decision-making problem is one of cooperatively deciding which of these behaviors should be performed by each of the agents at any time.

3.2.3.1 States

The robots are known to operate in a soccer field, which contains the agents themselves, the ball, and an unknown number of opponents. Except for these obstacles, their navigation is free inside the field. The state of the overall system can then be encoded through the localization information of the robots, the position of the ball, and the positions of possible obstacles:

- Regarding localization, the field of play is discretized into four different sections, similar to the representation of Figure 3.3b. The agent may be located in its own half-field, in the half-field of the opponent team, near the opponent team goal, or in a shooting opportunity, which requires the robot to be near the goal while carrying the ball, and also turned towards it. The robot may also use localization information to sense if it is ready to receive a pass from its partner;

- The information regarding the presence of obstacles can be encoded in a logical form, meaning that the robot is either blocked by obstacles or free to move in its current direction;

- Finally, the robot is also able to detect whether or not it is in possession of the ball, which constitutes an additional Boolean state variable.

The total number of states is $|\mathcal{S}| = 48$. The full state space of the model, encoded in factored form, can be found in Appendix A (as well as the action and observation spaces for this problem).

Note that the robots share their localization information —they require this information in order to be able to follow each other and to be able to sample their own observations. This is accomplished through explicit communication, as described in the Section 3.2.7.1.

3.2.3.2 Observations

The set of possible observations in our case study is described, for each agent, as follows:

- Ready —for an Attacker, this signals that the robot is in a shooting opportunity. For a Supporter, this implies that the robot is able to receive a pass;
- Has Ball Near the Goal —Having possession of the ball near the goal of the opposing team;
- Has Ball in Opponent's Half —Having possession of the ball in the opposing team's half-field;
- Has Ball in Own Half —Having possession of the ball in the half-field belonging to the agent's team;
- Blocked —if the agent is blocked by obstacles;
- Closest —Symbolizes that the agent is closer to the ball than its partner, if neither agent has the ball;
- Second Closest —Symbolizes that the agent is farther from the ball than its partner, if neither agent has the ball.

The preceding signals abstract all of the available information about the environment, resulting in 49 (7×7) possible joint observations. The construction of the associated observation model may be further simplified by exploiting observation independence between agents (Section 3.2.5).

Note that the observation set for each agent does not depend on its specific role as either an Attacker or a Supporter. In some instances (the *Ready* and *Not Ready* signals), it is possible to use the same representations implicitly for both cases, since each agent will take its own observation into context through the observation function.

3.2.3.3 Actions

The remaining component of this model that must be described is the set of joint actions, $\mathcal{A}$. As with the observation set, this set is identical for both agents. For this particular task, our decision-making process following is :

- Dribbling the ball towards the goal;
- Shooting (or kicking) the ball;
- Passing the ball towards the other robot;
- Recovering the ball if it becomes lost;
- Following the attacker;
- Finding a position where a pass can be received (finding clearance for the pass).

Logically, the first four actions described in this manner should be performed by the Attacker robot, while the remaining actions should be taken by the Supporter. Note that this action space is by no means the only possible instantiation that can be used to solve this particular decision-making problem. As it was said above, the selection of these particular actions as the inputs for our DT model establishes the level of abstraction at which we wish to coordinate the team of agents. In this case, our action space is composed of high-level behaviors, while still being sufficiently descriptive to contain the problem of whether to pass the ball between the robots, in the presence of obstacles and under the influence of uncertainty in the outcomes of these behaviors.

Each of these high-level actions is then interpreted by the software of the robotic agents, and triggers a series of more basic behaviors, that may possess its own local decision-making loops. When the robot decides to dribble towards the goal, for example, these lower-level behaviors ensure that the robot is always turned towards the goal, and supply the robot with the necessary controls so that it may drive the ball and try to avoid any imminent obstacles.

3.2.4 Real-Time Execution Strategies

In Chapter 2, we've hinted at the influence of continuous time on the operation of MDP-based physical agents. Evidently, physical systems experience continuous time during

execution, but discrete DT models, such as those we have been describing so far in this chapter, assume that decisions are taken in an episodical, or step-like, fashion, and abstract the temporal duration of each such decision episode. In fact, an issue that is often overlooked in the application of discrete DT methods to real systems is specifying how those step-like decisions should map to continuous time instants – describing not only *how*, but also *when* should a physical system be controlled.

Consider a hypothetical system of non-communicative robots, acting in a real environment according to a Dec-POMDP policy. Since agents are not allowed to exchange observations, and they cannot locally observe all possible changes to the system state, the only way to execute such a fully decentralized policy in real-time, and maintain coherence between the actions of all agents, is to force agents to act at a fixed temporal rate, or at predefined instants. Otherwise, different agents could be in different steps of the decision-making process at any moment in time. We will refer to this fixed-time approach as the *synchronous* execution strategy, since it assumes that all agents are synchronized to a global clock, prior to execution.

Synchronous execution is the most commonly used approach for MDP-based physical agents, even in single agent scenarios, and not limited to robotics applications (Lindqvist, 1962; Little, 1955; Matarić, 1997; Rong and Pedram, 2003). Besides the simplicity of its implementation, the most important quality to this approach is that, for sufficiently small rates, it allows the approximation of state transition probabilities in complex systems as time-invariant measures[1].

However, this approach also carries notable drawbacks. In the general case, there is no clear, systematic methodology to determine an appropriate time-step (the time between decision episodes) for a synchronous system. If this value is too small, *i.e.* decisions are taken too frequently, then the decision problem becomes more complex, both for planning and learning approaches. Agents would then have to select the same decisions repeatedly (think, for example, of a navigation problem where a robot has to repeatedly decide to move in the same direction before moving out of a state), which could actually violate the Markov assumption, since the number of times that the action would need to be re-applied in the same state would influence the probability of any subsequent state transitions. Furthermore, for multiagent systems with communication (such as systems controlled as MMDPs or MPOMDPs), this would imply a higher

[1]Naturally, what constitutes a "sufficiently small" rate depends on the particular system.

communication frequency; on the other hand, if the time-step is too large, it may be longer than the actual duration of a given action. In that case, agents would have to idle for the rest of a decision episode, until they can select a new action. Not only does this mean that tasks can take sub-optimal amounts of time to complete, but it also implies that agents would no longer immediately react to sudden changes in the system state. The latter feature may be important to the long-term outcome of the decision problem, particularly in robotics applications.

Remark 3.2.3. *Synchronous execution of discrete policies implies the loss of reactivity of physical agents to sudden changes in their environment (if the associated time-step is too large), or a higher operational complexity, higher use of communication, and/or loss of the Markovian property (if the time-step is too small).*

An alternative execution strategy, which avoids the aforementioned problems, is to take decisions whenever detectable *events* happen in the system, *i.e.* whenever there is a state transition. We refer to this as an *event-driven* or *asynchronous* execution strategy, since events can happen at random time instants.

In asynchronous execution, since the temporal duration of each decision episode is variable, the sojourn time of the system in a given state can affect its transition probabilities. Therefore, DT models of asynchronous systems must explicitly account for continuous-time. For that reason, applications using asynchronous execution of DT methods are uncommon (notable exceptions include the work of Mahadevan and Khaleeli (1999)). In the multiagent case, the dynamics of DT asynchronous systems become even more complex, and research on the topic has been sparse. In Chapter 5, we will explore the application of the only MDP-based framework, existing prior to this work, which is known to be able to handle asynchronous multiagent systems: the GSMDP framework.

For the moment, and with respect to our robotic soccer case study, we will assume synchronous operation.

3.2.5 Obtaining the Stochastic Models

For model-based solution methods, given a set of states, actions, and observations, it is then necessary to define the transition function T and observation function O[1].

In order to obtain these models in practice, it is necessary to estimate the respective probability distributions by collecting experimental data. For fully observable problems, the methodology behind the estimation of T is straightforward: for each $\langle s, \mathbf{a}, s' \rangle$, $T(s, \mathbf{a}, s')$ can be estimated as the relative frequency of the samples at $\langle s, \mathbf{a} \rangle$ that resulted in s'.

For partially observable problems, however, the problem is more complex, since the state of the system is not directly accessible. The problem is essentially similar to that of estimating the structure of a Hidden Markov Model from observation traces. A principled approach to this problem was reported by Koenig and Simmons (1998), using the Baum-Welch algorithm to estimate the transition and observation models of POMDPs.

One limitation to learning model structure from data, for teams of robots, is that this is typically a time-consuming, iterative process: at least some (sub-optimal) policy must be given to the agents *a priori*, or otherwise, the possible transitions / observations wouldn't be explored. After running the system with such a policy (for example, a completely random policy), and identifying the respective (M)POMDP model, a better policy can be obtained. This policy, in turn, further explores select regions of the system, causing the model to be re-estimated, and altering its respective solution. It is often difficult to operate a team of robots for long enough to collect a significant number of samples to complete this process. This also implies that, at any step of this process, there is also uncertainty over the values of the stochastic model parameters themselves. Some approaches to DT planning explicitly take this source of uncertainty into account (Witwicki et al., 2013), and so they can provide solutions that perform better in practice than alternatives that assume perfect knowledge of the stochastic models.

A common, pragmatical approach to the problem of modeling stochasticity is to simulate the physical system. Many readily available robotics simulators can be used

[1] We emphasize that, in model-free reinforcement learning methods, this step is naturally extraneous.

for that purpose (to name a few, Gazebo[1], V-REP[2], or Webots[3]). In that case, a large number of transition–observation samples can be efficiently collected. Furthermore, since the exact (simulated) state of the system is accessible, the model can be identified as if it were fully observable, greatly simplifying the estimation problem.

Also note that all decision-theoretic frameworks presented so far assume that the stochastic models for the problem are stationary (*i.e.* they are invariant to the steps of the decision-making process). Although, from a theoretical standpoint, this would not constitute a necessity, if that would not be the case, then the problem of obtaining reliable time-varying stochastic models for large scale scenarios would be, in practice, extremely complex.

An important practical issue regarding solution methods for MDPs and related models is that, if knowledge of accurate stochastic models for the environment is assumed, then the sensitivity of the resulting policy to changes in these models is unspecified. Since the solution can potentially be different, it must be calculated anew if any parameters are indeed changed, which may occur, for example, when more reliable data regarding the system becomes available.

Remark 3.2.4. *Decision-theoretic approaches which assume complete knowledge of the stochastic models of the system are inflexible to change. Any modification to these models implies a re-calculation of the associated policy.*

Note that this, in turn, represents a reason to advocate reinforcement learning approaches which do not assume any knowledge (or assume imperfect knowledge (Sutton, 1991)) about the environment.

We will now return to our running example in robotic soccer. In this context, we will also show how it is advantageous to reduce the size of the transition and observation models as much as possible, thereby reducing the number of parameters that should be estimated (and also increasing memory efficiency at run-time).

As discussed in Section 2.4.3, agent-wise factorizations of model components are typically available in situations where the interaction between agents is sparse. Consider

[1] http://gazebosim.org/
[2] http://www.coppeliarobotics.com/
[3] http://www.cyberbotics.com/

the problem of defining T. Ideally, if the local state of an agent was not influenced by the actions of the other agent, there would be an independent transition function for each agent, T_i, such that

$$T(s, \mathbf{a}, s') = T_1(s, a_1, s')T_2(s, a_2, s') \tag{3.1}$$

for every $s, s' \in \mathcal{S}, \langle a_1, a_2 \rangle \in \mathcal{A}$.

However, from the description of the action set for our robotic soccer agents, it is evident that for some of the actions (namely, passing and shooting), this assumption is not valid, since the execution of one of these actions by an agent may induce its partner to switch its role. Nevertheless, it is valid for all of the remaining actions. This constitutes a straightforward instance of context-specific independence.

The problem may still be further simplified by noting its symmetry. Since there is no characteristic feature to distinguish one agent from the other, their transition functions are identical, $T_1 = T_2$. This means that it is only necessary to consider the effects of the four possible independent actions for each agent (which is a considerable reduction from the thirty-six possible joint actions). Also, since only half of the states correspond to a specific agent being Attacker or Supporter, this means that, in matrix form, the transition function for each of the independent actions is block-diagonal (i.e. it is impossible to transition from being an Attacker to a Supporter by applying these actions). For the joint actions that are not conditional independent, their distribution over all of the possible joint states must be obtained.

For the observation model, a similar rationale can be made, but in this case the problem is further simplified by noting the full observation independence in this particular MPOMDP. At first sight, the passing and kicking actions could be understood to also influence the observations of the respective partner robot, but this is indeed not the case, since the observations have been defined independently for each agent in each state, and the actions taken by the partner robot do not influence the ability of each agent to perceive its respective information. The joint actions in this task can then be said to be non-informative, in that they may influence the state of the system, but not the information collected by the agents. This type of problem can also be appropriately termed as Observation Independent (Goldman and Zilberstein, 2004). In practice, this

means that:

$$\Pr(\mathbf{o}|\mathbf{a}, s') = \Pr(o_1|a_1, s')\Pr(o_2|a_2, s') \quad (3.2)$$

$$= O_1(o_1, a_1, s')O_2(o_2, a_2, s') \quad , \forall \mathbf{o} \in \mathcal{O}, \mathbf{a} \in \mathcal{A}, s' \in \mathcal{S} \quad (3.3)$$

It should also be noted that $O_1 = O_2$. Note that this property does not necessarily hold for multi-robot applications. A robot carrying a flashlight in a low-light environment, for example, could influence the observations of other agents in its vicinity. We intend to show, through our soccer robot example, that if this property does hold, it can be exploited for representational simplicity.

After having factored the transition and observation models of our case study, we've proceeded to estimate their values by simulating the system (as it will be further discussed in Section 3.2.7), and sampling action and observation traces.

3.2.6 Defining the Reward Model

As it has been so far made clear, any solution method will attempt to maximize some form of reward. However, with few exceptions, the concept of "reward" for a given task has little semantic value. In the general case, there is no definitive criteria as to how these rewards should be assigned. Even if there are well-defined, quantitative performance measures for a real system, such as least expended energy, shortest traveled distance, or shortest time for operation, these can only be loosely translated onto the reward structure of a decision-theoretic model. Furthermore, since discretized models can only approximate the actual dynamics of a physical system, the notion of "optimality" of a policy over such a model, with respect to the expected cumulated reward, does not imply optimal performance of the controlled physical system.

There is often a process of *trial-and-error* until a reward model is obtained which induces a policy with satisfactory practical quality. This is valid both for planning and reinforcement learning methods. However, model-based solutions suffer from the same problem of inflexibility with respect to reward as they do with stochastic models. This, coupled with the fact that the reward model may be re-defined multiple times during design, results in a overly lengthy process of planning.

Remark 3.2.5. *Model-based solution methods are inflexible to changes in the reward structure.*

It is here argued that a solution method oriented towards real-world applications should have some measure of adaptability, similar to the approach proposed by Sutton (1991), to account for possible changes in the definitions of the various models, without necessarily inducing a complete re-calculation of the respective solution.

In our robotic soccer case-study, the definition of the reward model was carried out by assigning a high reward for kicking the ball in a shooting opportunity, and penalizing every other step taken, in order to promote the fastest possible solutions.

3.2.7 Implementation and Results of the Robotic Soccer Case-Study

We will now discuss the steps that were taken in the practical implementation of our robotic soccer case-study in realistic simulation. We will also analyze the results of this case-study in practice.

3.2.7.1 Communication

Since we've opted to model this problem as an MPOMDP, we've assumed that agents can communicate their observations freely to each other. However, in practice, this communication must be managed explicitly, and some mechanism must be implemented that exchanges and synchronizes the observations of each agent. Although our implementation is here simulated, the communications between our agents were handled as if they *were* running on different robotic platforms.

The synchronization process and necessary explicit communication is performed according to the diagram in Figure 3.4. Since this is a MPOMDP, each agent has access to all observations, and it computes, locally, the maximizing joint action for the team at each step, from where it extracts its own local action. Each agent is assumed to execute that action for a fixed time step, τ. The observations of the agents at the next time step are only available after the outcome of that action is known. Therefore, after τ has passed, the agents sample their own observations and exchange it with that of their partner. If one of the agents is delayed, then its partner will wait for this information before proceeding. This step is where synchronization is enforced between both robots. This information is then used to locally calculate the joint belief through (2.21). After

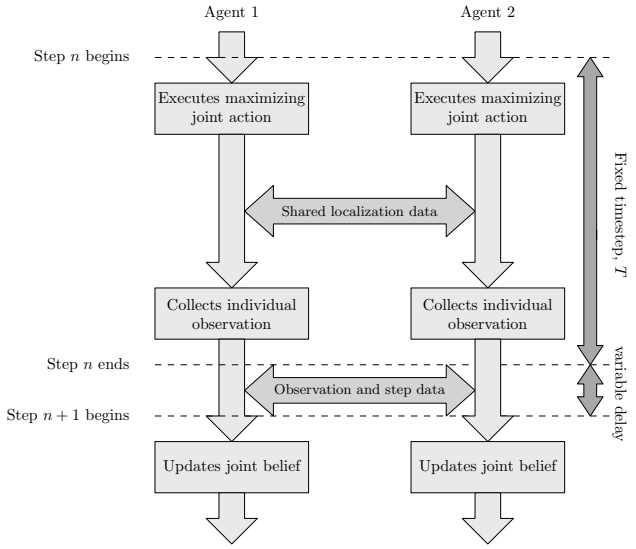


Figure 3.4: Synchronization timeline with instantaneous communication.

the joint belief is obtained, each agent computes a new maximizing joint action and proceeds to the next step.

Here, the imposition of synchronous decision-making shows once more a potential practical disadvantage. For large teams of agents, it might not be feasible to wait for synchronization, or to assume perfectly reliable communications at each step. This is often the case in robotic soccer, since the high number of agents quickly saturate the communication medium. In such a setting, it can be advantageous not to wait for synchronization between all involved agents. In this sense, the agent would select an optimal action based on its own local observation, and receive its partners' data throughout the decision step (Figure 3.5). We will analyze the influence that the latter approach to decision-making has on the performance of the system.

3.2.7.2 Solving the MPOMDP

The PERSEUS algorithm (Spaan and Vlassis, 2005) was selected to solve the MPOMDP, due to its efficiency in handling moderately-sized problems. PERSEUS belongs to the family of point-based POMDP solvers, but it is by no means the only one (Kurniawati et al., 2008; Pineau et al., 2003). While it is true that often the algorithm to solve a given POMDP model should be chosen according to the problem's structure, this does not create, in this case, a dependency on any particular algorithm.

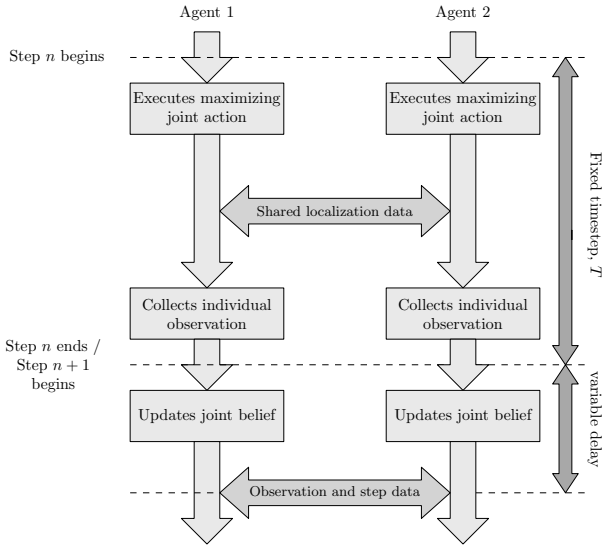


Figure 3.5: Synchronization timeline with delayed communication of observations.

The Perseus algorithm performed favorably, and converged in as few as 140 iterations to a residual value of 10^{-4}, as can be seen in Figure 3.6. Naturally, such a value function is a good approximation of a stationary solution, i.e., a solution that assumes an infinite horizon.

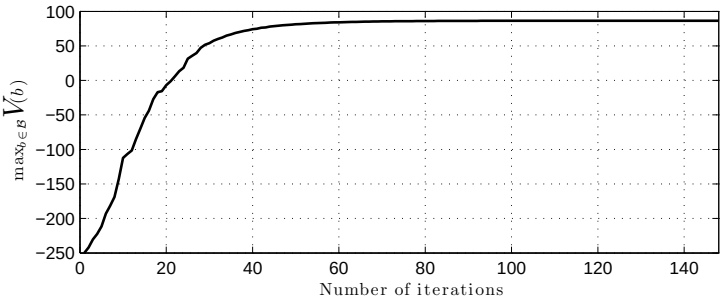


Figure 3.6: Convergence of the Perseus algorithm for the proposed MPOMDP.

3.2.7.3 Experimental Setup

The execution of the task itself was carried out using the Webots robotics simulator. The physical properties of our robotic agents were thoroughly modeled, including mass, friction, and actuator constraints. The two agents were placed in arbitrary initial positions in the field of play, and the ball was initially placed in the center of the field,

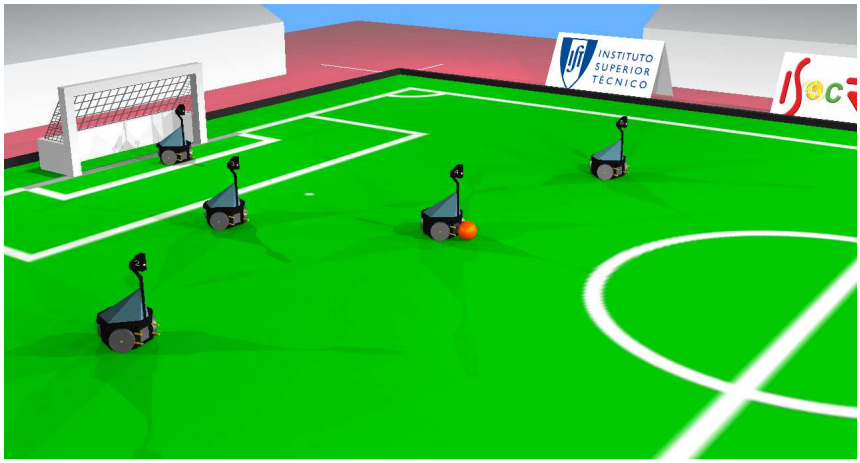


Figure 3.7: The simulated environment where our case-study was deployed.

which is the origin of the world frame for the robots. The time-step of the synchronous decision-making loop was set to 3 seconds. The transition model allowed a considerable (~ 0.4) chance of performing a transition to a neighbor state when dribbling the ball (estimated empirically). The observation model for these robots considered both the possibility of having false positive and false negative detections of obstacles: a 0.1 probability of failing to detect an obstacle and a 0.05 probability of detecting an inexistent obstacle. The immediate reward for scoring a goal was set to 150. For successfully completing a pass whenever the Attacker robot became blocked, the team received a reward of 60. Agents were also penalized for performing actions inconsistently with their role, with rewards of -20 (*e.g.* a supporter trying to dribble the ball, or an attacker trying to follow its partner). All other actions were associated with a -1 reward. The discount factor was set to 0.9.

3.2.7.4 Results

A video of the execution of this policy in our realistic simulator is made available at at `http://users.isr.ist.utl.pt/~jmessias/PhDthesis`, the online repository for the auxiliary files to this work. The MPOMDP model that was used in our experiments can be also be found in that repository, in a file format that is compatible with the Multiagent Decision Process (MADP) Toolbox (Spaan and Oliehoek, 2008).

The total discounted reward at run-time, using the value function obtained through

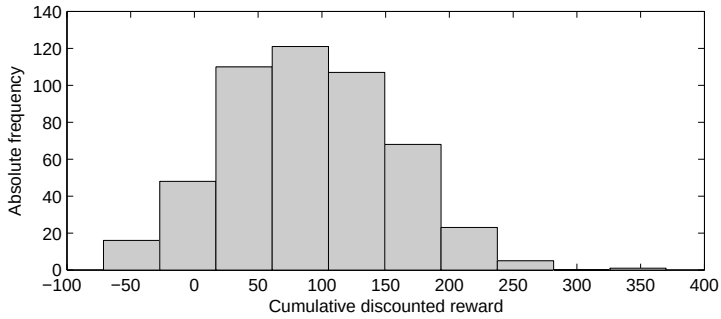


Figure 3.8: A histogram of accrued discounted reward for 500 simulated runs of the proposed task.

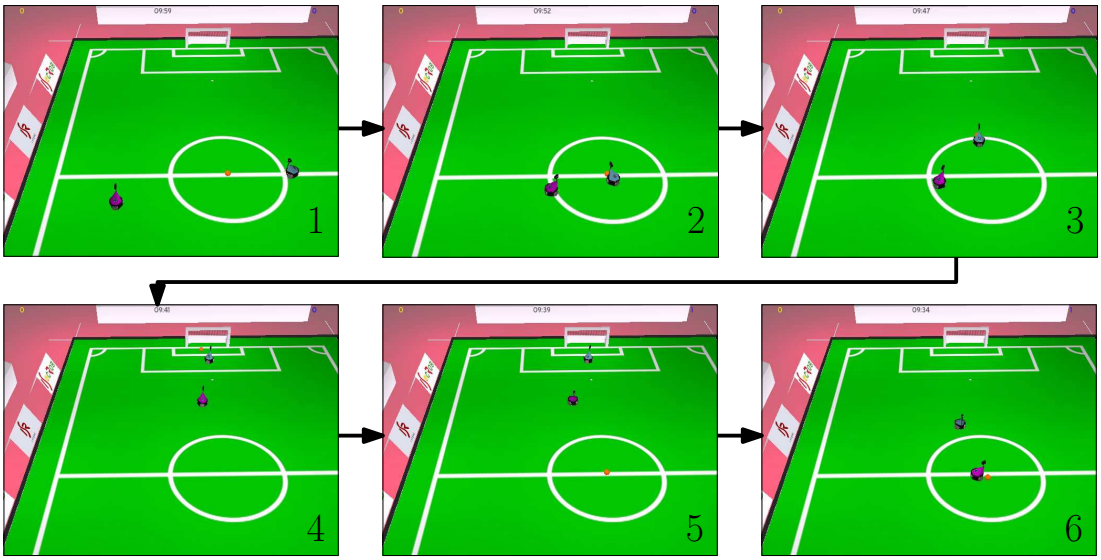


Figure 3.9: Behavior of the robots when no obstacles are present in the field.

PERSEUS for 500 simulated runs, was 90.14 ± 65.38. A histogram of this data is represented in Figure 3.8.

We will now describe the observed behavior of our team of robots in two different situations that showcase their performance.

In Figure 3.9, we represent, as a sequence of frames captured from our simulator during execution, a timeline of a situation where robots were allowed to dribble freely to the goal, without being hindered by obstacles. Both robots were covered in different colors (cyan or magenta), so that they can be visually identified throughout the sequence. In initial state (frame 1), the robot which was closest to the ball assumed the

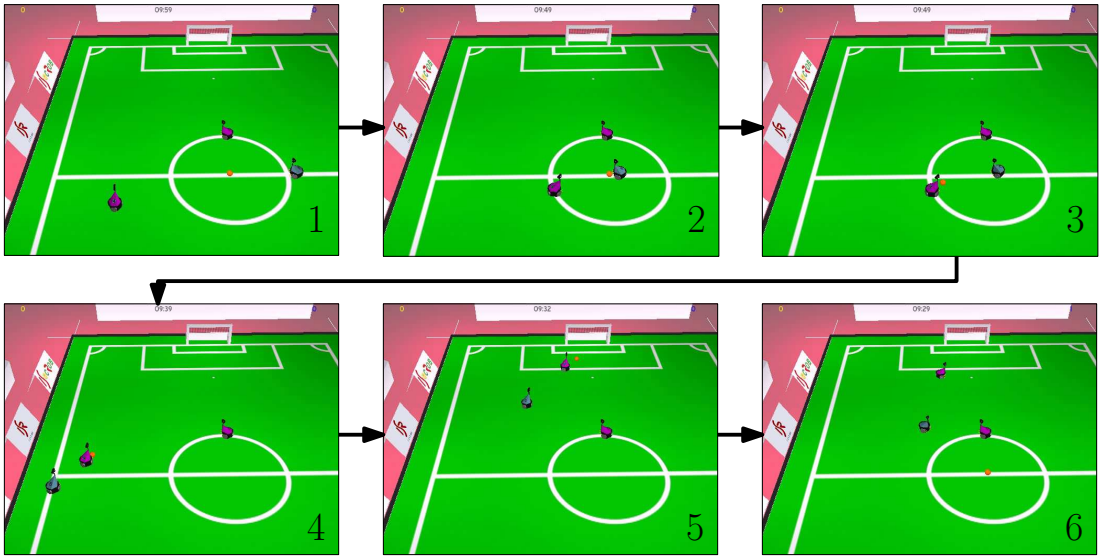


Figure 3.10: Behavior of the robots when passing the ball to avoid obstacles.

role of Attacker (the cyan robot), while its partner maintained a fixed distance. The Attacker robot proceeded to score a goal (frame 4). The ball was then reset by the simulator back to the $(0, 0)$ coordinate. Since the initial Supporter is now the closest robot to the ball, it assumes the role of Attacker, and in frame 6 their roles have been exchanged from their initial configurations. The process then repeats itself.

A second situation occurs in the presence of obstacles, and is depicted in Figure 3.10. An obstacle is placed in front of the initial Attacker. In frames $2-3$ it decides to pass the ball to its partner. Their roles then switch, and the magenta robot carries the ball until it scores a goal in frame 5. Note that, in this situation, the blocking robot (the obstacle) was stationary. In the cases where this wasn't true, the passing behavior would fail randomly, due to the fact that the Attacker agent would be committed to performing the same joint action (that is, dribbling) until the predefined time-step expired; by that time, however, the robot could collide with the moving obstacle and lose the ball. This exemplifies the problem of assuming a constant temporal duration for each action. In dynamic environments, the robot may not have enough time to select the optimal action when facing a sudden change in its state.

Another issue with this implementation is its indiscriminate use of communication. The robots must communicate synchronously every 3 seconds in order to maintain coherency in their actions, and it is assumed that all communication episodes are free

from failures or delays. If this wasn't the case, the representation of the joint belief that is maintained by each agent would acumulate irrecoverable errors, and the execution of the task would be compromised.

3.3 Summary

This chapter presented a brief review of the application areas of the MDP framework and its associated extensions.

With the goal of exploring the potential limitations of these frameworks when applied to cooperative robotics scenarios, we presented a "walkthrough" of the steps involved in modeling a decision-making problem with physical agents through an MDP-based framework. A conceptual, albeit realistic, small-scale task in that domain was introduced, serving as proof-of-concept. The following topics were perceived as the main shortcomings of decision-theoretic models when dealing with cooperative robotics scenarios:

- Multiagent frameworks with explicit communication models, or in which no communication is possible, are not generally applicable to real-world problems due to their intractability. Approaches that assume free communication are susceptible to considerable loss of quality in the event of communication delays or failures;

- States, actions, and observations in robotics domains are naturally defined over continuous variables. When modeling these elements in a discrete DT framework, the abstractions involved in that discretization are selected based on a trade-off between operational complexity and system performance. Discrete DT approximations can affect the Markovian property of the system;

- The synchronous execution of discrete multiagent policies in real time carries several negative drawbacks. Most notably, the loss of reactivity in dynamically changing environments, excessive use of communication, and / or an unnecessary increase in the horizon of the decision-making problem;

- Model based methods are inflexible to changes in any of their parameters. This means that a new solution must be calculated in the event that new information regarding the system is made available, or if the reward structure of the model

is tuned. Modeling a large-scale DT problem is a lengthy, time-consuming procedure, and this fact delays the deployment of these techniques in real environments.

The simulations that were performed in our case study show that, even though our team of robots could complete the proposed cooperative task in some conditions, its performance was evidently sub-optimal, given the idle time experienced by the robots between decisions, which in turn induces a lack of reactivity to urgent occurrences that impact the outcome of the task; its susceptibility to communication failures and delays; and its reliance on perfect knowledge of the transition and observation model parameters.

The remainder of this work will focus on addressing the issues which were here identified, with the goal of applying DT methods to a full scale scenario in cooperative robotics, motivated as a real-world application. We will also provide general guidelines and tools for the implementation of these frameworks in other realistic scenarios.

Chapter 4

Efficient Communication in Partially Observable Domains

As it has been previously mentioned, the computational complexity of decision-theoretic planning in multiagent systems is strongly influenced by the type of communication which is available in a given scenario. Most non-communicative problems, falling in the general Dec-POMDP class, are NEXP-hard to solve for finite horizons[1] (Pynadath and Tambe, 2002). Adding costs to communication actions, and requiring an explicit communication policy to be obtained offline, does not reduce this complexity, since non-communicative policies must also be evaluated in the worst case. However, if agents are able to share their observations at each step, without any associated cost, then the complexity of the multiagent problem is the same as that of a centralized, team-wise single agent model.

In many situations, however, communication is not free. A large team of agents may need to control their communication usage in order to avoid saturating the communication medium. Communicating may also expend valuable energy in scenarios where agent autonomy is the most important characteristic (such as space exploration). In military applications, if other entities are attempting to intercept inter-agent communications, then it also follows that communication should be kept to a minimum.

DT approaches in which communication between agents is possible have already been explored for non-factored Dec-POMDP models (Pynadath and Tambe, 2002; Roth et al.,

[1]The only exceptions are unrealistic scenarios in which the environment is either individually fully observable, or non-observable at all, by each agent, and in which communication makes no difference.

2005a; Spaan et al., 2008; Wu et al., 2009) as well as for factored Dec-MDPs (Roth et al., 2007). The work described in this chapter will focus on factored MPOMDP models. We propose a novel method that exploits sparse dependencies in such a model, in order to reduce the amount of inter-agent communication. To achieve this, we consider a *factored* joint belief state. We obtain individual policies for each agent that map beliefs over state factors to actions or communication decisions.

Maintaining an exact, factored belief state is typically not possible in cooperative problems. While bounded approximations are possible for probabilistic inference (Boyen and Koller, 1998), these results do not carry over directly to decision-making settings (but see McAllester and Singh (1999)). Intuitively, even a small difference in belief can lead to a different action being taken. However, when there are sparse dependencies between the actions of the agents, often the belief over its local state factors is sufficient for an agent to identify the action that it should take, and communication can be avoided. We formalize these notions as convex optimization problems, extracting those situations in which communication is superfluous. We present experimental results showing the savings in communication that can be obtained, and the overall impact on decision quality.

This chapter is organized as follows. Sections 4.1, 4.2, 4.3 introduce the relevant motivation and background for the current work. Section 4.4 presents the formalization of the proposed method to associate belief points over state factors to actions. Next, Section 4.5 illustrates the concepts with experimental results, and Section 4.6 provides a closing discussion.

4.1 Exploiting Sparse Dependencies in MPOMDPs

In the implementation of Multiagent POMDPs, an important practical issue is raised: since the joint policy arising from the value function maps joint beliefs to joint actions, all agents must maintain and update the joint belief equivalently for their decisions to remain consistent. The amount of communication required to make this possible can then become problematically large. Here, we will consider a fully-communicative team of agents, for planning purposes, but we will minimize the communication that is required to execute a given plan. Even if agents can communicate with each other freely, they might not need to always do so in order to act independently, or even cooperatively.

The problem of when and what to communicate has been studied before for Dec-MDPs (Roth et al., 2007), where *part* of the factored state space can be directly observed by each agent, with no associated uncertainty. In this case, the rationale is that the local information available to each agent typically maps to a *set* of possible local actions (whereas in an MDP it would map directly to a single action). By requesting more information regarding the system state from other agents, an agent can disambiguate his possible actions. For MPOMDPs, a similar methodology had been introduced, which operated at runtime. This means that the communication decisions could not be known *prior* to execution; and it implied keeping track and reasoning over a rapidly-growing number of possible joint belief points (Roth et al., 2005a).

We will here describe a method that maps a belief factor (or several factors) directly to a local action, or to a communication decision, when applicable. The proposed approach is the first to exploit, offline, the structure of the value function itself in order to identify regions of belief space where an agent may act independently. This raises the possibility of developing more flexible forms for joint policies which can be efficiently decoupled whenever this is advantageous in terms of communication. Furthermore, since this method runs offline, it is not mutually exclusive with online communication-reduction techniques: it can be used as a basis for further computations at runtime, thereby increasing their efficiency.

4.2 Decision-Making with Factored Beliefs

Recall from Section 2.3.1 that a *joint* belief state is a probability distribution over the set of states $\mathcal{S}$, and encodes all of the information gathered by all agents in the MPOMDP up to a given step n. We consider a factored state model (see Section 2.4.3) with k state factors $\mathcal{X}_i$. Let $\boldsymbol{\theta}_n$ represent the entire execution history of the multiagent system (its actions and observations) up to step n, that is, $\boldsymbol{\theta}_n = \langle b_0, \mathbf{o}_1, \mathbf{a}_1, \mathbf{o}_2, \mathbf{a}_2, \ldots, \mathbf{o}_{n-1}, \mathbf{a}_{n-1} \rangle$. The joint belief is then:

$$\begin{aligned} b_n(s) &= \Pr(s_n \,|\, \boldsymbol{\theta}_n) \\ &= \Pr(x_{1,n}, \ldots, x_{k,n} \,|\, \boldsymbol{\theta}_n) \quad , \end{aligned} \tag{4.1}$$

where $x_{i,n}$ is the value of $\mathcal{X}_i$ at step n. A factored belief state is a representation of this very same joint belief as the product of F assumed independent belief states, which will

be referred to as *belief factors*. Each belief factor $b_{\mathcal{Y}_i,n}$ is defined over a subset $\mathcal{Y}_i \subseteq \mathcal{X}$ of state factors. Then, for $\mathbf{y}_{i,n} \in \Pi_{Y \in \mathcal{Y}_i} Y$ at step n:

$$b_n(s) \simeq \Pr(\mathbf{y}_{1,n} \,|\, \boldsymbol{\theta}_n)\Pr(\mathbf{y}_{2,n} \,|\, \boldsymbol{\theta}_n) \cdots \Pr(\mathbf{y}_{F,n} \,|\, \boldsymbol{\theta}_n) \tag{4.2}$$

$$\simeq b_{\mathcal{Y}_1,n}(\mathbf{y}_{1,n}) b_{\mathcal{Y}_2,n}(\mathbf{y}_{2,n}) \cdots b_{\mathcal{Y}_{F,n}}(\mathbf{y}_{F,n}) \tag{4.3}$$

We also impose that these subsets of state factors are disjoint, *i.e.* $\mathcal{Y}_i \cap \mathcal{Y}_j = \emptyset \,, \forall i \neq j$. A belief point over factors $L \subseteq \mathcal{X}$ which are locally available to an agent is denoted b_L.

Let $\text{neg}(\mathcal{Y}) = \{\tilde{\mathbf{y}} : \tilde{\mathbf{y}} \in \Pi_{Y \in \mathcal{X} \setminus \mathcal{Y}} Y\}$ be the set of all possible assignments to all state factors which are *not* in $\mathcal{Y}$. The marginalization of b onto $b_\mathcal{Y}$ is then, for $\mathbf{y} \in \mathcal{Y}$:

$$\begin{aligned} b_{\mathcal{Y},n}(\mathbf{y}) &= \Pr\left(\mathbf{y} \,|\, \boldsymbol{\theta}_n\right) \\ &= \sum_{\tilde{\mathbf{y}} \in \text{neg}(\mathcal{Y})} \Pr\left(\tilde{\mathbf{y}}, \mathbf{y}_n | \boldsymbol{\theta}_n\right) \quad . \end{aligned} \tag{4.4}$$

Alternatively, this can be viewed as a projection of b onto the smaller subspace $\mathcal{B}_\mathcal{Y}$:

$$b_\mathcal{Y} = M_\mathcal{Y}^\mathcal{X} b \tag{4.5}$$

where $M_\mathcal{Y}^\mathcal{X}$ is a matrix where $M_\mathcal{Y}^\mathcal{X}(u, v) = 1$ if the assignments to all state factor variables contained in the u-th element of $\Pi_{Y \in \mathcal{Y}} Y$ are the same as in the v-th element of $\Pi_{X \in \mathcal{X}} X$ (or, equivalently, the v-th joint state), and 0 otherwise. This intuitively carries out the marginalization of points in $\mathcal{B}$ onto $\mathcal{B}_\mathcal{Y}$. An example of such a "marginalization matrix" is shown in Figure 4.2 for an illustrative scenario that will be introduced in the following section.

Note that, as fully described in Boyen and Koller (1998), the factorization (4.2) typically results in an approximation of the true joint belief, since it is seldom possible to decouple the dynamics of an MDP into strictly independent subprocesses. The dependencies between factors, induced by the transition and observation model of the joint process, quickly develop correlations when the horizon of the decision problem is increased, even if these dependencies are sparse. A significant result of Boyen and Koller (1998), on which our method depends, is that, if some of these dependencies are broken, the resulting error (measured as the KL-divergence) of the factored belief state, with respect to the true joint belief, is bounded.

We must also take into account that a small error in the belief state can lead to different actions being selected, which may significantly affect the decision quality of the multiagent team in some settings (McAllester and Singh, 1999; Poupart and Boutilier, 2000). Therefore, it should be expected that, when using factored belief states, the run-time performance of the agent team will be lower than when using the exact, joint representation. In rapidly-mixing processes (i.e., models with transition functions which quickly propagate uncertainty), the overall negative effect of using this approximation is minimized.

Each belief factor's dynamics can be described using a two-stage Dynamic Bayesian Network (DBN). Note that the *exact* propagation of a belief factor across decision steps (*i.e.* updating it after an action and observation) would require the marginalization of the full joint belief state at all steps, in general. This is the case, for example, when using the *Junction Tree* belief propagation algorithm (Lauritzen and Spiegelhalter, 1988). This would deny any saving in terms of communication during execution, since agents would always need to communicate just to maintain their local belief factors up-to-date. However, when using *approximate* propagation algorithms, such as *Factored Frontier* (Murphy and Weiss, 2001), it is possible to maintain and update fully factored belief states without requiring the marginalization of the joint belief. Instead, only the direct dependencies of the locally acessible state factors (their parents in the 2-DBN) are required in order to update the respective local belief factors. If locally accessible state factors do not have any dependencies on *non*-local state factors (*e.g.* in a Transition-Independent process), then the update process can be carried out using only the local observations of the agent, and without requiring any communication. Otherwise, the necessary information should be requested from other agents, and can be determined by using the aformentioned propagation algorithms. The amount of data to be communicated in this latter case, as well as its frequency, depends largely on the factorization scheme that is selected for a particular problem.

The problem of obtaining a suitable partition scheme of the joint belief onto its factors will not be addressed in this work. Such a partitioning is typically simple to identify for multi-agent teams which exhibit sparsity of interaction. Instead, **we focus on the amount of communication that is necessary for the joint decision-making of the multi-agent team**. To that end, we will remain abstracted, throughout the rest of this chapter, from the problem of propagating belief factors at run-time. It should

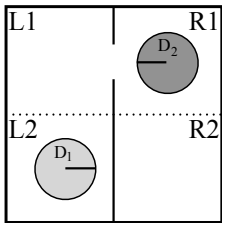


Figure 4.1: The *Relay-Small* problem.

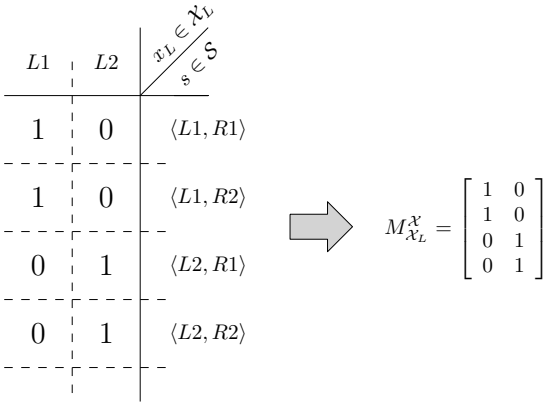


Figure 4.2: Defining the marginalization matrix $M^{\mathcal{X}}_{\mathcal{X}_L}$ for the *Relay-Small* problem. This matrix carries out the marginalization of the joint belief onto the belief factor $b_{\mathcal{X}_L}$.

be expected that, in some situations in which we regard communication as unnecessary in order to determine the actions of an agent, communication may still be needed at run-time in order to update local belief factors. Conversely, in some problems where communication is not needed in order to propagate local belief states (for example, in ND-POMDPs), it may still be necessary to identify the action that each agent should take.

4.3 An illustrative example: the *Relay-Small* problem

Consider the following small-scale factored MPOMDP, named *Relay-Small*, which will be used as a conceptual example. In this environment, two agents operate inside a four-state world, represented in Figure 4.1, in which each agent is confined to a two-state area. One of the agents possesses a package which it must hand over to the other agent. The goal of these agents is then to "relay" this package between each other through the opening between the rooms L1 and R1. The actions that each agent can perform are

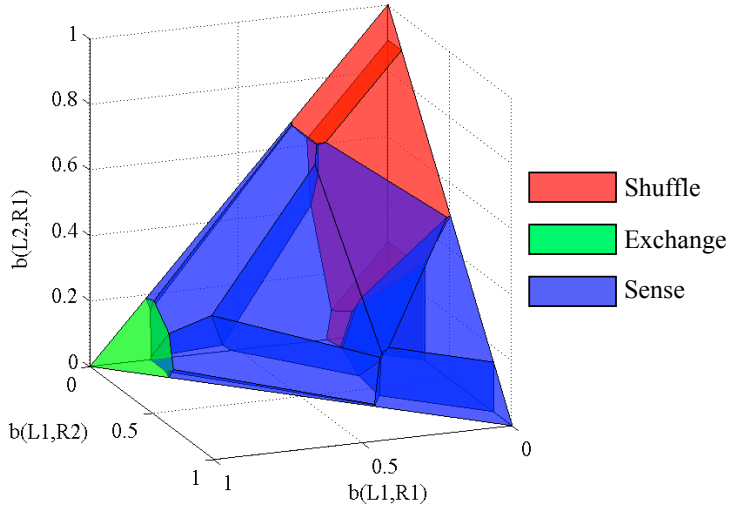


Figure 4.3: The linear supports of the optimal stationary joint value function for the *Relay-Small* problem, colored according to the respective local actions of agent 1.

to *Shuffle*, *Exchange*, or *Sense*. A *Shuffle* action moves the agent randomly, and with equal probability, to either position in its local area. The *Exchange* action attempts to perform the physical exchange of the package between the agents, and is only successful if both agents are in the correct position (L1 for the first agent, R1 for the second one) and if both agents perform this action at the same time. If it succeeds, the world is reset to a random state with uniform probability. The *Sense* action is an informative action, which allows the agent to sense whether it is in front of the opening or not, with probability of both false positives and false negatives (0.1 probability of incorrectly detecting an opening or lack thereof). The interesting feature of this small problem is its sparse dependency between the decision processes of these agents. Evidently, the only cooperative action that the agents may perform is a joint *Exchange*. Since this action can only succeed in a particular joint state, it stands to reason that an agent which is sufficiently certain of not being in that particular, corresponding local state should always attempt to move there first (via *Shuffle*). In such a case, this decision can be taken regardless of the other agent's state, actions or observations (since the agents cannot observe each other).

Figure 4.3 represents the linear supports of the optimal infinite-horizon value function for this example, and the associated local actions of the first agent. Note that the joint belief is four dimensional, but one of these dimensions may be omitted due to the

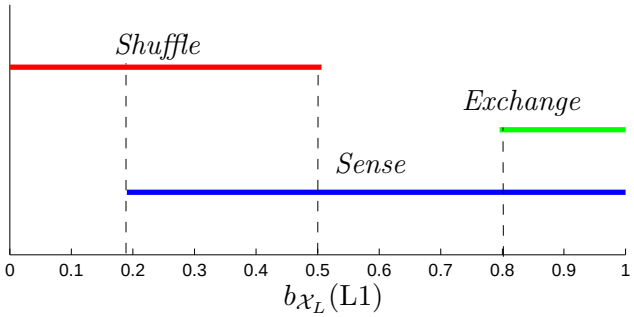


Figure 4.4: The projections of the linear supports of the joint value function onto the belief factor over the left room.

fact that $\sum_{s \in \mathcal{S}} b(s) = 1$. The expected behavior can be inferred from this representation, noting that the Exchange action should only be performed if the agents are certain of being both in the correct position ($b_{L1,R1}$ is high), and the Shuffle action should only be performed if the agent is sufficiently certain of *not* being in the correct position.

In this scenario, an evident space state factorization is $\mathcal{S} = \mathcal{X}_L \times \mathcal{X}_R$, where $\mathcal{X}_L = \{L1, L2\}$ and $\mathcal{X}_R = \{R1, R2\}$. The projection of these linear supports onto the smaller belief factor $b_{\mathcal{X}_L}$ is shown in Figure 4.4. Here, an interesting result is already evident: some regions of this belief space are covered only by a single linear support. In these situations, there is no ambiguity as to what agent 1 should do. Its expected behavior is here still contained: if the agent has low probability of being in room L1, then the optimal action is to Shuffle; if it is not certain enough of being in L1, it should Sense; and Exchange is always an ambiguous action, since it depends on factor $\mathcal{X}_R$.

The key idea in the proposed approach is, that in some situations, the local information of an agent is enough for it to take locally optimal decisions. If, furthermore, the belief states over the local state factors are maintained independently, then the agents might not need to communicate at all between two decisions. The explicit need to communicate would only arise in situations where one agent's optimal action is dependent upon the other agent's information. In this example, this corresponds to the case where one agent is fairly certain of being in the correct place for the exchange. It then needs to reason over the other agent's local belief to make sure that an Exchange action is profitable in terms of expected reward.

4.4 Formal model

The value function, and its associated quantities, are considered to be associated with a given decision step n, and, for simplicity, this dependency will be omitted. However, it should be noted that the value function does not need to be stationary – for a finite-horizon problem, the following methods can simply be applied for every $n = 1, \ldots, h$.

4.4.1 Value Bounds Over Local Belief Space

Recall from Section 2.3.1 that, for a given vector α belonging to a PWLC value function, $\nu(\alpha)$ is the action associated with that vector. Let $V_\alpha(b) = \alpha \cdot b$ represent the expected reward for selecting action $\nu(\alpha)$. Ideally, if the value associated with an action could be mapped from a local belief point b_L, then it would be possible to select the best action for an agent based only on its local information. This is typically not possible since the projection (4.5) is non-invertible. However, as it will be shown, it is possible to obtain bounds on the achievable value of any given vector, in local belief space.

The available information regarding $V_\alpha(b)$ in local space can be expressed in the following system of linear equations:

$$\begin{aligned} V_\alpha(b) &= \alpha \cdot b \\ \mathbf{1}_n^{\mathrm{T}} b &= 1 \\ M_L^{\mathcal{X}} b &= b_{\mathcal{L}} \end{aligned} \tag{4.6}$$

where $\mathbf{1}_n = \begin{bmatrix} 1 & 1 & \ldots & 1 \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^n$. Let m be the size of the local belief factor which contains b_L. Reducing the matrix corresponding to this system of linear equations, it is possible to associate $V_\alpha(b)$ with b and $b_{\mathcal{L}}$. The reduced matrix has at least $n - m$ free variables in its leading row, induced by the locally unavailable dimensions of b. The resulting equation of the reduced leading row can be rewritten as:

$$V_\alpha(b) = \beta \cdot b + \gamma \cdot b_L + \delta \quad , \tag{4.7}$$

with $\beta \in \mathbb{R}^n$, $\gamma \in \mathbb{R}^m$ and $\delta \in \mathbb{R}$. By maximizing (or minimizing) the terms associated with the potentially free variables, this form can be used to establish the maximum (and minimum) value that can be attained at b_L.

Theorem 1. *For a set of state factors $L \subseteq \mathcal{X}$, let $\mathcal{I}_u = \{v : M_L^{\mathcal{X}}(u,v) = 1\}$, $\overline{\beta} \in \mathbb{R}^m : \overline{\beta}_i = \max_{j \in \mathcal{I}_i} \beta_j, i = 1, \ldots, m$ and $\underline{\beta} \in \mathbb{R}^m : \underline{\beta_i} = \min_{j \in \mathcal{I}_i} \beta_j, i = 1, \ldots, m$. The maximum achievable value for a local belief point, b_L, according to α, is:*

$$\overline{V_\alpha}(b_L) = \left(\overline{\beta} + \gamma\right) \cdot b_L + \delta \quad . \tag{4.8}$$

Analogously, the minimum achievable value is

$$\underline{V_\alpha}(b_L) = \left(\underline{\beta} + \gamma\right) \cdot b_L + \delta \quad . \tag{4.9}$$

Proof. First, it shall be established that $\overline{V_\alpha}(b_L)$ is an upper bound on $V_\alpha(b)$. The set $\mathcal{I}_i$ contains the indices of the elements of b which marginalize onto $[b_L]_i$, and $\overline{\beta}_i$ is the maximum value of β for any of these indices. From the definition of $\overline{\beta}$ it follows that, for all belief points b in the belief space $\mathcal{B}$:

$$\begin{aligned} \sum_{j \in \mathcal{I}_i} \overline{\beta}_i b_j \geq \sum_{j \in \mathcal{I}_i} \beta_j b_j \quad , i = 1, \ldots, m \quad \Leftrightarrow \\ \Leftrightarrow \quad \overline{\beta}_i [b_L]_i \geq \sum_{j \in \mathcal{I}_i} \beta_j b_j \quad , i = 1, \ldots, m \quad , \end{aligned}$$

where the fact that $\sum_{j \in \mathcal{I}_i} b_j = [b_L]_i$ was used (marginalization). Summing over all i, this implies that $\overline{\beta} \cdot b_L \geq \beta \cdot b$. Using (4.7) and (4.8),

$$\overline{\beta} \cdot b_L + \gamma \cdot b_L + \delta \geq \beta \cdot b + \gamma \cdot b_L + \delta \quad \Leftrightarrow \quad \overline{V_\alpha}(b_L) \geq V_\alpha(b)$$

Next, it needs to be shown that $\exists b \in \mathcal{B} : \overline{V_\alpha}(b_L) = V_\alpha(b)$. Since $\mathbf{1}_n^{\mathrm{T}} b = 1$ and $b_i \geq 0 \; \forall i$, $\beta \cdot b$ is a convex combination of the elements in β. Consequently, $\max_{b \in \mathcal{B}} \beta \cdot b = \max_i \beta_i$, since a point b can always be taken such that $[b]_{\arg\max_i \beta} = 1$, that is, it is equal to 1 for the same index as the maximum element of β. Then, following the same rationale,

$$\max_i \beta_i = \max_{b \in \mathcal{B}} \overline{\beta} \cdot M_L^{\mathcal{X}} b \quad .$$

Therefore, for $b_m = \arg\max_{b \in \mathcal{B}} \beta \cdot b$, it results that $\overline{V_\alpha}(M_L^{\mathcal{X}} b_m) = V_\alpha(b_m)$.

The proof for the minimum achievable value $\underline{V_\alpha}(b_L)$ is analogous. □

By obtaining the bounds (4.8) and (4.9), a step has been taken towards identifying

the correct action for an agent to take, based on the local information contained in b_L. From their evaluation, the following remarks can be made: if α and α' are such that $\overline{V_{\alpha'}}(b_L) \leq \underline{V_{\alpha}}(b_L)$, then α' is surely not the maximizing vector at b; if this property holds for all α' such that $(\nu(\alpha'))_i \neq (\nu(\alpha))_i$, then by following the action associated with α, agent i will accrue at least as much value as with any other vector for all possible b subject to (4.5). That action can be safely selected without needing to communicate.

The complexity of obtaining the local value bounds for a given value function is basically that of reducing the system (4.6) for each vector. This is typically achieved through Gaussian Elimination, with an associated complexity of $O(n(m+2)^2)$ (Fang and Havas, 1997). Note that the dominant term corresponds to the size of the local belief factor, which is usually exponentially smaller than n. This is repeated for all vectors, and if pruning is then done over the resulting set (the respective cost is $O(|\Gamma|^2)$), the total complexity is $O(|\Gamma|n(m+2)^2 + |\Gamma|^2)$. The pruning process used here is the same as what is typically done by POMDP solvers (White, 1991).

4.4.2 Dealing With Locally Ambiguous Actions

The definition of the value bounds (4.8) and (4.9) only allows an agent to act in atypical situations in which an action is clearly dominant in terms of expected value. However, this is often not the case, particularly when considering a large decision horizon, since the present effects of any given action on the overall expected reward are typically not pronounced enough for these considerations to be practical. In a situation where multiple value bounds are conflicting (i.e. $\overline{V_{\alpha}}(b_L) > \underline{V_{\alpha'}}(b_L)$ and $\underline{V_{\alpha}}(b_L) < \overline{V_{\alpha'}}(b_L)$), an agent is forced to further reason about which of those actions is best.

In order to tackle this problem, assume that two actions a and a' have conflicting bounds at b_L. Then, given $\Gamma^a = \{\alpha \in \Gamma : (\nu(\alpha))_i = a\}$ and similarly defined $\Gamma^{a'}$, the matrices $A = [\Gamma_i^a]_{k \times n}, \quad i = 1, \ldots, |\Gamma^a|$ and $A' = [\Gamma_i^{a'}]_{k' \times n}, \quad i = 1, \ldots, |\Gamma^{a'}|$are defined. Then, the vectors $\mathbf{v} = Ab$ and $\mathbf{v}' = A'b$ (in $\mathbb{R}^k$ and $\mathbb{R}^{k'}$ respectively) contain all possible values attainable at b through the vectors in Γ^a and $\Gamma^{a'}$. Naturally, the maximum of these values for each action will be sought. In particular, the goal is to determine if $\max_i \mathbf{v}_i$ is greater than $\max_j \mathbf{v}'_j$ for all possible b such that $b_L = M_L^{\mathcal{X}} b$. If this is the case, then a should be selected as the best action, since it is guaranteed to provide a higher value at b_L than a'.

The problem is then to find some b where a' is the maximal action. It can be described as the constrained optimization:

$$\begin{array}{lll} \text{minimize} & \max_i \mathbf{v}_i - \max_j \mathbf{v}'_j & \\ \text{subject to} & \mathbf{v} = Ab & b \succeq \mathbf{0}_n \\ & \mathbf{v}' = A'b & \mathbf{1}_n^T b = 1 \\ & M_L^{\mathcal{X}} b = b_L & \end{array} \tag{4.10}$$

If the solution to this problem is negative, it follows that a' is maximal at some point b, which means that neither action can be taken without further information. Unfortunately, the target function in this optimization is non-convex. Taking the epigraph (*i.e.* a variable defined over the points above the graph) of the first term of the target function, the problem becomes:

$$\begin{array}{lll} \text{minimize} & s - \max_j \mathbf{v}'_j & \\ \text{subject to} & Ab \preceq \mathbf{1}_k s & b \succeq \mathbf{0}_n \\ & \mathbf{v}' = A'b & \mathbf{1}_n^T b = 1 \\ & M_L^{\mathcal{X}} b = b_L & \end{array} \tag{4.11}$$

Recall that b_L is the belief factor that is locally available to the agent whose actions are being considered, so it is a constant term in this problem – the only free variable is b. If the vectors in $|\Gamma^{a'}|$ (rows of A') are then taken individually, the problem trivially becomes the LP:

$$\begin{array}{lll} \forall i = 1, \ldots, |\Gamma^{a'}| & \text{maximize} & \Gamma_i^{a'} b - s \\ & \text{subject to} & Ab \preceq \mathbf{1}_k s \quad b \succeq \mathbf{0}_n \\ & & M_L^{\mathcal{X}} b = b_L \quad \mathbf{1}_n^T b = 1 \end{array} \tag{4.12}$$

If the solution b_{opt} to each of these LPs is such that $\max_i (Ab_{opt})_i \geq \max_j (A'b_{opt})_j$, then action a can be safely selected based on b_L. If this is not the case for any of the solutions, then it is not possible to map the agent's best action solely through b_L. In order to disambiguate every possible action, this optimization needs to be carried out for all conflicting pairs of actions.

An alternative is to introduce the slack variable ξ in the constraints of (4.11):

$$\begin{aligned} Ab &\preceq \mathbf{1}_k s & b &\succeq \mathbf{0}_n \\ A'b &= \mathbf{1}_{k'} s + \xi & \mathbf{1}_n^T b &= 1 \\ M_L^{\mathcal{X}} b &= b_L \end{aligned} \tag{4.13}$$

If the maximum element of ξ is positive at some b, then it can safely concluded that $\max_i \mathbf{v}_i \leq \max_j \mathbf{v}'_j$ and therefore the actions are undecidable. The problem of maximizing the maximum element of ξ, however, is only solvable by splitting ξ into its positive and negative components, ξ^+ and ξ^-, and requiring that $(\xi^+)^T \cdot \xi^- = 0$. The latter constraint is itself non-convex, and at best it increases the complexity of the optimization procedure beyond that of the exhaustive LP (4.12). The full optimization problem which is here considered is reducible to that of L_∞ error minimization, for which there is no known method (Patrascu et al., 2002). In order to contain this problem as an LP, these constraints can be relaxed, and the problem can be instead described as:

$$\begin{aligned} \text{maximize} \quad & \mathbf{1}_{k'}^T \xi \\ \text{subject to} \quad & Ab \preceq \mathbf{1}_k s & b &\succeq \mathbf{0}_n \\ & A'b = \mathbf{1}_{k'} s + \xi & \mathbf{1}_n^T b &= 1 \\ & M_L^{\mathcal{X}} b = b_L \end{aligned} \tag{4.14}$$

The target function in this optimization is not the same as in the original problem (4.10), since it instead seeks to find the point b with the highest average difference between the maximum element of $\mathbf{v}$ and the values of A' (highest mean value of ξ). While the optimal solution to this problem is typically achieved at a point where ξ has positive components, this is not necessarily so, and therefore this must be considered as an approximate solution to the original problem. Since the vectors in A and A' are arbitrary as long as the full value function is convex, it is also difficult to establish a bound on the quality of this approximation. In practice, for the examples which are here studied, is was found that using (4.14) instead of (4.12) does not noticeably affect the quality of the resulting communication map, and allows better scalability to larger domains.

4.4.3 Mapping Local Belief Points to Communication Decisions

For an environment with only two belief factors, the method described so far could already incorporate an explicit communication policy: given the local belief b_L of an agent, if it is possible to unequivocally identify any action as being maximal, then that action can be safely executed without any loss of expected value. Otherwise, the remaining belief factor should be requested from other agents, in order to reconstruct b through (4.2), and map that agent's action through the joint policy. However, in most scenarios, it is not sufficient to know whether or not to communicate: equally important are the issues of what to communicate, and with whom.

Consider now the general problem with F belief factors contained in the set $\mathcal{Y}$. In this case there are $2^{|\mathcal{Y}|-1}$ combinations of non-local factors which the agent can request. Our goal is to identify one such combination which contains enough information to disambiguate the agent's actions. Central to this process is the ability to quickly determine, for a given set of belief factors $\mathcal{G} \subseteq \mathcal{Y}$, if there are no points in $b_{\mathcal{G}}$ with non-decidable actions. The exact solution to this problem would require, in the worst case, the solution of $|\Gamma^a| \times |\Gamma^{a'}|$ LPs of the form (4.12) for every pair of actions with conflicting value bounds. However, a modification of the approximate LP (4.14) allows this problem to be tackled efficiently:

$$\begin{array}{lllll} \text{maximize} & \mathbf{1}_{k'}^{\mathrm{T}}\xi' + \mathbf{1}_{k}^{\mathrm{T}}\xi & & & \\ \text{subject to} & Ab \preceq \mathbf{1}_k s & A'b = \mathbf{1}_{k'} s + \xi & M_L^{\mathcal{X}} b = b_L & \\ & A'b' \preceq \mathbf{1}_{k'} s' & Ab' = \mathbf{1}_k s' + \xi' & M_L^{\mathcal{X}} b' = b_L & \\ & b \succeq \mathbf{0}_n & b' \succeq \mathbf{0}_n & M_{\mathcal{G}}^{\mathcal{X}} b = M_{\mathcal{G}}^{\mathcal{X}} b' & \end{array} \tag{4.15}$$

The rationale behind this formulation is that any solution to the LP, in which $\max_i \xi_i > 0$ and $\max_j \xi'_j > 0$ simultaneously, identifies two different points b and b' which map to the same point $b_{\mathcal{G}}$ in $\mathcal{G}$, but share different maximizing actions a' and a respectively. This implies that, in order to select an action unambiguously from the belief over $\mathcal{G}$, no such solution may be possible.

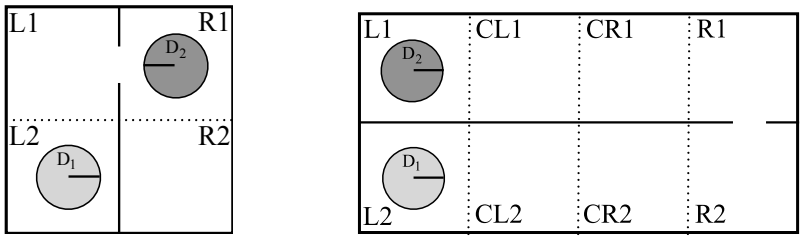
Equipped with this result, it is now possible formulate a general procedure that, for a set of belief points in local space, returns the corresponding belief factors which must be communicated in order for an agent to act unambiguously. This is referred to as obtaining the *communication map* for the problem. This procedure, detailed in

Algorithm 1, is as follows: begin by computing the value bounds of V over local factors L, and sampling N reachable local belief points b_L; for each of these points, if the value bounds of the best action are not conflicting (see Section 4.4.1), or any conflicting bounds are resolved by LP (4.14), mark b_L as *safe*, add it to the communication map, and continue on to the next point; otherwise, using LP (4.15), search for the minimum set of non-local factors $\mathcal{G}$ which resolves all conflicts; then associate b_L with $\mathcal{G}$ and add it to the map. The inputs to the algorithm are the set of local factors, L, the set of non-local factors $\mathcal{Y}$, the value function V, and the number of desired samples N. The output is a set of pairs $\langle b_L, \mathcal{G} \rangle$ of local belief points and associated communication decisions.

Algorithm 1 CreateCommunicationMap($L, \mathcal{Y}, V, N$)

1: $\{Single\_LP(b_L, a, a')$ refers to (4.14)$\}$
2: $\{Full\_LP(factors, b_L, a')$ refers to (4.15)$\}$
3: $Samples \leftarrow$ sample N reachable local belief points b_L;
4: $bounds \leftarrow$ obtain local value bounds of V; $Map \leftarrow \emptyset$;
5: **for all** $b_L \in Samples$ **do**
6: $\quad \alpha' \leftarrow \arg\max_\alpha \overline{V_\alpha}(b_L)$;
7: $\quad$ **if** $\underline{V_{\alpha'}}(b_L) \geq \overline{V_\alpha}(b_L) \quad \forall \alpha \neq \alpha'$ or $Single\_LP(b_L, a, a') > 0 \quad \forall \alpha \neq \alpha'$ **then**
8: $\quad\quad Map \leftarrow Map \cup \langle b_L, \emptyset \rangle$;
9: $\quad$ **else**
10: $\quad\quad \mathcal{G} \leftarrow \emptyset$; $\mathcal{H} \leftarrow \mathcal{Y}/L$;
11: $\quad\quad$ **while** $\mathcal{H}$ is not empty **do**
12: $\quad\quad\quad temp \leftarrow$ remove factor from $\mathcal{H}$; $factors \leftarrow \mathcal{H} \cup \mathcal{G}$;
13: $\quad\quad\quad$ **if** $Full\_LP(factors, b_L, a')$ returns both negative solutions **then**
14: $\quad\quad\quad\quad \mathcal{G} \leftarrow temp$;
15: $\quad\quad\quad$ **end if**
16: $\quad\quad$ **end while**
17: $\quad\quad Map \leftarrow Map \cup \langle b_L, \mathcal{G} \rangle$;
18: $\quad$ **end if**
19: **end for**
20: **return** Map

During execution, an agent updates its local information b_L, finds the nearest neighbor point in the communication map, and requests the corresponding factors from the other agents. The agent then selects the action associated with the highest maximum value bound given the resulting information.



(a) *Relay-Small.* **(b)** *Relay-Large.*

Figure 4.5: (a) Layout of the *Relay-Small* problem. (b) Layout of the *Relay-Large* problem.

4.5 Experiments

An analysis is now given to the results of applying the aforementioned offline communication mapping process to three different MPOMDP environments, each with a different degrees of interdependency between agents. The full state, action, and observation spaces of each of the following test problems can be found in Appendix A.

The first and smallest of the test problems is the *Relay-Small* problem introduced in Section 4.3, and is mainly used for explanatory purposes. The fact that, in this problem, each belief factor is two-dimensional (each factor spans one of the rooms) allows the visualization of the results of the proposed method. In Figure 4.7, it can be seen that some of the agent's expected behavior is already contained in the value bounds over its local factor: if an agent is certain of being in room R1 (i.e. $(b_{\mathcal{X}_1})_1 = 0$), then the action with the highest-valued bound is *Shuffle*. Likewise, an *Exchange* should only be carried out when the agent is certain of being in L1, but it is an ambiguous action since the agent needs to be sure that its teammate can cooperate. Figure 4.6 represents the communication map which was obtained offline through the proposed algorithm. Since there are only two factors, the agent only needs to make a binary decision of whether or not to communicate for a given local belief point. The belief points considered *safe* are marked as 0, and those associated with a communication decision are marked as 1. In terms of quantitative results, shown in Table 4.1, it can be seen that $\sim 30 - 40\%$ of communication episodes are avoided in this simple example, without a significant loss of collected reward.

Another test scenario is the OneDoor environment of Oliehoek et al. (2007). In this problem, two agents operate in a 49-state grid-like world, represented in Figure 4.8, and may each be in one of 7 possible positions. One of the agents is known to be in positions

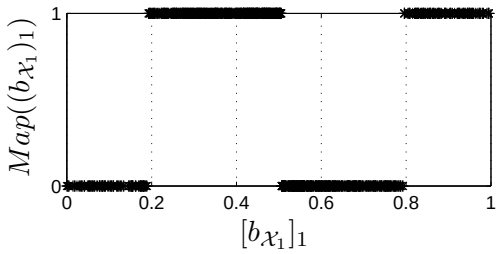


Figure 4.6: Communication map for the *Relay-Small* problem.

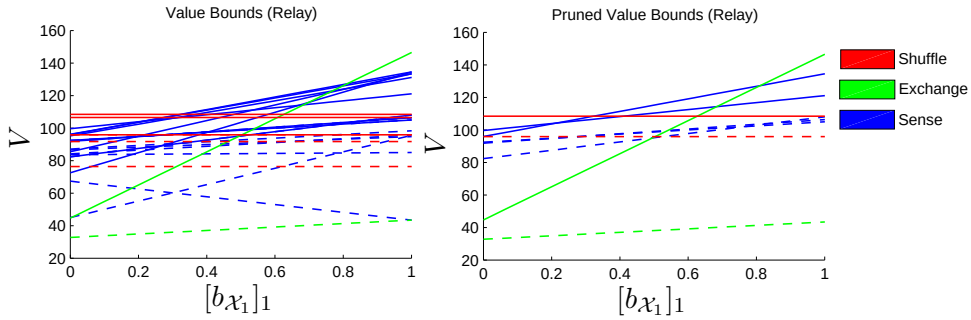


Figure 4.7: Value bounds for the *Relay-Small* problem. The dashed lines indicate the minimum value bounds, and the filled lines represent the maximum value bounds, for each action. Left: the bounds for all vectors in the joint value function. Right: pruned bounds.

1, 2 or 3 (with uniform probability) and has the goal of reaching position 5. The other starts in positions 5, 6 or 7 and must reach position 3. Each agent can move in any of the four directions, with an associated probability of ending up in an unintended neighbor state, and can observe positions 2, 4 and 6 with no noise. The remaining positions are indistinguishable to the agent (a case of perceptual aliasing). Therefore $|\mathcal{O}_i| = 4$. The robots may share the same position, and they receive a penalty for being both in position 4 at the same time. They receive a positive reward for reaching their goal, and no reward otherwise. The agents are uncoupled expect through the reward

	Relay-Small		OneDoor		Relay-Large	
h.	Full Comm.	Red. Comm.	Full Comm.	Red. Comm.	Full Comm.	Red. Comm.
6	15.4, 100%	14.8, 56.9%	0.35, 100%	0.30, 89.0%	27.4, 100%	25.8, 44.1%
10	39.8, 100%	38.7, 68.2%	1.47, 100%	1.38, 76.2%	-19.7, 100%	-21.6, 62,5%
∞	77.5, 100%	73.9, 46.1%	2.31, 100%	2.02, 61.3%	134.0, 100%	129.7, 58.9%

Table 4.1: Results of the proposed method for various environments. For settings assuming full and reduced communication, results show (average accumulated discounted reward, online communication usage).

	Relay-Small			OneDoor			Relay-Large		
h	6	10	∞	6	10	∞	6	10	∞
PERSEUS	1.1	4.3	0.1	7.3	33.3	5.3	239.5	643.0	31.5
Comm. Map	5.9	21.4	7.4	12.4	57.7	5.9	368.7	859.5	138.1

Table 4.2: Running time (in seconds) of the proposed method in comparison to the PERSEUS point-based POMDP solver. The version of PERSEUS that was used for the stationary case ($h = \infty$) was optimized to that setting, which explains its faster solution times.

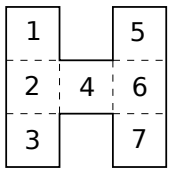


Figure 4.8: Representation of the OneDoor scenario.

function (i.e. this is a transition-observation independent version of the problem). Even so, this means that an acceptable policy in this problem must be such that one of the agents waits for the other to clear the "door" in position 4 until it attempts to move there.

If a sufficiently large horizon is considered, this problem allows for a significant reduction in communication, using the proposed method — up to 38.7% for $h = \infty$. This is because a near-optimal joint policy defines wich agent should take priority, and since that agent always moves first, it rarely needs to communicate (only when the other agent has a sufficient probability of moving to position 4 due to the noise in its actions). The other agent, in turn, must communicate until its partner clears the door, and afterwards, its local actions can also be taken independently and so it ceases communication. For horizons smaller than 10, however, the agents may not have enough decisions left to gather any positive reward, and in these cases they both communicate in order to avoid any possible collisions. For $h = 6$, communication is only avoided in 11% of all decision steps.

Note that this relationship between the problem's horizon and the amount of communication savings does not hold for all of the problems. The proposed method exploits the invariance of local policies over subsets of the joint belief space, and this may arbitrarily change with the problem's horizon.

A larger example is displayed in Figure 4.5b. This is an adaptation of the *Relay-Small* problem (named *Relay-Large*) to a setting in which each room has four different states, and each agent may be carrying a package at a given time. Agent D_1 may retrieve new packages from position L1, and D_2 can deliver them to L2, receiving for that a positive reward. There are a total of 64 possible states for the environment. Here, the agents can act independently for a larger number of steps, since they must first collect a package, move to the correct position for their interaction, and only then cooperate. The communication savings are more pronounced, as shown in Table 4.1, with up to 41.1% reduction for $h = \infty$.

Finally, it is here argued that the running time of the proposed algorithm is comparable to that of general POMDP solvers for these same environments. Even though both the solver and the mapper algorithms must be executed in sequence, the results in Table 4.2 show that they are typically both in the same order of magnitude, for all of the tested scenarios, and regardless of the horizon that is selected for the respective decision-making problem.

4.6 Summary

Traditional multiagent planning on partially observable environments mostly deals with fully-communicative or non-communicative situations. For a more realistic scenario where communication should be used only when necessary, state-of-the-art methods are only capable of approximating the optimal policy at run-time (Roth et al., 2005a; Wu et al., 2009). The properties of MPOMDP models which can be exploited in order to increase the efficiency of communication between agents were here analyzed. It was shown that these properties hold, for various MPOMDP scenarios, and that the decision quality can be maintained while significantly reducing the amount of communication, as long as the dependencies within the model are sparse.

Chapter 5

Continuous-Time Execution and Planning for Teams of Robots

As we have briefly discussed in Section 3.2.4, one of the fundamental steps involved in modeling of real-world systems through DT frameworks is the selection of an appropriate *execution strategy* for the decision-making agent(s). Tractable classes of DT models assume that the underlying process evolves according to discrete-time sequential decisions, but for physical agents acting on the real world, the underlying process operates over continuous time. We have also seen that synchronous execution strategies, while conceptually simple, exhibit undesirable characteristics, such as the loss of reactivity, or the induction of superflous actions, resulting in unnecessarily large planning horizons.

In contrast, the system modeling formalisms which are most ubiquitous in robotics applications, which stem from the theory of Discrete-Event Systems, assume by default that the decision-making process is driven by asynchronous, randomly occurring events. This is the case, most notably, for robotic controllers based on Finite State Automata (Damas and Lima, 2004; Quottrup et al., 2004), or Petri Nets (Costelha and Lima, 2007; Herrero-Perez and Martinez-Barbera, 2008; Rosell et al., 2003). By themselves, DES frameworks do not address the problem of automated planning or learning; instead, they assume that a plan is available beforehand, and they can be used to represent that plan and to exeucte it, formally analyzing its relevant properties.

It is a reasonable hypothesis, then, that there is a strong potential for real-world applicability, particularly in the domain of robotics, in the common ground between DT and DES modeling formalisms. This is one of the central claims of this thesis. In this

and in the following chapters, we will evaluate existing approaches, and propose novel methodologies, that exploit the synergy between the real-world descriptive capabilities of DES theory, and the automated planning and learning methodologies of decision theory. The distinctive feature to these approaches is that they are simultaneously multiagent *and* event-driven.

Existing approaches that combine DT and DES include the recent works of Neto (2010) and Yamasaki and Ushio (2005), although the topic remains otherwise largely unexplored. These approaches consider the problem of obtaining a *supervisor* for a DES as one of reinforcement learning over an embedded (PO)MDP. That is, they start from DES representations, and later obtain equivalent DT models for learning / planning purposes. In the approaches which we present in this thesis, the system models are wholly within the domain of DT to begin with, but at the same time they subscribe to the DES event-driven paradigm.

Other related work on event-driven MDPs deals with such events without explicitly modeling the effect of continuous time: by keeping track of event histories in the system state (Becker et al., 2004), or by considering the occurrence of non-Markovian events as being unpredictable (Witwicki et al., 2013).

In the current chapter, we will describe the requirements to the modeling of multi-robot decision-making as an event-driven process. We will then evaluate the performance of the DT framework that best fits the aforementioned requirements – the GSMDP framework. GSMDPs are explicitly event-driven, and will be shown to have several advantages over synchronous MDP models. First, by explicitly modeling the temporal occurrence of events, the non-Markovian effects of state and action space discretization can be minimized, increasing solution quality. Second, since events are allowed to occur at any time, the system is fully reactive to sudden changes. And finally, communication between agents will only be required upon the occurrence of an event, as opposed to having a fixed rate.

Despite their suitability, there haven't been any reported applications of GSMDPs to real-world multiagent systems. This constitutes the central contribution of this chapter. We will revisit the robotic soccer case study introduced in Chapter 3 from an event driven perspective, and comparatively analyze synchronous (standard MDP) and asynchronous (GSMDP) modeling approaches with respect to real time performance. Finally, we note that, for this analysis to be possible, we will here assume that our

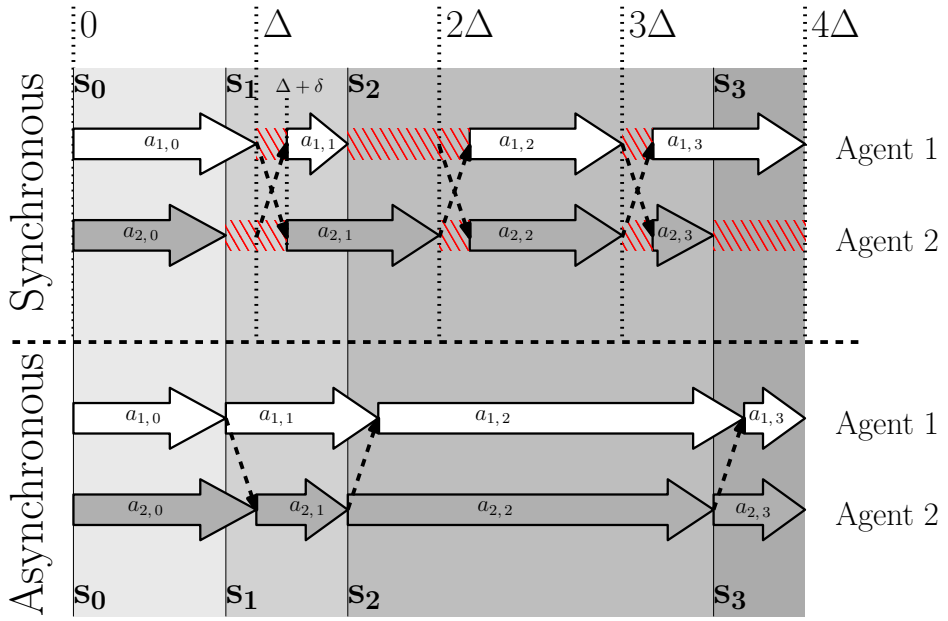


Figure 5.1: Action selection in synchronous and asynchronous execution of a multi-robot system. In these conceptual examples, the solid arrows represent the temporal duration of the actions of each agent $a_{\{1,2\},n}$, and the underlying shaded regions represent the sojourn time of the system in states $s_{0,\dots,3}$. In synchronous operation, actions are jointly taken at positive multiples of Δ. During the gaps between the end of a given local action and the beginning of the next decision step, agents are forced to idle (red patterned intervals). Dashed arrows between agents represent communication instances, where δ is a communication delay; In asynchronous execution, a new decision step starts immediately after a transition is detected, so there is no idle time. Furthermore, only the agent that detects an event needs to communicate.

multi-robot system is jointly fully observable – the partially observable extension to these methods will be discussed in Chapter 6; and that, for planning purposes, communication is free, following the arguments presented in Chapter 4.

5.1 Event-Driven Multi-Robot Systems: Beyond SMDPs

We have discussed the properties of synchronous execution of discrete-time policies for physical agents, and, in particular, we saw that it carries notable drawbacks. The natural alternative to this execution strategy is an "asynchronous", or event-driven scheme, where agents can be required to act at any given point in time, as a response to changes in their enveloping environment. We will now show, however, that multi-robot systems with asynchronous dynamics cannot be easily modeled as Markovian or semi-Markovian

decision processes, and that a generalization to the latter is indeed necessary under these circumstances.

The most important difference in the dynamics of synchronous and event-driven systems concerns the relationship between *decision instants* and *state transitions* in real time. In either case, state transitions are a property of the physical system, so their stochasticity is independent of the execution strategy selected for its respective agents. For generality, let us assume that the state transitions of our physical system occur at random points in time. Then:

- In a synchronous execution strategy, decisions are taken at predetermined instants, for example at $t = k\Delta$ for some period $\Delta > 0$ and $k \in \{0, \dots, h-1\}$. That is, decision instants are not simultaneous with state transitions[1];
- In an event-driven decision-making process, decisions are taken only at state transition instants. In other words, a state transition is a necessary, but not sufficient, condition for a decision to be taken[2].

In practice, this implies that, while a discrete-time model evolves between decision instants according to the time-invariant distribution $\Pr(s' \mid s, \mathbf{a})$, an event-driven model should account for the fact that state transitions may take variable amounts of time, i.e., the system dynamics should be defined as $p(t, s' \mid s, \mathbf{a})$.

In the single agent case, an event-driven system can be modeled by an SMDP with a non-factored state space (Puterman, 1994), where $p(t, s' \mid s, \mathbf{a}) = f^{\mathbf{a}}_{s,s'}(t)T(s, \mathbf{a}, s')$, as per Definition 2.2.1. In that case, the time for the state transition $\langle s, a, s' \rangle$ to resolve is a random variable following an arbitrary distribution $f^{\mathbf{a}}_{s,s'}$.

For factored state spaces, which are typical of multiagent systems, this analysis is slightly more complex. Consider a model with a factored state space description $\langle \mathcal{S}, \mathcal{X} \rangle$ with two state factors, $\mathcal{X} = \{\mathcal{X}_1, \mathcal{X}_2\}$. In particular, assume that the dynamics of each state factor variable are naturally independent of one another, representing, for

[1] Note that discrete-time models are usually defined, for analytical tractability, over equivalent systems with the same stochastic properties, in which the state is only allowed to change at $t = k\Delta$ (Puterman, 1994). In that sense, decisions can be seen as simultaneous with state transitions. While this may be equivalent from an analytical standpoint, it is merely a simplifying assumption. It is typically not true that a physical system only changes at periodic instants.

[2] This is the most general interpretation (Puterman, 1994), but note that, as we had underlined in Section 2.2.1, we are restricting our attention, in this work, to systems in which decisions are *always* taken at state transitions.

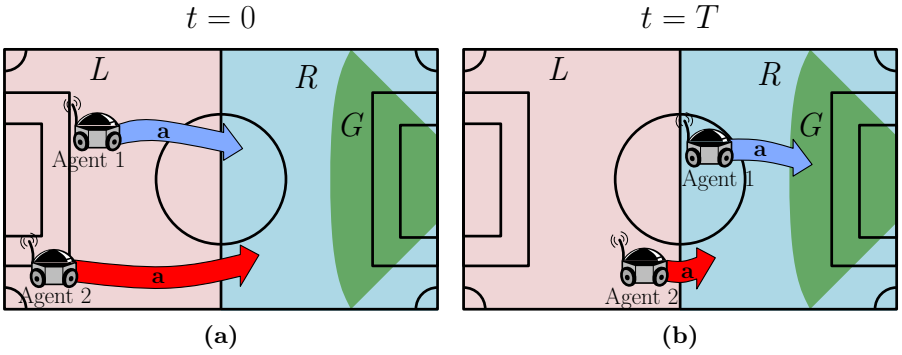


Figure 5.2: An example of an environment in which persistently enabled events are an issue: (a) At time $\mathcal{T}_1$, two agents attempt to move from state L to G through a simple navigation action a. (a) At $\mathcal{T}_2 > \mathcal{T}_1$, agent 1 detects that it has changed its local state factor (x_1), triggering a new *joint* decision. For agent 2, $\Pr(x'_2 = R \,|\, x_2 = L, a)$ is now intuitively higher, given the time that it has been moving so far. However, a memoryless discrete MDP cannot use this information. The system is not strictly Markovian.

example, the position of two mobile robots moving in parallel, such as in the conditions of Figure 5.2. In synchronous execution, we could exploit this independence to decompose the joint state transition probabilities, that is, for $s = (x_1, x_2)$ and $s' = (x'_1, x'_2)$, $\Pr(s' \,|\, s, \mathbf{a}) = \Pr(x'_1 \,|\, x_1, \mathbf{a}) \Pr(x'_2 \,|\, x_2, \mathbf{a})$. However, in an event-driven perspective, if these state factor variables represent subprocesses which experience transitions asynchronously and independently from one another, then transitions $\langle x_1, \mathbf{a}, x'_1 \rangle$ and $\langle x_2, \mathbf{a}, x'_2 \rangle$ will *never happen simultaneously*[1]. In other words, if these transitions trigger at times $\mathcal{T}_1 \sim f^{\mathbf{a}}_{x_1, x'_1}$ and $\mathcal{T}_2 \sim f^{\mathbf{a}}_{x_2, x'_2}$, then naturally $\Pr(\mathcal{T}_1 = \mathcal{T}_2) = 0$. Furthermore, recall that any state transition can trigger a new decision step. This raises two important issues: first, in this case, we cannot factor joint transition probabilities $\Pr(s' \,|\, s, \mathbf{a})$, since only one of the state factor variables is allowed to change between any two decision steps (we must actually reason over the joint $p(t, s' \,|\, s, \mathbf{a})$); and since $f^{\mathbf{a}}_{x_1, x'_1}, f^{\mathbf{a}}_{x_1, x'_1}$ are not necessarily memoryless, then if, for example, transition $\langle x_1, \mathbf{a}, x'_1 \rangle$ fires first at time $\mathcal{T}_1$, we have that in the general case $\Pr(t > \delta\,,\, x'_2 \,|\, x_2, \mathbf{a}) \neq \Pr(t > \mathcal{T}_1 + \delta\,,\, x'_2 \,|\, x_2, \mathbf{a}, t > \mathcal{T}_1)$ for any $\delta > 0$. This implies that we must keep track of the triggering time $\mathcal{T}_1$ in order to predict $\mathcal{T}_2$, which violates the Markovian assumption. This constitutes a problem of concurrent, persistently enabled events, as per Definition 2.2.4.

[1] Note that this result only applies to asynchronous subprocesses. We could also model, with event-driven dynamics, two or more state factors that are correlated in such a way that they always experience transitions simultaneously.

Note that this would not be a problem if all $f^{\mathbf{a}}_{s,s'}$ are exponential, since, in that case, through the memoryless property of that distribution, $\Pr(t > \delta\,,\, s' \,|\, s, \mathbf{a}) = \Pr(t > \mathcal{T}_1 + \delta\,,\, s' \,|\, s, \mathbf{a}, t > \mathcal{T}_1)$. That is, even if exponential events are interrupted, there is never the need to maintain the time at which they became in enabled in order to predict their triggering time. This means that multiagent, event-driven systems with fully exponential transitions could be modeled as CTMDPs (Section 2.2.2). However, **CTMDPs cannot model non-Markovian temporal distributions** (Howard, 1960).

Many of the physical processes involved in the operation of a robotic platform do not have exponential duration. For example:

- The battery life of a mobile robot, which can be modeled as a Weibull distribution;
- The time taken by a mobile robot to traverse an arbitrary path to a desired pose, which is multi-modal or infeasible to parametrize, in the general case;
- Fixed-time operations, such as timeout signals, or deliberately induced pauses in the control of a robot, which are, theoretically, Dirac delta probability density functions.

To model the above temporal distributions, or at least those of the above with well-defined Laplace transforms, we could describe the system dynamics as that of an SMDP (Section 2.2.1). But **SMDPs cannot model persistently enabled transitions**. After any transition, an SMDP loses all memory of the past execution of the system. Therefore, SMDPs cannot model, exactly and in the general case, asynchronous multiagent decision-making problems.

We see, therefore, that neither CTMDPs or SMDPs can model event-driven multi-robot systems. For this reason, we need to resort to a generalization of those frameworks.

5.2 GSMDPs as a Framework for Multi-Robot Decision-Making

The framework of Generalized Semi-Markov Decision Processes (GSMDPs), proposed by Younes and Simmons (2004) is ideally suited for the requirements of this work. It allows generic temporal probability distributions over events, while maintaining the

possibility of modeling persistently enabled (concurrent) events, which is essential in multi-robot domains.

However, to our knowledge, this framework has never been applied in a realistic multi-robot context. We show that, by allowing event-driven plan execution, the application of the GSMDP framework to multi-robot planning problems allows us to avoid the negative effects of its synchronous alternatives, resulting in greater performance.

GSMDP models can be solved by commonly used discrete-time MDP algorithms, by first obtaining an approximate Markovian model through the use of *Phase-Type* temporal distributions (Younes, 2005; Younes and Simmons, 2004). Here, we also take into account the fact that some events which are characteristic of robotic systems are not amenable to Phase-Type approximations, and that, if so, the resulting approximate systems remain semi-Markovian.

This section discusses the methodology involved in applying GSMDPs to a generic multi-robot problem, and in obtaining a useful plan from a given GSMDP model. It also describes the aspects of this work which contribute to the practical use of the theory of GSMDPs in real multi-robot scenarios.

5.2.1 From DES to GSMDPs

The similarities between the event-driven dynamics of GSMDPs (which we've reviewed in Section 2.2.3), and those considered in the theory of DES, are owed to the earlier work of Glynn (1989), who introduced the underlying Generalized Semi-Markov Process (GSMP) formalism with the purpose of analyzing *stochastic* discrete-event systems. In particular, GSMPs were introduced as a way of modeling the dynamics of Stochastic Timed Automata (STAs), which experience uncertainty both in the outcome of their transitions and in the timing of the events which drive said transitions. GSMDPs are an extension to GSMPs which add a dimension of decision-making to the dynamics of the latter.

In the theory of DES, decision-making is typically modeled through the inclusion of a suitably defined "supervisor" to the system (Cassandras and Lafortune, 1999). This supervisor, which is itself an accessory subsystem to the actual process model, interacts with the rest of the system by "controlling" certain events, *i.e.* by enabling or disabling certain transitions, in order to maintain the system in a desired set of states, or to drive it there from some arbitrary initial configuration.

This interpretation of decision-making as system supervision was also taken by Younes and Simmons (2004) in the original formulation of the GSMDP framework. In that work, "actions" are considered to be controllable events; the respective definition of "events", on the other hand, is that of Glynn (1989), who had considered them as semantically independent from states. Events were seen as the signals driving the transitions of a (possibly stochastic) state machine. There may be uncertainty in the resulting state of the system, if an event is known to happen.

However, we note that that definition departs from the interpretation taken in other modeling languages for the description of DES, such as Deterministic Finite State Automata, where sequences (*strings*) of events unambiguously identify paths in the state space of a fully observable system (Cassandras and Lafortune, 1999); or Petri Nets (or Generalized Stochastic Petri Nets), where events are seen as semantically equivalent to transitions between markings. In these examples, if the system is fully observable, then knowing that a certain "event" has happened not only informs the decision-maker that the state has changed but also *how* it changed. If, instead, by observing an event, the decision-maker can only have partial knowledge over the resulting states of the system (as in the GSMP framework), then that system would be more aptly described as partially observable – in this case, the distinction between uncertainty in state transitions and uncertainty in observations becomes muddled.

In Section 2.2.3, we've noted that our interpretation of a decision-theoretic discrete event-driven system is different from that of Younes and Simmons (2004). According to Definition 2.2.2, events are regarded as abstractions of state transition pairs, which in turn may be enabled (made possible) at certain states by certain actions. Since state transitions are stochastic, then so are events; but knowing that an event has happened in a given state gives agents full knowledge of the resulting state of the system. This means that such a system can be controlled equivalently by observing either states or events. Drawing a parallel to the theory of DES, we note that our interpretation is closest to that of Garg (1992) and Lawford and Wonham (1993), in their definition of Probabilistic Discrete Event Systems (PDES), where events have a well-defined probability of occurring in a state; although we restrict our attention to systems in which, given the occurrence of an event, the induced state transition is deterministic.

In Chapter 6, we will show how this interpretation can also be extended to handle partial observability over events. Using our interpretation, there is a clear distinction between what constitutes a fully or partially observable event-driven system, in a way that is consistent with the decision-theoretic concepts of full or partial observability.

Finally, we also consider that our interpretation allows for a clearer description of the GSMDP model dynamics in multiagent scenarios, consistently with other multiagent MDP-based frameworks. In Younes (2005), the author suggests that a multiagent GSMDP can be modeled by selecting a *set of events* at every decision, rather than just selecting one controllable event at a time. Each enabled event in this set would correspond to a *local* action of one of the agents. However, even if a set of events is enabled at a given decision step, only one of them may fire at a given time. If only one event can influence the next-state probabilities at any decision step, and actions are equated to events, then this contrasts with the typical dynamics of multiagent DT models, where the input of *all* agents can affect state transitions. The execution of particular *joint* (simultaneous) actions can be crucial to the evolution of a multiagent decision-making process. Specific joint actions could modeled explicitly as single events, but in that case, the model would be centralized, and undistinguishable from a single agent system[1].

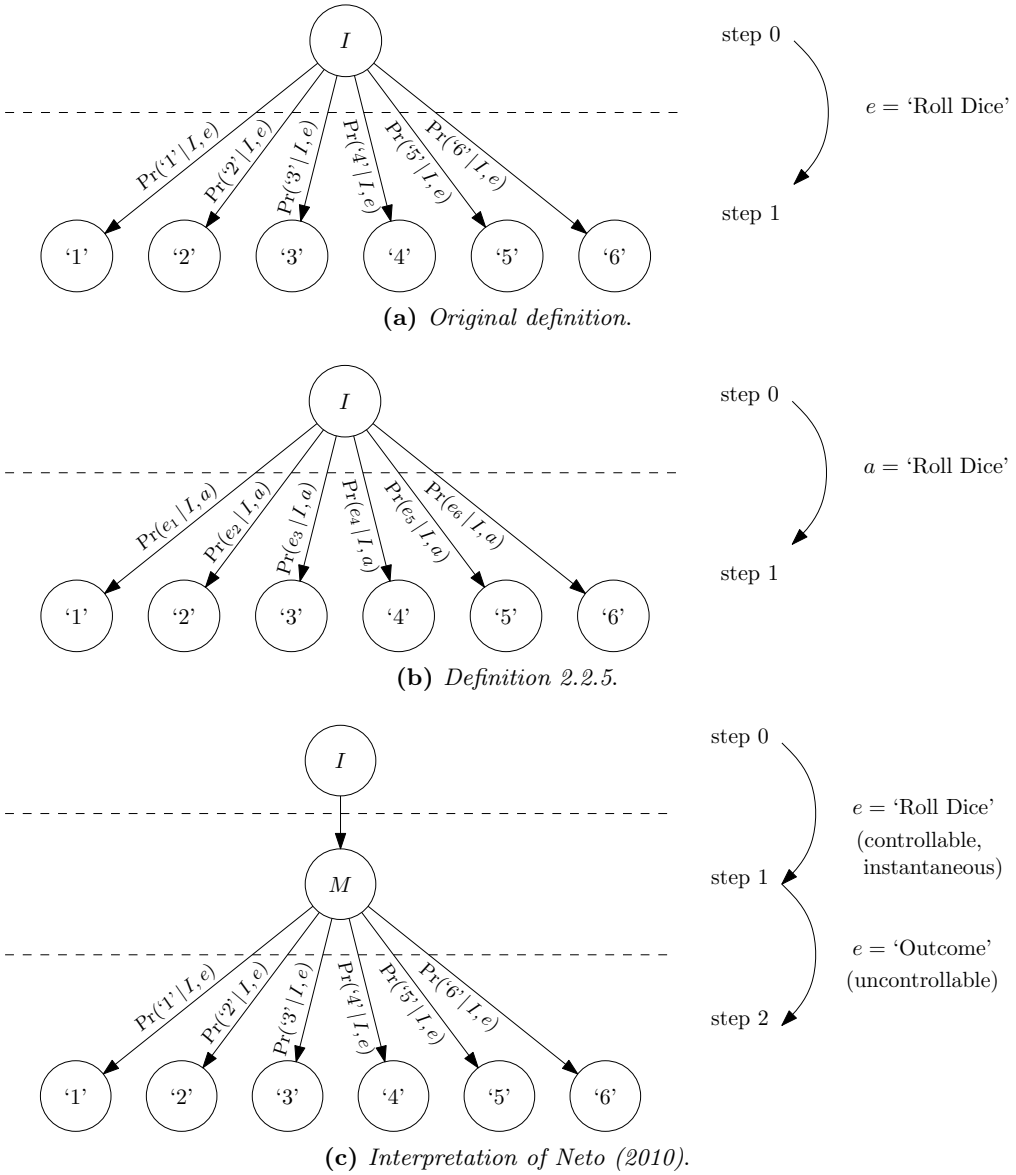
The following example illustrates the practical differences, when modeling event-driven stochastic systems, between the set of definitions introduced in this work, and those used by Younes and Simmons (2004) in the original formulation of GSMDPs.

Example 5.2.1. Modeling a dice throw as an event-driven process: *Consider the throw of a fair dice, modeled as a stochastic process with* $S = \{I, \text{`1'}, \text{`2'}, \text{`3'}, \text{`4'}, \text{`5'}, \text{`6'}\}$. *The initial state of the process is* $s_0 = I$, *and* $S \backslash I$ *is the space of possible outcomes at step* 1.

Using the definitions of Younes and Simmons (2004) and Glynn (1989), this problem can be modeled as a GSMDP, by considering that it evolves from step 0 *to step* 1 *following the occurrence of a single controllable event,* $e =$ *'Roll Dice'. As the agent enables* e, *the instant of the next decision step is sampled from* $p(t \mid e)$. *At that instant, the system experiences a transition into* $s_1 \in S \backslash I$, *according to the transition probabilities* $Pr(s_1 \mid I, e) = \frac{1}{6}$. *This process is depicted in Figure 5.3a.*

According to Definition 2.2.5, which we introduce in this work, each of the possible outcomes of this stochastic process is itself an "event". Our interpretation of the process

[1] Another solution to obtain equivalent behavior is to add more intermediate states to the system, adding further to its complexity



(a) *Original definition.*

(b) *Definition 2.2.5.*

(c) *Interpretation of Neto (2010).*

Figure 5.3: An example of the practical difference in the definition of events, in the context of modeling a dice throw. **a)** the original GSMDP formulation of Younes and Simmons (2004); **b)** Through Definition 2.2.5 used in this work; **c)** The interpretation of controllable events as instantaneous transitions of Neto (2010).

is shown in Figure 5.3b. At step 0*, the agent performs action* $a =$ *'Roll Dice'; at which point, one of the six possible events* $e_{i=1,\dots,6}$ *is sampled from the uniform distribution* $p(e|s,a)$ *(that is, from* $T(s,a,e)$*). The instant of the next decision step is then sampled from* $f^a_{e_i}(t) = p(t|a,e_i)$*. Events are here seen as possible consequences of actions, rather*

than considering actions to be special cases of events.

Another way of modeling this problem as a GSMDP, through the definitions of Younes and Simmons (2004), but with comparable dynamics to those of the process shown in Figure 5.3b, is shown in Figure 5.3c, and considers an additional intermediate state, M, between the initial state of the system and the possible outcomes. If M is reached immediately, and deterministically, after the controllable event is enabled, then, at step 3*, each of the different outcomes of the process could be reached by a distinct* exogenous *event e'_i. For these events, $\Pr(s_j \mid M, e'_i) = 1$ if and only if $i = j$, and is* 0 *otherwise. This approach of considering controllable events as instantaneous transitions into intermediate states is taken by Neto (2010) to model discrete event systems. However, from a DT perspective, this approach is not useful, not only because it adds, unnecessarily, to the operational complexity of the model (by adding more states), but also because it artificially inflates the horizon of the decision-making problem (since more steps are considered).*

We will now show that GSMDP models can be converted across both sets of definitions, to show that the differences in our interpretation of event-driven dynamics are not analytically restrictive, and so they do not result in any loss of generality in the discussion and methodologies that are presented this work with respect to event-driven systems.

Proposition 5.2.1. *A GSMDP $G = \langle \mathcal{S}, \mathcal{A}, T, \mathcal{F}, R, C, \mathcal{E} \rangle$, which satisfies Definition 2.2.5, can be converted to a single-agent GSMDP $\overline{G} = \langle \overline{\mathcal{S}}, \overline{T}, \overline{\mathcal{F}}, \overline{R}, \overline{C}, \overline{\mathcal{E}} \rangle$ which satisfies the definition of Younes and Simmons (2004), namely, where: $\overline{\mathcal{S}}$ is a set of states; $\overline{\mathcal{E}}$ is a set of events, of which some are* controllable*; $\overline{T} : \overline{\mathcal{E}} \times \overline{\mathcal{S}} \times \overline{\mathcal{S}} \to [0, 1]$ models transition probabilities; $\overline{\mathcal{F}}$ contains temporal distributions for events; $\overline{R} : \overline{\mathcal{S}} \times \overline{\mathcal{E}} \times \overline{\mathcal{S}} \to \mathbb{R}$ contains immediate rewards; and $\overline{C} : \overline{\mathcal{S}} \times \overline{\mathcal{E}} \to \mathbb{R}$ contains cumulative reward rates.*

Proof. Let $\overline{\mathcal{E}}$ be composed entirely of *controllable* events, and let $\Psi : \mathcal{A} \to \overline{\mathcal{E}}$ be a bijection between the actions of G and the set of events of $\overline{G}$. Ψ identifies the event in $\overline{\mathcal{E}}$ that should be enabled, given an action $a \in \mathcal{A}$. Note that, since $\overline{G}$ is explicitly single agent, as per Younes (2005), one (and only one) event in $\overline{\mathcal{E}}$ must be enabled at any decision step. Also, from Definition 2.2.2, $\Phi : \mathcal{S} \times \mathcal{S} \to \mathcal{E}$ is implicitly defined, and maps state transitions in G to events in G. Then, we can define $\overline{\mathcal{S}}$, $\overline{T}$, $\overline{\mathcal{F}}$, $\overline{\mathcal{R}}$ and $\overline{C}$ such that:

1. $\overline{\mathcal{S}} = \mathcal{S}$ (same state space);

2. $\overline{T}(\Psi(a), s, s') = T(s, a, \Phi(s, s'))$, $\forall a \in \mathcal{A}, \langle s, s' \rangle \in \mathcal{S} \times \mathcal{S}$ (the events and actions of G are converted, respectively, to state transitions and events in $\overline{G}$);

3. $\overline{f}_{s,s'}^{\Psi(a)}(t) = f_{\Phi(s,s')}^{a}(t)$, $\forall a \in \mathcal{A}, \langle s, s' \rangle \in \mathcal{S} \times \mathcal{S}$ and $\overline{\mathcal{F}} = \bigcup_{\langle s,s' \rangle \in \overline{\mathcal{S}} \times \overline{\mathcal{S}}, e \in \overline{\mathcal{E}}} \overline{f}_{s,s'}^{e}(t)$ (same rationale as in Point 2);

4. $\overline{\mathcal{R}}(s, \Psi(a), s') = R(s, a, s')$, $\forall a \in \mathcal{A}, \langle s, s' \rangle \in \mathcal{S} \times \mathcal{S}$[1] (actions in G are converted to events in $\overline{G}$);

5. $\overline{C}(s, \Psi(a)) = C(s, a)$, $\forall a \in \mathcal{A}, s \in \mathcal{S}$ (same rationale as in Point 4).

□

Proposition 5.2.2. *A single-agent GSMDP $\overline{G} = \langle \overline{\mathcal{S}}, \overline{T}, \overline{\mathcal{F}}, \overline{R}, \overline{C}, \overline{\mathcal{E}} \rangle$ which satisfies the definition of Younes and Simmons (2004), can be converted to a GSMDP $G = \langle \mathcal{S}, \mathcal{A}, T, \mathcal{F}, R, C, \mathcal{E} \rangle$, which satisfies Definition 2.2.5.*

Proof. Let $\overline{\mathcal{E}}_c \subseteq \overline{\mathcal{E}}$ represent the controllable events in $\overline{G}$. Define $\mathcal{A}$ so that $|\mathcal{A}| = |\overline{\mathcal{E}}_c|$ and so that there is a bijection $\Psi : \mathcal{A} \to \overline{\mathcal{E}}_c$. Then, we can define $\mathcal{S}$, T, $\mathcal{F}$, R, C and $\mathcal{E}$ so that:

1. $\overline{\mathcal{S}} = \mathcal{S}$ (same state space);

2. $\mathcal{E}$ is arbitrary, as long it satisfies Definition 2.2.2 over $\mathcal{S}$;

3. $T(s, \Psi^{-1}(e), \Phi(s, s')) = \overline{T}(e, s, s')$, $\forall e \in \overline{\mathcal{E}}_c, \langle s, s' \rangle \in \mathcal{S} \times \mathcal{S}$, and, for each action $a \in \mathcal{A}$, $T(s, a, \Phi(s, s')) = \overline{T}(e, s, s')$, $\forall e \in \overline{\mathcal{E}} \backslash \overline{\mathcal{E}}_c, \langle s, s' \rangle \in \mathcal{S} \times \mathcal{S}$ (controllable events in $\overline{G}$ are converted to actions in G, and transitions depending on uncontrollable events are converted to transitions that are independent from actions);

4. $f_{\Phi(s,s')}^{\Psi^{-1}(e)}(t) = \overline{f}_{s,s'}^{e}(t)$, $\forall e \in \overline{\mathcal{E}}_c, \langle s, s' \rangle \in \mathcal{S} \times \mathcal{S}$, and, for each action $a \in \mathcal{A}$, $f_{\Phi(s,s')}^{a}(t) = \overline{f}_{s,s'}^{e}(t)$, $\forall e \in \overline{\mathcal{E}} \backslash \overline{\mathcal{E}}_c, \langle s, s' \rangle \in \mathcal{S} \times \mathcal{S}$ (same rationale as in Point 3);

5. $R(s, \Psi^{-1}(e), s') = \overline{R}(s, e, s')$, $\forall e \in \overline{\mathcal{E}}_c, (s, s') \in \mathcal{S} \times \mathcal{S}$, and, for each action $a \in \mathcal{A}$, $R(s, a, s') = \overline{R}(s, e, s')$, $\forall e \in \overline{\mathcal{E}} \backslash \overline{\mathcal{E}}_c, \langle s, s' \rangle \in \mathcal{S} \times \mathcal{S}$ (same rationale as in Point 3);

6. $C(s, \Psi^{-1}(e)) = \overline{C}(s, e)$, $\forall e \in \overline{\mathcal{E}}_c, s \in \mathcal{S}$ and, for each action $a \in \mathcal{A}$, $C(s, a) = \overline{C}(s, e)$, $\forall e \in \overline{\mathcal{E}} \backslash \overline{\mathcal{E}}_c, s \in \mathcal{S}$ (same rationale as in Point 3).

□

[1] Recall from Chapter 2 that $R(s, a, s')$ models can be reduced to $R(s, a)$ form. We've used the latter in Definition 2.2.5, but this is not a limiting assumption.

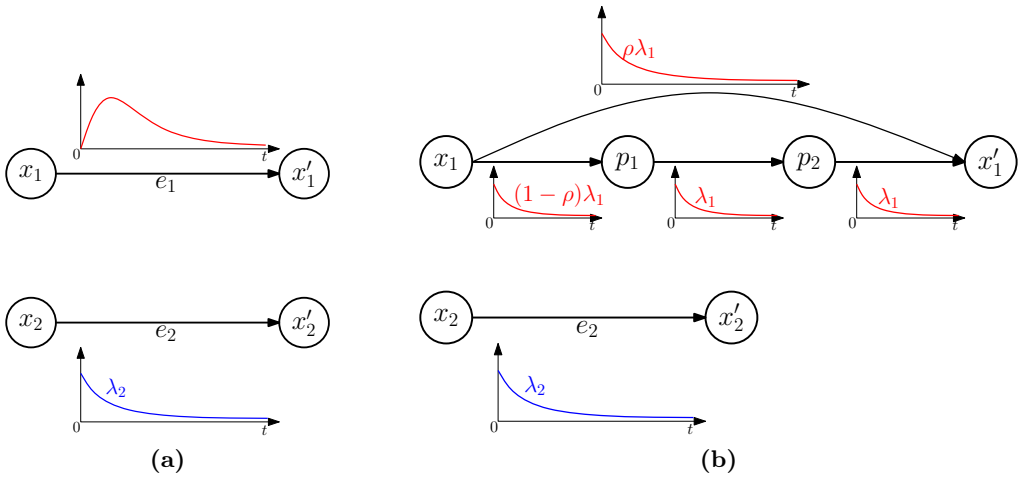


Figure 5.4: Approximating non-Markovian events through Phase-Type distributions. The topmost temporal distribution in Figure 5.4a is approximated through the Markov chain shown in its place, in Figure 5.4b. The newly added Markov chain models a Generalized Erlang distribution with three phases, which matches the first two moments of the original distribution. Note that event e_2 is already governed by an exponential distribution, so it remains unchanged by this process. The system in Figure 5.4b is fully Markovian.

5.2.2 Modeling and Solving a GSMDP

In Section 3.2.3 we saw that, if we intend to use model-based planning or learning methods for DT frameworks applied to physical systems, an important practical step is the estimation of the involved stochastic models. In the GSMDP case, in this step, we need to identify the stochastic models T and $\mathcal{F}$.

For every event in $\mathcal{E}$, the identification procedure for T and $\mathcal{F}$ is technically simple: as before, since the system is fully observable, T can be estimated through the relative frequency of the occurrence of each transition; and by timing this transition data, it is straightforward to fit a probabilistic model over the resulting data to obtain $\mathcal{F}$. The family of that distribution is, however, unknown *a priori*, in the general case.

Since events abstract state transitions, a representation of T and $\mathcal{F}$ over events naturally exploits the structure of the process, reducing the number of parameters that should be estimated. For example, for a set of identical robots, each with a state factor representing battery level, the same event (running low on battery, for example) would be mapped by the same change in any of those factors, and so only one temporal distribution and state transition probability would need to be specified.

As we've discussed in Section 2.2.3, the direct solution of a general GSMDP is a

difficult problem, since typical dynamic programming approaches cannot be directly applied under its non-Markovian dynamics. Younes and Simmons (2004) proposed to approximate each non-exponential $f^{\mathbf{a}}_{s,s'}$ in $\mathcal{F}$ as a *Phase-Type* distribution. These distributions govern the total time required for a continuous-time Markov chain to enter an absorbing state (a state with no outgoing transitions). Several methods exist that can generate approximate, arbitrarily good Phase-Type representations of general distributions (Osogami and Harchol-Balter, 2006).

Using such approximation methods, we can essentially replace a given non-Markovian event $e = \Phi(s, s')$ in the system with a continuous-time Markov chain with initial state s, N_p intermediate states, and absorbing state s'. This process is represented in Figure 5.4. Each of the N_p states of this Markov chain is regarded as a *phase* of the approximate distribution, and every transition in the chain is governed by an exponential distribution. The respective phases are then added as variables in the factored state space of the original model. If this replacement is possible for every event, then the approximate system is fully Markovian, allowing it to be solved as an MDP.

There are, however, limitations to this approach. An arbitrary non-Markovian distribution, with a coefficient of variation $cv = \sigma/\mu$, where μ is its mean and σ^2 its variance, requires the $\lceil \frac{1}{cv^2} \rceil$ phases to be approximated as a Generalized Erlang distribution (one such Phase-Type distribution), if $cv^2 < 0.5$. This number can quickly become unreasonably large for many processes which are characteristic of robotic systems. In particular, this affects actions with a clear minimum time to their outcome, dictated by the physical restrictions of a robot (*e.g.*, navigation actions given known initial positions), since μ can be arbitrarily large. This is also especially true of deterministically timed transitions, which, as we saw in Section 5.1, are commonplace in robotics applications.

Another relevant issue regarding the introduction of phase variables into the state space of a GSMDP model concerns the resulting exponential increase in the total number of states. If a system has N_e non-Markovian events, and each of them is to be replaced by a Markov chain with N_p phases, then there is an increase by a factor of $N_p^{N_e}$ on the size of the resulting inflated state space $\mathcal{S}'$. This can quickly become impractical for planning and learning over the approximate system. This problem can, however, be mitigated. Younes (2005) proposed the use of a "filtering" technique, which essentially forces a value of 0 for *inconsistent assignments* of phase variables (for example, any

phase different than 0 for a disabled event). For planning algorithms which make explicit use of ADDs when representing and computing value functions, this effectively avoids much of the computational pitfall associated with the exponential increase in the state space. An alternative method, which we here propose, is to simply *re-use* the same state variables to model the phases of multiple, mutually exclusive non-Markovian events (that is, events that are never enabled simultaneously).

Systems with non-Markovian events which do not admit Phase-Type approximations can still be analyzed as semi-Markovian Decision Processes (SMDPs), **but only if those events are never persistently enabled**, since memory between transitions cannot be kept. We propose a pragmatical alternative, for situations in which this is not a valid assumption, and which can be used to model events with minimum triggering times: such an event can be decomposed into a sequence of deterministically timed transitions, followed by a positive distribution (typically a "lifetime" distribution, see Figure 5.5b). The latter can then be better approximated by a Phase-Type distribution with a small number of phases. This requires the addition of intermediate observable states to the system, similar in purpose to the phases of a Phase-Type approximation, which act as "memory" for the original non-Markovian event. The length of this deterministic sequence can be adjusted to increase the quality of the approximation. Note that deterministically timed transitions are non-Markovian themselves, so the system is still an SMDP.

A GSMDP which can be formulated approximately as an SMDP, either by applying the above methodology or by keeping non-Markovian events in their original form (if they are not persistently enabled), can then be solved using Equation (2.17). This, in turn, forms a very positive practical result, since virtually all solution algorithms for MDPs can also be applied to an SMDP in this way.

5.2.3 Tracking Phase Variables

If Phase-Type approximations are used to replace non-Markovian events, by following the methodology of the preceding section, then it is important to note that any "phase" variables added to the system state are not observable variables. That is, each of the states in an acyclic Markov chains describing a Phase-Type distribution bears no physical meaning. During execution, the agent will only be able to observe this "virtual" subsystem at its initial and absorbing states. However, the approximate SMDP policy

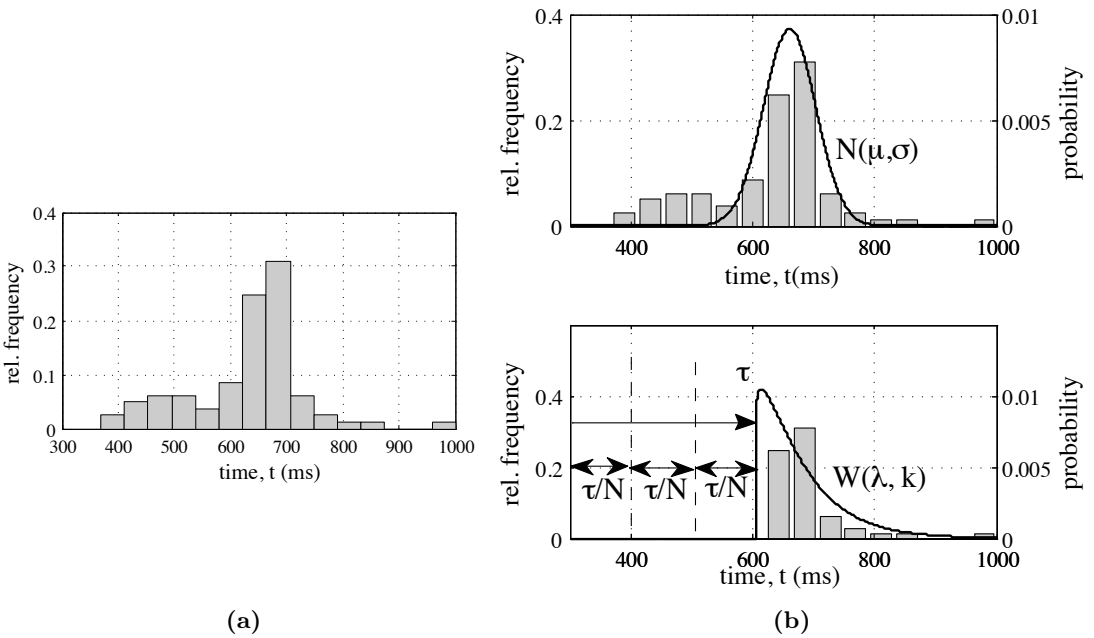


Figure 5.5: (a) Temporal distribution of the event of switching agent roles after a pass, in our experimental domain. (b) Two modeling approaches. **Top**: As a (truncated) normal distribution. Would require 238 phases for a direct Generalized Erlang approximation; **Bottom**: A sequence of deterministically timed events, followed by a Weibull distribution (2 phases). 8 states are required, in total, for the approximation. Right-tailed probability mass is discarded.

may depend on these virtual phase variables, so an agent acting according to that policy must maintain an estimate of which phase is being occupied at any given point in time.

Younes 2005 addresses this problem by propagating a belief state over phase variables in continuous-time, and using the most likely state (MLS) as a heuristic. We note, however, that agents are only supposed to act following events; and transitions between virtual phase variables do not constitute events in the original model of the system. For that reason, in practice, we only evaluate phase variables whenever an observable event occurs, and not continuously over time.

5.2.4 Effects on Communication

In Section 3.2.7.1, we have presented an example of how to implement communication in real-time while performing synchronous decision-making, in order to maintain coherency between the actions of different agents.

The differences in the communication processes required for synchronous and asynchronous decision-making can be seen in Figure 5.1. The most important of these is that, in asynchronous multiagent decision-making, we no longer need to delay decision-making until a consensus is reached on the joint state of the system. This is because, since multiple events cannot happen simultaneously, the necessary communication of their respective information is implicitly one-directional: the agent that detects an event is assumed to send that information immediately to its partners. On the receiving end, agents only have to update their joint state information with the events that they receive and proceed immediately with a new decision step. Besides eliminating extraneous delays in the decision-making process, asynchronous communication has the added advantage of effectively minimizing the number of communication episodes across the team.

We emphasize that we're assuming joint full observability of the system. As before, that means that none of the agents has access to all of the state variables, but also that the joint state can be inferred from the local information of each agent. In the case of asynchronous decision-making, this implies that any event in the system must be observed by at least one agent.

If delays are present in the communication of events, then the decision-making process is not permanently affected. If events are paired with the time at which they are detected, when they are transmitted, then any agent can ultimately reconstruct the correct joint state, even if events are detected or received out of order. This process is exemplified in Figure 5.6.

5.3 Results: Revisiting the Robotic Soccer Case Study

We will now revisit the robotic soccer case study that we have originally introduced in Chapter 3, with the purpose of comparing the performance of synchronous and event-driven approaches to decision-making in that problem. We have modeled the scenario both as a GSMDP and as a standard, synchronous MDP with configurable time-step. The resulting policies are tested both in our realistic simulator and also on the actual soccer robots.

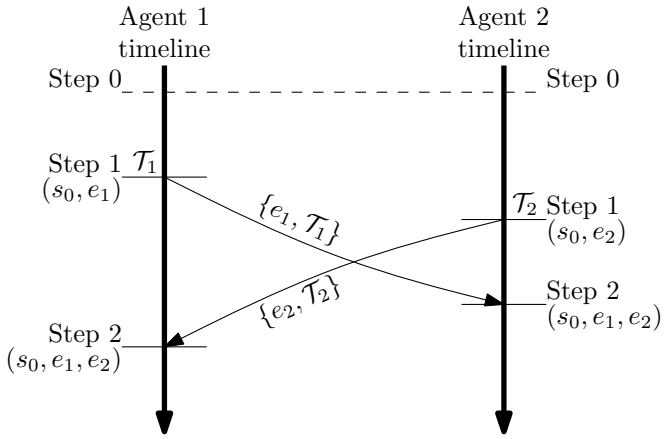


Figure 5.6: A timeline of asynchronous communication with delays, in a fully observable setting. At time $\mathcal{T}_1$ agent 1 observes event e_1, and at time $\mathcal{T}_2 > \mathcal{T}_1$ agent 2 observes e_2. At each of these instants, the respective agent sends to its partner its event information paired with a timestamp. At each decision step, each agent infers the present joint state from the sequence $(s_0, e_1, \ldots, e_n)$. Agent 2, however, only receives the information regarding e_1 after it has taken a possibly erroneous decision at $\mathcal{T}_2$. However, the timestamp information allows the agent to reconstruct the correct joint state, and at decision step 2, both agents are acting on the same information.

5.3.1 Experimental Setup

The multi-robot task that we will evaluate in this section is, essentially, a fully observable version of the Attacker-Supporter problem that was described in Section 3.2.7. We emphasize that the assumption of full observability is here made solely for the purpose of modeling the problem as a GSMDP. Strictly speaking, as we have previously noted, there are several sources of perceptual uncertainty in this problem. We expect, then, for the performance of the resulting fully-observable policy to be consequently affected by this assumption (but not to a point where it is no longer functional). However, the hypothesis that we aim to test, at this point, is that event-driven DT policies can have better performance than comparable synchronous alternatives, for teams of physical agents. For this comparison to be fair, we re-cast the original MPOMDP model of Section 3.2.7 as a multiagent MDP, which will serve as a synchronous baseline.

Given the physical restrictions of our robotic soccer testbed, slight modifications were made to the state space of our models with respect to the earlier MPOMDP formulation. The updated state space only considers half of the soccer field as the admissible playing area for the robots, as that is the layout of their physical environment

(see Figure 5.8). Note that, in the GSMDP case, this description includes a state factor that describes the phase of a non-Markovian event. The resulting model has 126 states across 4 state factors. In the MDP case, that state factor is extraneous, so the model has only 42 possible states. The full state space description can be found in Appendix A (Figure A.3).

In both the GSMDP and MDP cases, each agent has access to only a subset of the possible state factors. Namely, the Attacker robot can observe the role assignments, phase variables, and its own attacker state; the supporter robot can also observe the role assignment variable (hence, it is a globally observable factor) and its local state as a supporter. The robots communicate events over local state factors.

As in the MPOMDP of Chapter 3, there are 36 joint actions. Agents are rewarded for scoring a goal (150) and for successfully switching roles whenever obstacles are blocking the attacker (60).

Every transition was timed and modeled, either according to exponential distributions, where possible; through uniform distributions — the time of entry of the dribbling robot into one of the field partitions; or through normal distributions — the time to a role switch after a pass occurs, represented in Figure 5.5a. The latter was kept in its normal parameterization, since no concurrent events can trigger in that situation. The model was then reduced to an SMDP by replacing all uniform distributions with Phase-Type approximations. Uniform distributions can be approximated by a Generalized Erlang distribution with 3 phases (Younes and Simmons, 2004). In order to minimize the state space size, the same phase variable (*i.e.* the same state factor) was used to model all Phase-Type distributions, depending on the context. As discussed in Section 5.2.3, phase variables are not observable, but we use the QMDP heuristic to estimate their state based on the time that their respective event has been enabled.

The value iteration algorithm was used to solve the approximate SMDP.

5.3.2 Simulation Results

Part of our experimental results were gathered using our realistic robotics simulator. In an initial analysis, the abilities of the simulated robots were extended in order to allow them to more efficiently dribble and kick the ball, so that their reactivity to events is not affected by their physical limitations when acting. Figure 5.7 compares real-time profiles of the system, under these conditions, when executing an event-driven GSMDP solution

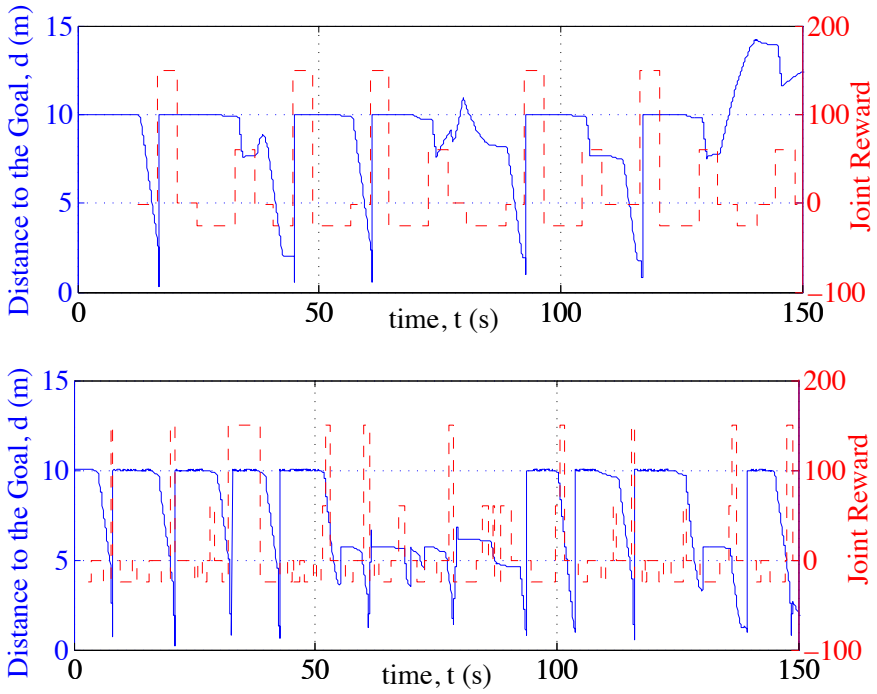


Figure 5.7: Simulated results. Distance from the ball to the goal (blue, solid) and accrued joint reward (red, dashed) over time. Top: using an MDP model with fixed time-step $\Delta = 4\,s$; Bottom: using the GSMDP formulation of the same problem. Jumps in reward correspond to new decision steps. Rewards of 150 correspond to a shooting actions, and those equal to 60 correspond to passing instances in which robots switch roles. Whenever a goal is scored (the distance tends to 0), the ball is reset to its original position. Here, the robots could control and kick the ball efficiently.

and a discrete-time multiagent MDP solution with a fixed time-step. These execution profiles are characterized by the distance between the ball and the goal, alongside the reward associated with the joint state and action, accrued at decision instants. While the MDP system is committed to performing the same action throughout the duration of the time-step, the GSMDP reacts asynchronously to events, which eliminates any idle time in the actions of the robots, resulting in more frequently scored goals.

5.3.3 Real Robot Results

The performance of the synchronous (fixed time-step) and event-driven (GSMDP) approaches to this problem in the real team of robots was quantitatively evaluated by testing a synchronous MDP solution with a series of different fixed time-steps, as well as the GSMDP solution. The performance metric for this comparison is the time be-

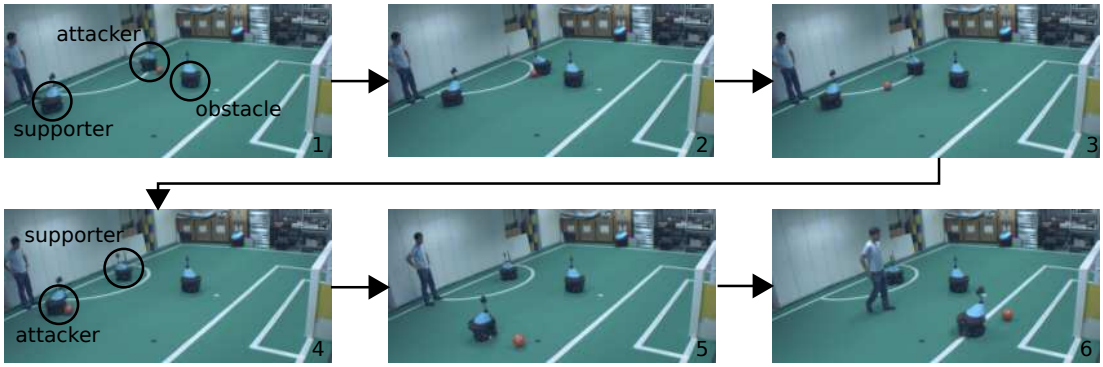


Figure 5.8: Sequence showing two robots cooperating in order to avoid an obstacle and score a goal (from left to right, top to bottom), in our experimental setup. The team was executing an event-driven GSMDP policy. The ability of the robots to handle the ball individually is very limited, which makes this type of cooperation necessary. In image 4 a role switch has occurred, after the successful passing of the ball. A video of this experiment can be seen at: `http://users.isr.ist.utl.pt/~jmessias/PhDthesis`.

tween consecutive goals using each model. The results are shown in Figure 5.9. The amount of trials that could be run on the real robots was limited by total time: the average sample size is 5 scored goals for each model (9 for the GSMDP and the best performing MDP). In order to provide further statistical validity for these real robot results, simulated trials were run under equivalent conditions (considering all actuation limitations), in runs of 120 seconds each, to a total of 50 goals per model (box plot in Figure 5.9a).

The average and median time between goals was shorter with the GSMDP solution than with any of the synchronous MDPs. The time-step of the synchronous models was shown to have a significant, non-monotonic effect on performance. The best MDP model, with a time-step of 0.4 seconds, underperformed the GSMDP model both in the real robot trials (one-way ANOVA, $p = 0.063$), and in the corresponding simulated trials ($p = 0.079$). For time-steps below this value, agents acted on outdated information, due to communication/processing delays, and had difficulty in switching roles (note from Figure 5.5b that the minimum time for a role switch during a pass is also $\sim 0.4s$). For larger time-steps, loss of reactivity and the corresponding idle time in the system also lowered the resulting performance. The average duration of decision steps (and communication period) with the GSMDP model was $1.09s$. Since the frequency of communication episodes for synchronous MDP models is $2/T$ (Figure 5.1), this implies a reduction in communication usage of 81.7% with respect to the best MDP model.

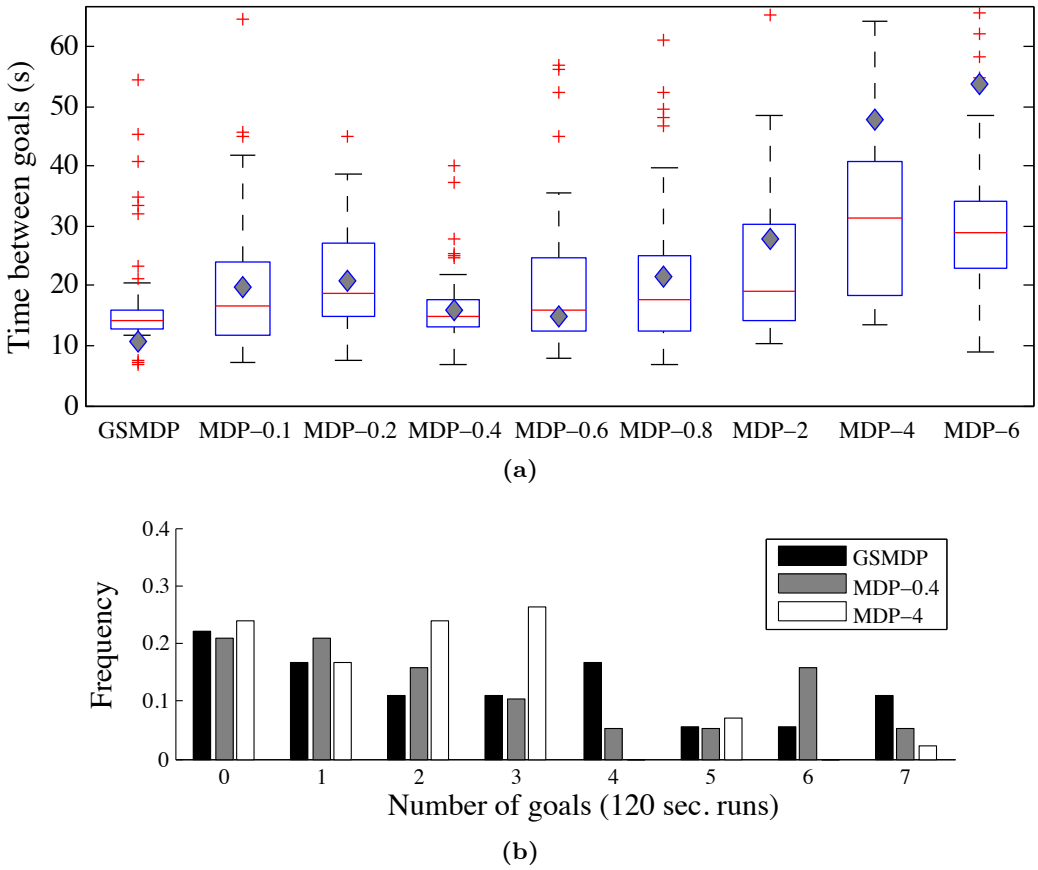


Figure 5.9: Performance of GSMDP / MDP models. Synchronous models are labeled as MDP-T, with T the decision period (seconds). (a) Median time difference between goals, on the real robot trials (diamond markers). Equivalent simulated trials are represented in the underlying box plot. (b) Frequency of goals per each trial, for the GSMDP, and the best and worst MDP models ($0.4s$, $4s$, respectively). Trials with 0 goals are indicative of random system failures.

Random system failures, occurring mostly due to robot navigation problems, or unmodeled spurious effects, were independent of the modeling approach (Figure 5.9b).

Figure 5.8 shows an image sequence of a typical trial. A video containing this trial, and showcasing the differences in behavior between the synchronous MDP and GSMDP approaches to this problem, both in the real and simulated environments, can be found at: `http://users.isr.ist.utl.pt/~jmessias/PhDthesis`.

5.4 Summary

There are non-trivial and often overlooked problems involved in the application of inherently discrete models such as MDPs to dynamic, physical systems which are naturally continuous.

In this chapter, we showed how discrete models of multi-robot systems are not fully Markovian, and how the most common work-around (which is to assume synchronous operation) impacts the performance of the system. We discussed how the GSMDP framework fits the requirements for a more efficient, event-driven solution, and the methodologies required for GSMDPs to be implemented in practice. We have noted that robotics systems typically experience events that are governed by temporal distributions that are not amenable to Phase-Type approximations, and we have presented pragmatical alternatives. We have shown how communication can be implemented in practice, in an event-driven multiagent system. Ultimately, the application of the methods described in this chapter has resulted in the first successful application of the GSMDP framework to a decision-making problem in a team of real robots.

We have also presented and discussed a novel definition for the dynamics of decision-theoretic, event-driven systems, which shares a closer relationship to DES formalisms, without losing generality. In the next chapter, we will describe how that definition can be exploited to extend event-driven multiagent decision-making to partially observable domains.

Chapter 6

Asynchronous Multiagent Decision-Making under Partial Observability

6.1 Introduction

In the preceding chapter, we began to explore the problem of event-driven decision-making for teams of physical agents. To that end, we have studied and applied the DT frameworks which are most appropriate to that domain. In particular, we saw that the GSMDP framework performs well in practice, but that it does not account for the problem of partial observability.

On the other hand, as we saw in Chapter 3, physical agents are commonly subject to uncertainty in their observations. Mobile robots, in particular, are subject to partial observability, not only due to the limited coverage of their sensors, but also due to noise in their respective measurements. It stands to reason, then, that being able to deal with perceptual uncertainty should be a requirement for any DT approach that aims at being applicable to general decision-making problems in the domain of robotics.

These considerations establish the motivation for the work presented in this chapter: to extend the DT frameworks and methodologies described so far, enabling them to model event-driven physical multiagent systems with uncertain observations.

To do so, at first, we isolate this problem from that of modeling the effect of continuous time, which was studied in Chapter 5. Even in discrete-time scenarios, the existing

DT frameworks that that take into account partial observability are not completely appropriate to model multiagent asynchronous systems.

To address this problem, we propose Event-Driven MPOMDPs, as an extension to the dynamics of discrete multiagent POMDPs which is specifically suited to asynchronous operation. Our framework explicitly models uncertainty over event detections, and considers the possibility of false positive and false negative detections, both during planning and also during execution.

In our approach, agents must react to events, which are detected locally, and asynchronously, by each agent. Through the assumption of free communication, each local event triggers a observation that is shared by the team. Since multiple events cannot occur simultaneously, this means that the total number of observations in this model grows *linearly* in the number of agents, instead of exponentially. This positive result allows these methods to scale better to larger scenarios, while retaining MPOMDP functionality.

We prove that the optimal value function is piecewise linear and convex, allowing us to extend a point-based solver to the event-driven setting. We also show how belief states can be updated in run-time, while considering the possibility of false negative event detections. Moreover, on a practical note, we provide a methodology to represent Event-Driven MPOMDPs in a factored, graphical format, which simplifies the practical implementation of these methods in large multiagent systems. We then consolidate the proposed methods through simulated results, comparing the performance of our event-driven models to that of equivalent synchronous MPOMDPs.

Having established the functionality of our framework in discrete time, we will describe how it can be extended to continuous-time domains, drawing from the methodologies that were analyzed in Chapter 5. The end result of this effort is a single, coherent framework that satisfies the most important requirements for decision-making under uncertainty for teams of real robots.

6.2 Event-Driven MPOMDPs

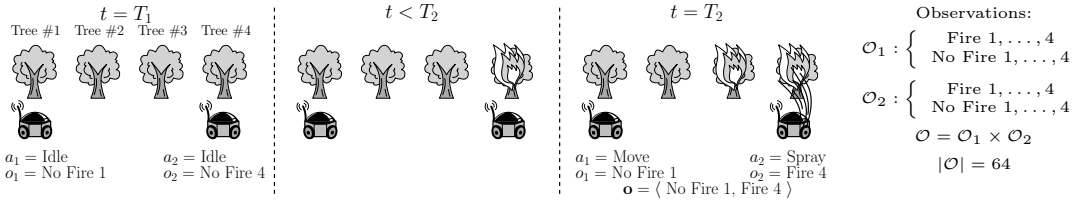
In this section we propose a novel modeling approach to multiagent decision-making under partial observability. We will first provide an overview the operation of our proposed asynchronous MPOMDPs for multi-robot systems, and discuss the advantages that it

provides with respect to synchronous formulations. We then formalize our proposed approach, and discuss practical issues related to its implementation.

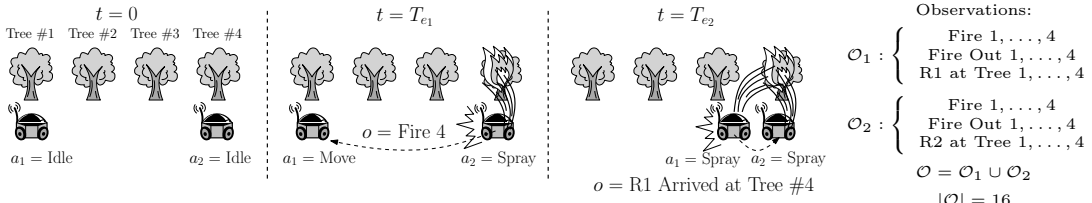
6.2.1 Synchronous vs. Asynchronous Execution in Multiagent Systems with Partial Observability and Free Communication

In MPOMDPs, by definition, each agent is expected to know the observations of all of its teammates, before taking a new action. In practice, as we saw in Section 3.2.7.1, this implies that agents must only communicate their observations to each other at the end of each step, so as not to exclude any potential information. However, these observation dynamics are implicitly synchronous – a communication step cannot be unilaterally carried out by any single agent, since it must wait for all other agents to respond back with their own information. The communication of observations, therefore, is typically executed at a fixed frequency. One of the well-known consequences of this schema, which is often overlooked as being inevitable, is that exponentially many possible combinations of individual observations can be shared amongst the team, which is an evident computational drawback to any planning algorithm which iterates over possible joint observations. This constitutes one of the most significant (if not the most significant) hindrances to the scalability of MPOMDPs. The operation of a synchronous MPOMDP is illustrated in Figure 6.1a.

In contrast, we see that event-driven execution, represented in Figure 6.1b, is once again more intuitive. As we have seen in Chapter 5, in this case, decisions are triggered by changes in the state of the system. In the partially observable case, observations represent events, and are, therefore, also asynchronous. This means that, when an observation is detected *locally* by an agent, it can be promptly communicated to the rest of the team. The detecting agent can select an appropriate action immediately, and other agents may do so as soon as they receive that information. This communication strategy is essentially what we had proposed in Section 5.2.4, but exchanging *observations of events* instead of the events themselves, since these are not directly accessible. Since observations are not synchronized, the concept of *joint* observation is no longer applicable; all observations are jointly experienced, even if they originate from a single agent. *Local* observation spaces are still definable, and represent the set of events that each agent can detect (see Figure 6.1b, right). An important implication of this approach is that the observation space for the team is now the *union* of all local observation spaces.



(a) Synchronous



(b) Asynchronous

Figure 6.1: A graphical representation of the dynamics of a synchronous MPOMDP (a) and of an Event-Driven MPOMDP (b). Two agents must extinguish forest fires. In (a), decisions are taken periodically, and agents receive simultaneous observations. Agents cannot react instantaneously when a fire breaks out between instants $t = T_1$ and $t = T_2$; In (b), at instants $t = T_{e_1}$ and $t = T_{e_2}$, agents 1 and 2 respectively detect events, and share that information with their partner, triggering new decision episodes for the team. To the right of each example, we describe the corresponding observation space. In the event-driven case, this is the union of local observation spaces, and consequently, common elements are not repeated.

As a function of the number of agents, then, there are at most linearly many possible observations to be considered, corresponding to the sum of the possible detections of each of them. This replicates a well known result in the theory of DES, that set of possible events for the *composition* of multiple, concurrently operating automata is also the union of their respective, local event sets (Cassandras and Lafortune, 1999). This event-driven interpretation of MPOMDP dynamics forms the core of our approach.

6.2.2 Formal Definition

Before formally defining a model which operates according to the requirements stated above, we will need to slightly extend our earlier definition of "events" (Definition 2.2.2), to account for the presence of uncertainty over observations.

Definition 6.2.1. *A* partially observable event *is an event $e \in \mathcal{E}$ over $\mathcal{S}$, as per Definition 2.2.2, which also verifies $O(u, \mathbf{a}, u', \mathbf{o}) = O(v, \mathbf{a}, v', \mathbf{o})$, $\forall \mathbf{a} \in \mathcal{A}$.*

Note that this definition, as in Definition 2.2.2, also considers the possibility of mod-

eling continuous-time distributions over transition instants. We will use this property in Section 6.5; for now, in order to remained focused on the problem of partial observability, we assume that $f_e^{\mathbf{a}}(t) = \lambda e^{-\lambda t}$ for all $e \in \mathcal{E}$,$\mathbf{a} \in \mathcal{A}$ and for some $\lambda \in \mathbb{R}^+$. That is, all transitions are governed by exponential distributions with the same rate. As we saw in Sections 2.2.1 and 2.2.2, this allows us to analyze the operation of this event-driven system in discrete time, and with fully Markovian dynamics, which only describe the system at event instants. Therefore, we can omit the influence of $f_e^{\mathbf{a}}(t)$, and consider that it is implicitly included in the discrete discount factor γ (which is actually a function of λ). This does not assume that events are synchronous.

In contrast to standard POMDP models, we include both s and s' in O as we are interested in observing characteristics of *state transitions*, as opposed to characteristics of states. As such, we have $O(s, \mathbf{a}, s', \mathbf{o}) = \Pr(\mathbf{o} \,|\, s, \mathbf{a}, s')$, allowing the observation model to be indexed through events, as $O(\mathbf{a}, e, \mathbf{o})$.

Noisy event detection processes are typically characterized through their susceptibility to *false negative* and *false positive* errors. False negative detections, by definition, are not observable, but their incidence is clearly relevant to the decision making process. In order to describe their stochasticity, we will model false negative detections as a *virtual*, symbolic observation[1]. This virtual detection will be represented as $o_f \in \mathcal{O}$, and its probability of occurrence for $\langle a, e \rangle$ as $O(\mathbf{a}, e, o_f)$. We emphasize that this observation is never received by an agent during plan execution; it is merely a theoretical construct that will facilitate the definition and analysis of our framework.

False positive detections, on the other hand, can be modeled as the emission of event detections when the system actually remains in the same state. If an observation $o \in \mathcal{O} \backslash o_f$ is considered a false positive detection, then $O(\mathbf{a}, e, o) > 0$ with $e = \Phi(s, s)$ for some $s \in \mathcal{S}$. Notice that the converse is not necessarily true, since we might want to model the true positive observation of same state transitions (for example, if a robot executes a self-diagnostic procedure to check for hardware problems, but no errors are found). The underlying event to a false positive detection can also be considered as a "virtual", idle event (as opposed to the virtual *observation* o_f for the false negative case), but won't require any special treatment from a formal standpoint.

[1] This is a similar idea to labeling unobservable events in DES with empty strings (Cassandras and Lafortune, 1999).

If an actual event is detected, there can also be uncertainty over the label that is associated to it by the detection mechanism. That is, a true event may be incorrectly classified, informing agents that the state of the system has changed, but misleading them as to *how* it changed (*e.g.* a facial recognition program signaling that a person has been detected, but identifying that person incorrectly). These observations can also be modeled in our framework, in which case, the joint observation function, in tabular form, resembles a "confusion" matrix over the events occurring in the system and their associated label in $\mathcal{O}$[1].

We will now formally introduce our framework:

Definition 6.2.2. *An Event-Driven Multiagent POMDP is a tuple* $\langle d, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{E}, T, O, R \rangle$ *where:*

d *is the number of agents;*

$\mathcal{S} = \mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \times \mathcal{X}_k$ *is the (factored)* state space*;*

$\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times \ldots \times \mathcal{A}_d$ *is the set of* joint *actions;*

$\mathcal{O} = \mathcal{O}_1 \cup \mathcal{O}_2 \cup \ldots \cup \mathcal{O}_d \cup o_f$ *is the space of observations (or* event detections*) for the team, where* $\mathcal{O}_i$ *is the space of possible observations of each agent, and* o_f *represents false negative event detections;*

$\mathcal{E}$ *is a set of events over* $\mathcal{S}$ *(Definition 2.2.2);*

$T : \mathcal{S} \times \mathcal{A} \times \mathcal{E} \to [0, 1]$ *is the* transition function*, such that* $T(s, \mathbf{a}, e) = \Pr(e \mid s, \mathbf{a})$ *for* $e \in \mathcal{E}$, $s \in \mathcal{S}$, $\mathbf{a} \in \mathcal{A}$*;*

$O : \mathcal{A} \times \mathcal{E} \times \mathcal{O} \to [0, 1]$ *is the* observation function*, such that* $O(\mathbf{a}, e, o) = \Pr(o \mid \mathbf{a}, e)$ *for* $o \in \mathcal{O}$, $e \in \mathcal{E}$, $\mathbf{a} \in \mathcal{A}$*;*

This definition is equally valid for single agent models, setting $d = 1$. We note that the most important distinction of an Event-Driven MPOMDP, with respect to a "synchronous" MPOMDP, resides in the form of its observation space, which is taken as the union of all local observations; and explicitly accounts for uninformative, false negative detections. In the subsequent section, we will see how this latter property affects decision-making.

[1] In this case, there can also be "false positive" and "false negative" identifications of each event due to the association with incorrect labels. In this work, we reserve those terms to mean, exclusively and respectively, detection of inexistant events, and inability to detect occurring events.

6.2.3 Decision-Making with Partially Observable Events

An event-driven system subject to uncertainty in the detection of its events raises a fundamental problem for decision-making – if all agents fail to detect the occurrence of an event, agents will not be able to change their actions accordingly. Nevertheless, if the system experiences a state transition, any consequent instantaneous reward must be accounted for, even if the agents are not able to perceive that transition. If this wasn't the case, arbitrary state-action pairs could be visited between two decision episodes, but would be irrelevant to the decision making process, as long as the probability over resulting states at the subsequent step remained the same. For robotic systems, this means that undesirable or potentially harmful $\langle s, \mathbf{a} \rangle$ could be visited[1].

Given a probabilistic description of false negatives, a straightforward approach to account for their effect is consider that decision episodes only occur when an event is detected (see Figure 6.2a), *i.e.* when any $o \in \mathcal{O} \backslash o_f$ is received. False negative detections of events, and their respective intermediate transitions, could then be folded into the "apparent" transition model of the system as per the infinite series:

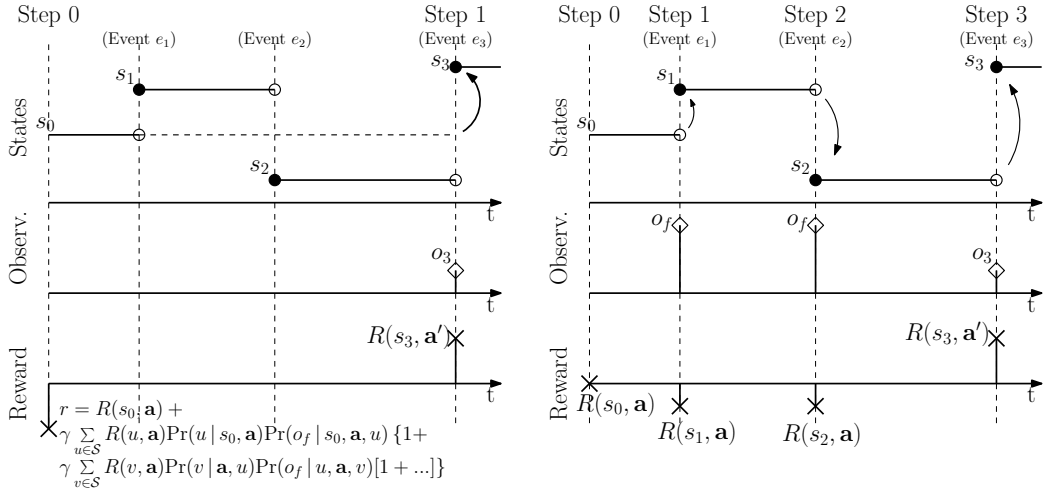
$$T(s, \mathbf{a}, s') = \Pr(s' \,|\, \mathbf{a}, s) + \sum_{u \in \mathcal{S}} \Pr(u \,|\, \mathbf{a}, s) \Pr(o_f \,|\, s, \mathbf{a}, u) \Pr(s' \,|\, \mathbf{a}, u) \quad + \\ \sum_{u \in \mathcal{S}} \Pr(u \,|\, \mathbf{a}, s) \Pr(o_f \,|\, s, \mathbf{a}, u) \sum_{v \in \mathcal{S}} \Pr(v \,|\, \mathbf{a}, u) \Pr(o_f \,|\, u, \mathbf{a}, v) \Pr(s' \,|\, \mathbf{a}, v) + \ldots$$

This explicitly considers all possible sequences of intermediate states $(u, v, \ldots)$ that may be visited unknowingly due to o_f. The reward for each $\langle s, \mathbf{a} \rangle$ can also be adjusted as:

$$\begin{aligned} R'(s, \mathbf{a}) = & R(s, \mathbf{a}) \quad + \\ & \gamma \sum_{u \in \mathcal{S}} R(u, \mathbf{a}) \Pr(u \,|\, s, \mathbf{a}) \Pr(o_f \,|\, s, \mathbf{a}, u) \Big\{ 1 + \\ & \gamma \sum_{v \in \mathcal{S}} R(v, \mathbf{a}) \Pr(v \,|\, \mathbf{a}, u) \Pr(o_f \,|\, u, \mathbf{a}, v) \Big[1 + ... \Big] \Big\} \end{aligned}$$

However, this approach is not extendable to the continuous-time considerations of

[1] In some continuous-time models, such as in SMDPs (White, 1976), intermediate state transitions are also allowed. However, a continuous *reward rate* can then be assigned to intermediate states. Furthermore, in that case, since all transitions are observable, decisions can be skipped by design. In our case, decisions are missed due to the limitations of the agents.



(a) Driven by event detections **(b)** Driven by events

Figure 6.2: The effect of false negative detections on the transition dynamics of a decision-making process. In this example, action $\mathbf{a}$ is selected at step 0; events $e_{\{1,2,3\}}$ occur, but agents fail to detect e_1 and e_2. The resulting timelines of states, observations, and rewards are shown, according to different interpretations: (a) If the steps of the process are driven by *detected* events, then the only transition that occurs in this timeline and that is modeled in T, is from s_0 to s_3 (represented by an arc). The probabilities of intermediate unobserved events can be collapsed into $T(s_0, \mathbf{a}, s_3)$, and their reward into $R(s_0, \mathbf{a})$; (b) If, instead, we consider that *any* event results in a new step of the decision-making process, then the virtual observation o_f is emitted by e_1 and e_2, and agents are unable to change their actions at the corresponding steps. Since all transitions map to events, rewards $R(s_1, \mathbf{a})$, $R(s_2, \mathbf{a})$ are still accounted for.

Chapter 5[1], and it is only practical for infinite-horizon policies, otherwise the transition and reward models would not be time-homogeneous.

An alternative approach is to consider that all events, and not their detections, trigger decision episodes (see Figure 6.2b). If agents experience false negative detections, then they are forced to select the same actions in consecutive steps, which can be interpreted as having "missed" an opportunity to change actions. From a planning perspective, this means that all state changes in the system, even those that are unoberved, are evaluated in separate dynamic programming steps. This approach is useful because it allows us to consider finite-length sequences of unobserved transitions when planning, which is fundamental if we wish to apply planning algorithms that approx-

[1]The inclusion of probability density functions over the firing time of each transition would imply that these infinite series would contain nested integrals, which would become analytically intractable in the general case.

imate stationary solutions through a finite number of dynamic programming backups (as we will see in Section 6.3.1). This is the approach that we take in this work.

We can formalize this mechanism of action persistence by defining an "action constraint" function $\mathcal{C} : \mathcal{A} \times \mathcal{O} \rightarrow PS(A)\backslash\emptyset$, where $PS(A)$ is the power set of A, which returns, for each pair $\langle \mathbf{a}, o \rangle$, a constrained action set $\mathcal{C}(\mathbf{a}, o) \subseteq \mathcal{A}$. This set represents the joint actions which are available to the agents at the onset of a decision episode, given that, at the previous step, the team of agents executed $\mathbf{a}$ and observed o. That is, at each step n, agents select the best available action from the set $\mathcal{C}(\mathbf{a}_{n-1}, o_{n-1})$, instead of selecting actions from $\mathcal{A}$. Then, in order to model the effect of false negative detections on decision-making, we will be interested in $\mathcal{C}(\mathbf{a}, o)$ with the following form:

$$\mathcal{C}(\mathbf{a}, o) = \begin{cases} \{\mathbf{a}\} & \text{if } o = o_f \\ \mathcal{A} & \text{if } o \in \mathcal{O}\backslash o_f \end{cases} \tag{6.1}$$

Finally, we note that, for planning purposes, false positives detections can be trivially handled in the same way true positives, since their underlying events are essentially observable (albeit the event itself does not change the system). In this way, we explicitly reward actions that are elicited by false positive detections (typically with negative values). This implicitly promotes policies that minimize the susceptibility of the agents to false positive detections.

6.2.4 Jointly Observed Events

In some multi-robot applications, more than one robot may contribute to the observation of the same relevant features of the environment (for example, in problems of active cooperative perception). For those scenarios, our definition of the joint observation space of an Event-Driven MPOMDP as the union of the possible individual observations for each agent may seem restrictive. In our framework, since all observations are jointly experienced, if more than one agent can broadcast the same event detection (that is, if $\mathcal{O}_i \cap \mathcal{O}_j \neq \emptyset$ for $i \neq j$), then the number of agents that simultaneously emit that observation, and their respective identity, is indifferent to the decision-making process. This would not be an accurate representation of the distributed sensing process, since, from a Bayesian perspective, the simultaneous observation of an event by more than one agent should reduce the amount of uncertainty that is associated with that detection.

We note, however, that this comes as a consequence of having assumed asynchronous dynamics, not only with respect to events, but also to their associated detections. Even though they may be arbitrarily close in time, different observations of the same event by more than one agent will never be strictly simultaneous, unless agents are explicitly synchronized (and, for those cases, the use of our event-driven framework is not appropriate). Consequently, under our framework, a jointly detected event should be represented as a sequence, rather than as a combination, of individual observations (see Figure 6.3a). For each such event, the state of the system only changes once, producing the first element of its sequence of detections. Subsequent observations can be associated with idle events (the event that preserves the state of the system). This, however, considers that each agent can observe the same event multiple times. When this is unrealistic, additional states can be included in S, that encode whether or not each agent has already observed the given event.

Since each event detection implies a new decision episode, modeling joint, near-simultaneous event detections as sequences of observations naturally increases the problem horizon when compared to a synchronous solution. The trade-off is that, at each step of the decision-making process up to the problem horizon, there is a much smaller number of possible observations to consider, in an asynchronous setting.

In the general case where agents have heterogenous observation functions, $\mathcal{O}_i \cap \mathcal{O}_j = \emptyset$ for $i \neq j$. However, we do not formulate this property as a requirement in our framework, since it also allows for a different approach to the problem of joint event detection: if during execution, the data from multiple distributed sensors is fused into a single coherent estimate, by any mechanism that is external to the decision-making process, then an observation in an Event-Driven MPOMDP can actually represent the consensus of multiple agents as to the occurrence of an event. For example, if multiple cameras with overlapping fields-of-view detect an object of interest, then, before broadcasting this detection to all other agents, the cameras can consolidate their observations with their neighbors through a suitable filtering mechanism, in order to determine if it is the same object, and only then broadcast this fused information for decision-making purposes (we use this approach in our implementation of a multiagent surveillance system, described in Chapter 7). This, however, introduces a delay between the occurrence of an event and the selection of an appropriate action, which should be expected during execution.

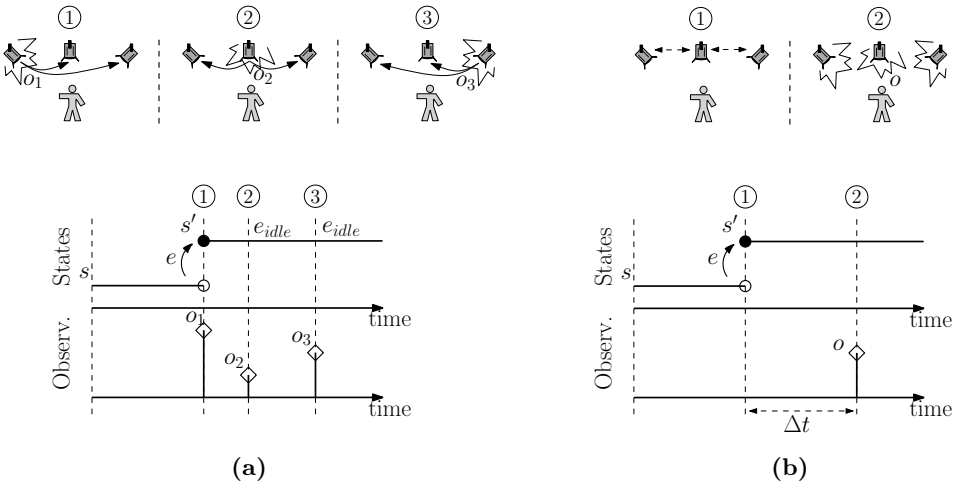


Figure 6.3: Modeling the joint detection of events. A set of three cameras jointly detect the presence of a person (an event $e = \Phi(s, s')$). We show the respective timelines of states and observations. (a) Joint detections can be modeled as a sequence of individual observations. The first detection, o_1, is associated with the occurrence of e. Subsequent observations o_2, o_3 are associated with the idle event $e_{idle} = \Phi(s', s')$. (b) Alternatively, if the data from different cameras is fused outside of the decision-making loop, then, after a communication and processing delay Δt, a single observation o is emitted, representing the consensus of different sensors over the occurrence of the event.

6.2.5 Factored Graphical Representations

In Section 2.4.3, we have reviewed the use of factored models in MDPs and their related frameworks, which has proven itself to be often indispensable when approaching large-scale problems. However, a factored 2-TDBN assumes that all variables evolve simultaneously when a state transition is experienced. This is not necessarily true for an asynchronous system, since events may be *local*, in the sense that they only elicit transitions in a subset of state factors.

To clarify, we will now return to our simple scenario in Figure 6.1b. Suppose that we have a factored state space $\mathcal{S} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_4$ (one for each tree) where each $x_i \in \mathcal{X}_i$ can take on binary values (indicating fire), and that each tree is sufficiently spread apart that it is reasonable to assume that the probability of it catching fire depends only on its own previous state. If $s = \langle x_1, \ldots, x_4 \rangle$ and $s' = \langle x'_1, \ldots, x'_4 \rangle$, we would then like to have:

$$\Pr(s'|s) = \Pr(x_1|x'_1)\Pr(x_2|x'_2)\Pr(x_3|x'_3)\Pr(x_4|x'_4),$$

since that would greatly simplify the problem of defining the transition model, as one

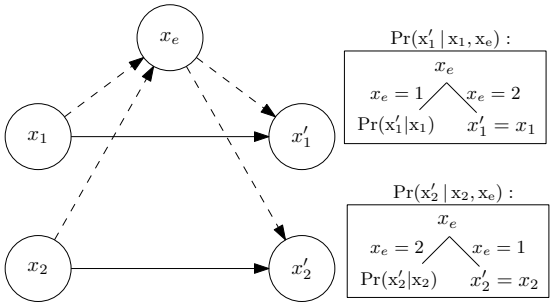


Figure 6.4: An example of a Dynamic Bayesian Network for the state space dependencies of an event-driven model. These state factors are conditionally independent of each other given $x_e \in \mathcal{X}_e$, a virtual factor which models their mutual exclusivity. The values that this factor can take are labels ("1", "2") that identify one of the remaining state factors in the problem (x_1, x_2), *allowing it to change* after an event. This factor will be marginalized out when computing the full transition model. Examples of decision diagrams for the CPD of either state factor are also shown. Note their dependency on $x_e \in \mathcal{X}_e$: if $x_e = i$, the CPD of factor x_i is $\Pr(x'_i|x_i)$, the same as if it were independent; if $x_e \neq i$, the factor keeps the same value ($x'_i = x_i$). $\Pr(x_e \,|x_1, x_2)$ is the prior probability that each factor will change in the next event.

of defining 2×2 probability tables. We can quickly see that the above expression is *not* valid for an event-driven model, since the event of "catching fire" is local to each of the state factors — there is 0 probability of more than one tree catching fire simultaneously. There is, therefore, an underlying dependency between all state factors due to their mutual exclusivity, since a change in one factor must explicitly inhibit undesired changes in other factors. When representing an event-driven model in a factored format, all factor variables are coupled in a single step.

In order to mitigate this problem, and take advantage of the representational tools which are available to factored models, we can isolate the inherent dependencies between all variables due to possible mutual exclusivity, through the addition of a new state factor into the state space description. This factor allows us to represent the probability that *each other* factor will change, given the current state. This process is shown in Figure 6.4.

This step, together with the ability to write a trivial decision diagram as those shown in Figure 6.4, enables us to write the CPDs for each of our trees *as if* they were independent. This extraneous factor can then be *marginalized* out (see (Pearl, 1988) for the detailed process), and we recover our full transition model, without altering the size of the problem.

6.3 Solving Event-Driven MPOMDPs

Having defined a framework for Event-Driven MPOMDPs, we will show in this section that these models retain the necessary properties that allow them to be solved through dynamic programming approaches. We will then present an example of how to modify a typical POMDP solution algorithm to allow it to provide (optimal or approximate) policies for our proposed framework.

6.3.1 Dynamic Programming

We can show that a value function for an Event-Driven MPOMDP in the presence of action constraints is still PWLC, which enables the use of dynamic programming techniques to calculate (or approximate) an optimal policy.

Theorem 2. *For an Event-Driven MPOMDP, and for finite n, the optimal value function V_n^* can be written as:*

$$V_n^*(b) = \max_{\upsilon_n \in \Upsilon_n} \upsilon_n \cdot b \quad ,$$

where Υ_n is a set of $|S|$-dimensional vectors.

Proof. Given a constraint-generating function $\mathcal{C}$, any policy in an Event-Driven MPOMDP is subject to the following restrictions:

$$\begin{cases} \pi_n(b_n) \in \mathcal{A} & \text{if } n = 0 \\ \pi_n(b_n) \in \mathcal{C}(\pi_{n-1}(b_{n-1}), o_{n-1}) & \text{if } 0 < n \leq h \end{cases} ,$$

where b_n and o_n are, respectively, the belief state and the observation received at the n-th step[1]. Taking these restrictions into consideration, the Bellman backup for this

[1] In this analysis, and in order to maintain the notation consistent throughout this thesis, we consider that n represents the number of steps *taken*. That is, the system begins its execution at $n = 0$. Note that many authors commonly represent n as the number of steps *left to take* (*e.g.* (Kaelbling et al., 1998)). Both options are equally valid from a theoretical standpoint.

model can be written as:

$$V_n^*(b) = \max_{\mathbf{a}\in\mathcal{A}} \Big\{ \sum_{s\in S} b(s)R(s,\mathbf{a}) \quad + \\ \gamma \sum_{\substack{s\in\mathcal{S},o\in\mathcal{O}\\ e\in E(s,\mathbf{a}),}} b(s)O(\mathbf{a},e,o)T(s,\mathbf{a},e) \max_{\mathbf{a}'\in\mathcal{C}(\mathbf{a},o)} Q_{n+1}^*(b^{\mathbf{a},o},\mathbf{a}')\Big\}, \tag{6.2}$$

where $b^{\mathbf{a},o}$ is the updated belief state according to $\langle \mathbf{a}, o\rangle$.

For $\mathbf{a}\in\mathcal{A}$, $o\in\mathcal{O}$, let $H^{\mathbf{a},o}$ be a $|\mathcal{S}|\times|\mathcal{S}|$ matrix with

$$\begin{aligned}[H^{\mathbf{a},o}]_{i,j} &= \Pr(s_i|s_j,\mathbf{a})\Pr(o|\, s_j,\mathbf{a},s_i) \\ &= T(s_j,\mathbf{a},\Phi(s_j,s_i))O(\mathbf{a},\Phi(s_j,s_i),o) \quad .\end{aligned}$$

The Bellman backup is then, in vectorial form:

$$V_n^*(b) = \max_{\mathbf{a}\in\mathcal{A}} \Big\{ r^{\mathbf{a}}\cdot b + \gamma \sum_{o\in\mathcal{O}} \mathbf{1}^T H^{\mathbf{a},o} b \max_{\mathbf{a}'\in\mathcal{C}(\mathbf{a},o)} Q_{n+1}^*(b^{\mathbf{a},o},\mathbf{a}')\Big\}, \tag{6.3}$$

where $r^{\mathbf{a}}$ denotes the $\mathbf{a}-$th column of $R(s,\mathbf{a})$. Also in this notation, the belief update step is:

$$b^{\mathbf{a},o} = \frac{H^{\mathbf{a},o}b}{\mathbf{1}^T H^{\mathbf{a},o}b} \quad , \tag{6.4}$$

where $\mathbf{1}_i = 1$, $i = \{1\ldots|\mathcal{S}|\}$ (a vector of ones).

At the final decision step, $h-1$, we have that:

$$V_{h-1}^*(b) = \max_{\mathbf{a}\in\mathcal{A}} r^{\mathbf{a}}\cdot b, \tag{6.5}$$

And therefore V_{h-1}^* is clearly PWLC with $\Upsilon_{h-1} = \{r^{\mathbf{a}}\,|\,\mathbf{a}\in\mathcal{A}\}$. Inductively, at step $n+1$:

$$V_{n+1}^*(b^{\mathbf{a},o}) = \max_{v_{n+1}\in\Upsilon_{n+1}} v_{n+1}\cdot b^{\mathbf{a},o} \quad .$$

Also, since $V_{n+1}^*(b^{\mathbf{a},o})$ is PWLC iff $Q_{n+1}^*(b^{\mathbf{a},o},\mathbf{a}')$ is PWLC:

$$Q_{n+1}^*(b^{\mathbf{a},o},\mathbf{a}') = \max_{q_{n+1}^{\mathbf{a}'}\in\mathcal{K}_{n+1}^{\mathbf{a}'}} q_{n+1}^{\mathbf{a}'}\cdot b^{\mathbf{a},o} \quad , \tag{6.6}$$

where $\mathcal{K}_n^{\mathbf{a}}$ is a set of $|\mathcal{S}|-$dimensional vectors. Taking this form for the Q-value, and

substituting (6.4):

$$Q^*_{n+1}(b^{\mathbf{a},o}, \mathbf{a}') = \max_{q^{\mathbf{a}'}_{n+1} \in \mathcal{K}_{n+1}} \frac{(q^{\mathbf{a}'}_{n+1})^T H^{\mathbf{a},o} b}{\mathbf{1}^T H^{\mathbf{a},o} b} \quad .$$

Let

$$q^{*,\mathbf{a}'}_{n+1} | \, b, \mathbf{a}, o = \operatorname*{arg\,max}_{q^{\mathbf{a}'}_{n+1} \in \mathcal{K}_{n+1}} \left\{ (q^{\mathbf{a}'}_{n+1})^T H^{\mathbf{a},o} b \right\} \quad . \tag{6.7}$$

Then,

$$Q^*_{n+1}(b^{\mathbf{a},o}, \mathbf{a}') = \frac{(q^{*,\mathbf{a}'}_{n+1})^T H^{\mathbf{a},o} b}{\mathbf{1}^T H^{\mathbf{a},o} b} \quad . \tag{6.8}$$

Returning to (6.3), and reorganizing and simplifying terms:

$$\begin{aligned} V^*_n(b) &= \max_{\mathbf{a} \in \mathcal{A}} \left\{ \left(r^{\mathbf{a}} \cdot b + \gamma \sum_{o \in \mathcal{O}} \mathbf{1}^T H^{\mathbf{a},o} b \max_{\mathbf{a}' \in \mathcal{C}(\mathbf{a},o)} \frac{(q^{*,\mathbf{a}'}_{n+1})^T H^{\mathbf{a},o} b}{\mathbf{1}^T H^{\mathbf{a},o} b} \right) \right\} \\ &= \max_{\mathbf{a} \in \mathcal{A}} \left\{ \left(r^{\mathbf{a}} \cdot b + \gamma \sum_{o \in \mathcal{O}} \max_{\mathbf{a}' \in \mathcal{C}(\mathbf{a},o)} (q^{*,\mathbf{a}'}_{n+1})^T H^{\mathbf{a},o} b \right) \right\} \end{aligned} \tag{6.9}$$

Therefore, $V^*_n(b) = \max_{\upsilon_n \in \Upsilon_n} \upsilon_n \cdot b$, with

$$\upsilon_n = r^{\mathbf{a}} + \left(\gamma \sum_{o \in \mathcal{O}} \max_{\mathbf{a}' \in \mathcal{C}(\mathbf{a},o)} (q^{*,\mathbf{a}'}_{n+1})^T H^{\mathbf{a},o} \right)^T \quad . \tag{6.10}$$

It should be noted that each vector υ is associated with a particular action $\mathbf{a}$. This concludes the proof that Event-Driven MPOMDPs have PWLC optimal value functions. □

Next, we will show how this result can be used by most current POMDP planning algorithms, with minor modifications, for Event-Driven MPOMDPs.

6.3.2 A Randomized Point-Based Algorithm

We now turn our attention to the problem of calculating approximately optimal policies for Event-Driven MPOMDPs. We here focus explicitly on *approximately* optimal policies for computational reasons. However, that does not preclude the possibility of adapting optimal algorithms as well.

As we have mentioned in Section 2.3.1.1, a particularly efficient family of approximate POMDP solvers is that of point-based algorithms. We propose an adaptation of the PERSEUS randomized point-based algorithm that can handle Event-Driven MPOMDPs. The basic premise of any point-based algorithm is that, given a belief state $b \in \Pi(\mathcal{S})$, and a value function (or set of $Q-$value functions) at stage $n+1$, it is possible to obtain the stage-n maximizing vector at that point at a relatively low computational cost.

We are therefore interested in obtaining:

$$q_n^{\mathbf{a},b} = \arg\max_{q_n^{\mathbf{a}} \in \mathcal{K}_n^{\mathbf{a}}} q_n^{\mathbf{a}} \cdot b \tag{6.11}$$

From (6.10), we have that:

$$q_n^{\mathbf{a}} = r^{\mathbf{a}} + \left(\gamma \sum_{o \in \mathcal{O}} \max_{\mathbf{a}' \in \mathcal{C}(\mathbf{a},o)} (q_{n+1}^{*,\mathbf{a}'})^T H^{\mathbf{a},o}\right)^T \quad . \tag{6.12}$$

Note that this already implicitly defines an optimal $q_{n+1}^{*,\mathbf{a}'}$ at a given point b for a particular $\langle \mathbf{a}, o \rangle$ pair, see Eq. (6.7). If, instead of taking the maximum, we evaluate the expected future reward for each of the vectors $q_{n+1}^{k,\mathbf{a}'} \in \mathcal{K}_{n+1}^{\mathbf{a}'}$, and for a given $\langle \mathbf{a}, o \rangle$:

$$q_n^{k,\mathbf{a}',\mathbf{a},o} = \left((q_{n+1}^{k,\mathbf{a}'})^T H^{\mathbf{a},o}\right)^T \quad . \tag{6.13}$$

Taking the action constraints into consideration, we can select from these vectors the best at b:

$$q_n^{\mathbf{a},o,b} = \arg\max_{q_n^{k,\mathbf{a}',\mathbf{a},o} \mid \mathbf{a}' \in \mathcal{C}(\mathbf{a},o)} q_n^{k,\mathbf{a}',\mathbf{a},o} \cdot b \quad . \tag{6.14}$$

And finally, summing over observations and adding the immediate reward:

$$q_n^{\mathbf{a},b} = r^{\mathbf{a}} + \gamma \sum_{o \in \mathcal{O}} q_n^{\mathbf{a},o,b} \quad . \tag{6.15}$$

Equipped with this result, we can formulate our variant of the PERSEUS algorithm, which we refer to as *Constraint-Compliant* PERSEUS (CC-PERSEUS). This algorithm works by randomly selecting a point b from an appropriately selected set of belief points, $\mathcal{B}$ (these can be obtained, for example, by sampling a number of trajectories in the

system) , at step n. The belief-backup (6.15) is applied to a subset of belief points in a sampled set $\mathcal{B}$ (Spaan and Vlassis, 2005), for all possible actions. Any point $\mathcal{B}$ whose value is improved by this backup is removed from the set, and another point is randomly sampled, until $\mathcal{B}$ is empty. The resulting set of vectors for each action is taken as an approximation of $Q_{n+1}^{*,\mathbf{a}}$, and their union as V_{n+1}^{*}. Our explicit use of the $Q-$value functions stems from the fact that the sets $\mathcal{K}_n^{\mathbf{a}}$ (c.f. (6.6)) must never be empty. Otherwise, if an action had no previous-stage vectors associated to it, we would not have an estimate of its expected value when "forced" by $\mathcal{C}$. The pseudo-code for CC-Perseus is detailed in Algorithm 2.

A note on complexity: since this algorithm is keeping track of all $Q-$value functions, in the worst case, it has to perform $|\mathcal{A}|$ times as many evaluations over the set $\mathcal{B}$ as the standard version of Perseus. It is expected, then, that it should underperform Perseus when running over the same model. However, recall that the main advantage of our formulation is that it allows considerably smaller representations (particularly in $|\mathcal{O}|$) of the same problem.

Finally, we emphasize that the considerations made in this section could be applied to virtually any POMDP solution algorithm. The only requirements are that: $Q-$value functions must be maintained for all actions (so value functions can only be pruned action-wise); and the constraints generated by $\mathcal{C}$ should be satisfied when performing dynamic programming backups.

6.3.3 Execution-Time Belief Updates

In a standard POMDP, a belief state b can be updated by an agent during plan execution, following the execution of $\mathbf{a}$ (by the team), and observation of o, through Eq. (6.4). In an Event-Driven model, this update step is not always applicable. Planning algorithms, such as CC-Perseus, can explicitly model the occurrence of false negative detections of events as symbolic observations. During execution, however, agents will not have access to any information indicating false negative detections. Therefore, agents must take into account the fact that the system can undergo several unobserved transitions between any two belief update steps.

Algorithm 2 Constraint-Compliant PERSEUS

Require: $V_n \neq \emptyset, \mathcal{B} \neq \emptyset$
Ensure: $V_{n+1}(b) \geq V_n(b)$, $\forall b \in \mathcal{B}$
$\mathcal{B}' \leftarrow \mathcal{B}$;
Split V_n into $Q_n^{\mathbf{a}_1}, \ldots, Q_n^{\mathbf{a}_{|\mathcal{A}|}}$;
$Q_{n+1}^{\mathbf{a}_1}, \ldots, Q_{n+1}^{\mathbf{a}_{|\mathcal{A}|}} \leftarrow \emptyset$;
while $\mathcal{B}' \neq \emptyset$ **do**
  $b \leftarrow$ Randomly sample $\mathcal{B}'$;
  $\mathcal{B}' \leftarrow \mathcal{B}' \backslash b$;
  for all $\mathbf{a} \in \mathcal{A}$ **do**
    $q_n^{\mathbf{a},b} \leftarrow$ Backup $Q_n^{\mathbf{a}}$ at b (Eq. (6.15));
    $Q_{n+1}^{\mathbf{a}} \leftarrow Q_{n+1}^{\mathbf{a}} \cup q_n^{\mathbf{a},b}$;
    for all $b' \in \mathcal{B}'$ **do**
      if $Q_{n+1}^{\mathbf{a}'} \neq \emptyset \wedge Q_{n+1}^{\mathbf{a}'}(b') \geq Q_n^{\mathbf{a}'}(b'), \forall \mathbf{a}' \in \mathcal{A}$ **then**
        $\mathcal{B}' \leftarrow \mathcal{B}' \backslash b'$;
      end if
    end for
  end for
end while
$V_{n+1} \leftarrow \cup_{\mathbf{a} \in \mathcal{A}} Q_{n+1}^{\mathbf{a}}$;
return V_{n+1}

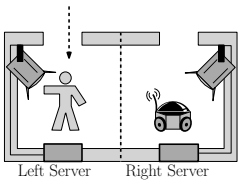
Theorem 3. *Let $o_f \in \mathcal{O}$ represent false negative detections of events. For an infinite-horizon agent in an Event-Driven POMDP, given that the team is executing $\mathbf{a}$ and observing o in belief state $\hat{b}$, the belief update step is:*

$$\hat{b}^{\mathbf{a},o} = \frac{\left(H^{\mathbf{a},o}(I - H^{\mathbf{a},o_f})^{-1}\hat{b}\right)}{\mathbf{1}^T\left(H^{\mathbf{a},o}(I - H^{\mathbf{a},o_f})^{-1}\hat{b}\right)} \quad , \tag{6.16}$$

iff for all eigenvalues λ of $H^{\mathbf{a},o_f}$, $|\lambda_i| < 1$.

Proof. For $o_f \in \mathcal{O}$ indicating false negative observations, $o \in \mathcal{O} \backslash o_f$ and $\mathbf{a} \in \mathcal{A}$, we have that:

$$\hat{b}^{\mathbf{a},o}(s) \propto \sum_{s_1 \in \mathcal{S}} \Pr(o|s_1, \mathbf{a}, s) \Pr(s|s_1, \mathbf{a}) \times \Big(b(s_1) + \sum_{s_2 \in \mathcal{S}} \Pr(o_f|s_2, \mathbf{a}, s_1) \Pr(s_1|s_2, \mathbf{a}) \times \Big(b(s_2) + \sum_{s_3 \in \mathcal{S}} \Pr(o_f|s_3, \mathbf{a}, s_2) \Pr(s_2|s_3, \mathbf{a}) \times \Big(b(s_3) + \ldots\Big)\Big)\Big)$$



Problem	Model	$\|\mathcal{S}\|$	$\|\mathcal{A}\|$	$\|\mathcal{O}\|$	d
Access2	E	72	6	9	3
	S	72	18	96	3
Access3	E	216	8	10	4
	S	216	54	256	4

Figure 6.5: Left: A layout of the *Access2* problem; Right: Size of the model components for the tested scenarios, using event-driven (E) and synchronous (S) approaches.

In matrix notation, as before, this is:

$$\hat{b}^{\mathbf{a},o} \propto H^{\mathbf{a},o}\Big(\sum_{k=0}^{\infty}(H^{\mathbf{a},o_f})^k\Big)\hat{b} \tag{6.17}$$

If $|\lambda_i| < 1$ for all eigenvalues $|\lambda|$ of $H^{\mathbf{a},o_f}$:

$$\hat{b}^{\mathbf{a},o} = \frac{1}{\eta}H^{\mathbf{a},o}(I - H^{\mathbf{a},o_f})^{-1}\hat{b} \tag{6.18}$$

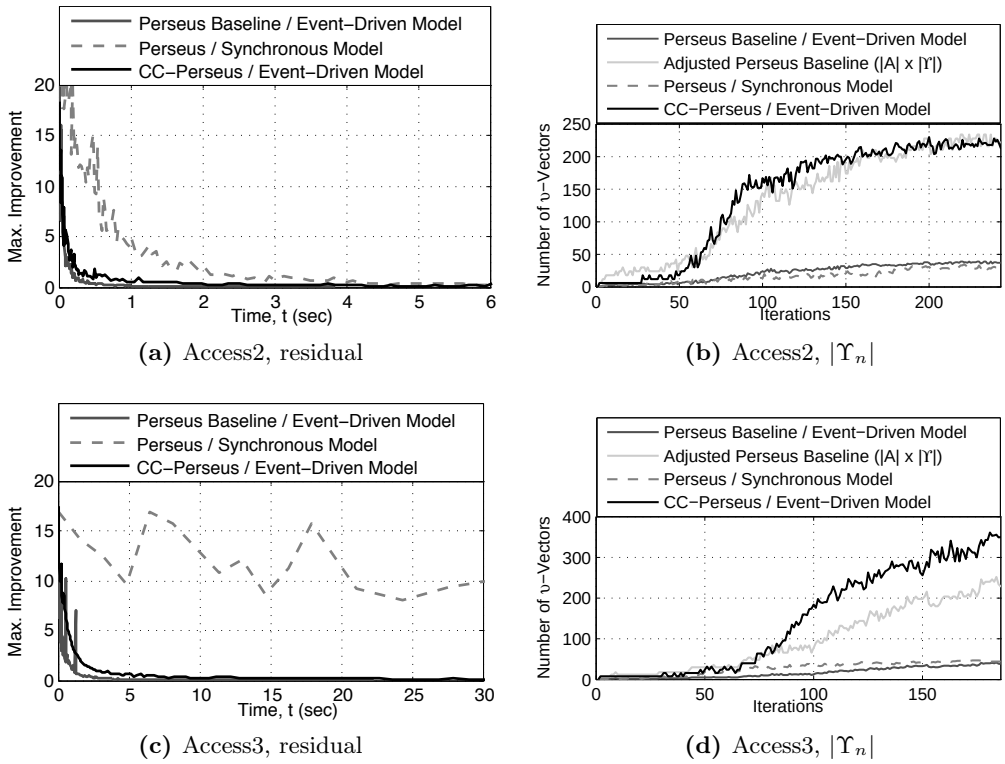
with $\eta = \mathbf{1}^T H^{\mathbf{a},o}(I - H^{\mathbf{a},o_f})^{-1}\hat{b}$, since $\mathbf{1}^T\hat{b}^{\mathbf{a},o} = 1$. We note that if any $|\lambda_i| = 1$, this implies that the system has completely unobservable ($\Pr(o_f|\cdot) = 1$) loops. In those conditions, the belief state cannot be tracked. □

The notation $\hat{b}$ here indicates run-time belief states, so as to clarify that Eqs. (6.4) and (6.16) will produce different outputs. In large systems, if the computational complexity of obtaining $(I - H^{\mathbf{a},o_f})^{-1}$ is prohibitive, it can be approximated through a finite number of sums (see proof). The requirement that $H^{\mathbf{a},o_f}$ must have eigenvalues less than 1 can be seen as a generalization of a recent result of Cabasino et al. (2013) in DES theory, in which it was shown that tracking the possible states of a Petri Net with unobservable or perceptually aliased transitions is only possible if those transitions are acyclic.

6.4 Experiments

In this section we evaluate our proposed methodology, in simulated environments, by comparing its performance to that of synchronous models of the same decision-making problems.

We consider an autonomous multiagent surveillance system, where static agents (e.g. sensors) and mobile agents (e.g. robots) must cooperate in order to control the



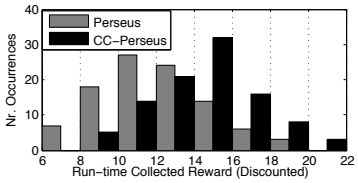
(a) Access2, residual **(b)** Access2, $|\Upsilon_n|$

(c) Access3, residual **(d)** Access3, $|\Upsilon_n|$

Figure 6.6: (a), (c) Residual difference between successive value function approximations, $\max_{\mathcal{B}}\{V_n(b) - V_{n+1}(b)\}$ (b), (d) Size of the value function, $|\Upsilon_n|$, as a function of n.

access of human operators to sensitive equipment. In the *Access*2 problem, two sensors which provide biometric information (e.g., cameras) are connected to two restricted-access servers located in adjacent rooms (see Figure 6.5). Either sensor can mistake the validity of a user's credentials, or fail to detect that a user is there at all (0.2 probability each). A robot aids these sensors by performing additional measurements, reducing the possibility of false positives and negatives, and also acts as a failsafe in case one of the sensors malfunctions. The robot can move between the two rooms (0.8 probability of successfully switching rooms), and knows its position with certainty. The goal of the problem is to authorize as many valid users as possible (a reward of 10 for each), and to explicitly reject all others, or at least idly refrain from authorizing them (10 and 0), respectively. There is a -10 penalty for giving access to non-authorized user.

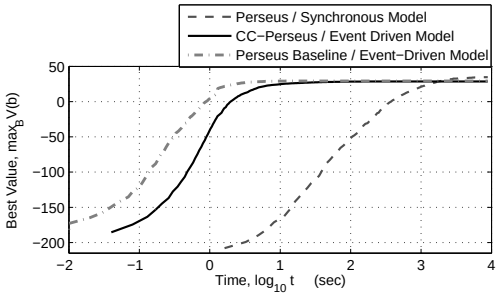
This task was represented using both an event-driven and a standard synchronous approach, using factored models. The sizes of the respective model components can be seen in Figure 6.5. To see why there is such a pronounced difference in $|\mathcal{O}|$, consider



(a) Reward (histogram)

Algorithm	Run-Time Reward	$\bar{\delta}_V$
PERSEUS	12.94 ± 2.82	4.73
CC-PERSEUS	15.45 ± 2.75	0.60

(b) Reward (relevant statistics)



(c) Planner convergence vs. time

Figure 6.7: (a) Reward accumulated at run-time for *Access2*, as a comparative histogram for 100 runs of 25 steps. (b) Respective mean/deviation, and mean error between collected reward and expected value ($\bar{\delta}_V$). (c) Evolution of $\max_{\mathcal{B}} V_n(b)$ in real time for the various models/solvers in *Access3*, showing similar final results, but faster convergence in the event-driven case.

that each sensor can observe authorized and non-authorized users, hardware failures, or nothing at all. The robot can also observe users (or nothing), along with its own position. In the synchronous approach, we must take the product of all of these possibilities. In the event-driven approach, the only detections that we need to model are the arrival/exit of a user, the failure of either sensor, and a change of state by the robot, plus a "false negative" observation, as per Definition 6.2.2. Additionally, we can reduce the number of joint actions by ruling out inconsistent decisions (for example, there is never the need to authorize two users simultaneously).

Both the normal PERSEUS algorithm (Spaan and Vlassis, 2005) and CC-PERSEUS were run on this problem, over a set of 50 sampled belief points with $\gamma = 0.95$ and $h = \infty$. All experiments were run on an Intel Core i5 2.67Ghz computer with 4GB RAM. As a baseline, the PERSEUS algorithm was run on our event-driven model, disregarding the effect of unobservable events. Figure 6.6a shows the maximum improvement (or residual) to the value function between steps using these algorithms, which is indicative of their real-time convergence. From here we can see that CC-PERSEUS on the event-driven model outperforms its standard version in the synchronous setting (total run-time was $22.78\,s$ vs. $29.03\,s$, respectively, until a residual of 10^{-4}). Figure 6.6b also shows how the total number of vectors of CC-PERSEUS follows $|\mathcal{A}|$ times that of the baseline, since $Q-$functions are explicitly maintained for each action. In Figure 6.7 we show how the baseline policy, even though faster to compute, is outperformed by CC-PERSEUS,

since action constraints are not considered in the former case. Due to this fact, the expected value calculated by PERSEUS does not correspond to the reward accrued at run-time (an error of 27%, vs. 3% with CC-PERSEUS).

In order to showcase the scalability of these methods, a larger version of the above problem was implemented. *Access3* has 3 rooms/sensors/servers along a corridor, but the sensor at the center can only detect authorized personnel (or nothing). Therefore, there is only one more event with respect to the previous problem, but the presence of another agent causes an exponential increase in the number of observations of the synchronous model. Figure 6.5 shows the sizes of the models for this problem, and Figures 6.6c and 6.6d show performance results. Although synchronous and event-driven models are not strictly comparable with regard to expected reward, since their dynamics are inherently different, we show in Figure 6.7c an overlay of the best expected value (in the sample set) for *Access*3 using either model, to establish that they in fact converged to similar near-optimal policies. Running times to a residual of 10^{-4} were $6\,m\,52.39\,s$ for event-driven CC-PERSEUS and $2h\,27\,m\,24.27\,s$ for synchronous PERSEUS. This shows that the simple addition of an agent increased the computational advantage of the event-driven model over its alternative by more than an order of magnitude (also clearly visible in Figure 6.7c). The problem files for these experiments can be accessed at `http://users.isr.ist.utl.pt/~jmessias/PhDthesis`[1].

6.5 Extension to Generalized Semi-Markovian Domains

We will now describe how our Event-Driven MPOMDP framework can be extended beyond fully-Markovian domains, using the methodologies of Chapter 5.

As we have previously seen for fully observable cases, if event firing times are governed by non-exponential distributions $f_e^{\mathbf{a}}$, then a factored asynchronous system, in which more than one event can be enabled in parallel, can only be fully modeled by a GSMDP. The resulting model can, however, be approximated as an SMDP, by replacing those non-exponential distributions with Phase-Type distributions. Therefore, in order to extend Event-Driven MPOMDPs to the domain of continuous-time, we begin by incorporating Semi-Markovian dynamics into our framework.

[1] Our problem files are written in the ProbModelXML format (Arias et al., 2012), and are solvable with the MultiAgent Decision Process (MADP) toolbox (Spaan and Oliehoek, 2008)

Definition 6.5.1. *An Event-Driven Multiagent Partially Observable Semi-Markov Decision Process (Event-Driven MPOSMDP) is a tuple* $\langle d, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{E}, T, O, \mathcal{F}, R, C \rangle$ *where:*

$\langle d, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{E}, T, O, R \rangle$ *are defined as in a discrete-time Event-Driven MPOMDP (Definition 6.2.2);*

$\langle \mathcal{F}, C \rangle$ *are defined as in an SMDP (Definition 2.2.1), and describe, respectively, probability distributions over the firing time of each event, and cumulative reward rates.*

From Definition 6.5.1, we see that the extension of our framework to Semi-Markovian domains is conceptually very simple, since we have simply introduced the components of SMDPs that weren't already present in Definition 6.2.2. The resulting extended framework is still analytically tractable, but only because we have opted to explicitly model false negative detections as constraints on the decision-making process (see Section 6.2.3).

In this case, as a stochastic process, our multiagent decision-making process evolves as follows:

1. An action $\mathbf{a} \in \mathcal{A}$ is selected by the team at belief state $b \in \mathcal{B}$;
2. An event $e \in \mathcal{E}$ is sampled from $T(s, \mathbf{a}, e)$;
3. A time t is sampled from $f_e^{\mathbf{a}}(t)$. This is the time at which e will trigger;
4. As e triggers, an observation of that event is sampled from $O(\mathbf{a}, e, o)$, and a new step of the process begins;
5. **If and only if** o is not a false negative observation ($o \neq o_f$), the team is allowed to select a new action $\mathbf{a}' \in \mathcal{A}$. Otherwise, $a' = a$.

Note that the planning horizon, for such a system, can be defined over the number of decision steps *or* in the total amount of continuous time that the system experiences. We are typically interested in agents that are able to operate indefinitely, particularly in Robotics applications. However, as in the case of discrete POMDPs, for the solution to these models to be practical, we need to approximate the optimal stationary solution, by calculating finite-step value functions with sufficiently large n.

With this, we will now show that value functions for Event-Driven MPOSMDPs are also PWLC, for a finite number of steps and infinite time.

Theorem 4. *The optimal value function V_n^* for an Event-Driven MPOSMDP is PWLC for finite n and infinite t. That is, it can be written as:*

$$V_n^*(b) = \max_{\upsilon_n \in \Upsilon_n} \upsilon_n \cdot b \quad ,$$

where Υ_n is a set of $|S|$-dimensional vectors.

Proof. Adapting Eqs. (2.15) and (6.2), the optimal value at $b \in \mathcal{B}$, and at step n, is:

$$V_n^*(b) = \max_{\mathbf{a}\in\mathcal{A}} \Big\{ \sum_{s\in S} b(s)U(s,\mathbf{a}) \quad + \\ \sum_{\substack{s\in\mathcal{S},o\in\mathcal{O}\\ \varepsilon\in E(s,\mathbf{a}),}} \int_0^\infty b(s)p(o,\varepsilon,t\,|\,s,\mathbf{a}) \left(\max_{\mathbf{a}'\in\mathcal{C}(\mathbf{a},o)} Q_{n+1}^*(b^{\mathbf{a},o,t},\mathbf{a}') \right) e^{-\lambda t} dt \Big\} \quad . \tag{6.19}$$

Here, $\int_0^\tau p(o,\varepsilon,t\,|\,s,\mathbf{a})dt$ represents the probability that the next transition is due to event e, at or before time τ, and the corresponding detection is o. $\mathcal{C}(\mathbf{a},o)$ is defined as in Eq. (6.1), and $U(s,\mathbf{a})$ is obtained through Eq. (2.14). From Bayes' rule we have:

$$\begin{aligned} p(o,e,t\,|\,s,\mathbf{a}) &= p(o,t\,|\,\mathbf{a},e)\Pr(e\,|\,s,\mathbf{a}) \\ &= p(t\,|\,\mathbf{a},e)\Pr(o\,|\,\mathbf{a},e)\Pr(e\,|\,s,\mathbf{a}) \\ &= f_e^{\mathbf{a}}(t)O(\mathbf{a},e,o)T(s,\mathbf{a},e) \quad , \end{aligned}$$

where we used the properties that $p(o,t\,|\,e,s,\mathbf{a}) = p(o,t\,|\,e,\mathbf{a})$ (from Definition 2.2.2) and $p(t\,|\,o,e,\mathbf{a}) = p(t\,|\,e,\mathbf{a})$.

Also, note that the belief update now also depends on the elapsed time. From (Mahadevan, 1998):

$$b^{\mathbf{a},o,\tau}(s') = \frac{\sum_{s\in\mathcal{S}} f^{\mathbf{a}}_{\Phi(s,s')}(\tau)T(s,\mathbf{a},\Phi(s,s'))O(\mathbf{a},\Phi(s,s'),o)b(s)}{\sum_{u,u'\in\mathcal{S}} f^{\mathbf{a}}_{\Phi(u,u')}(\tau)T(u,\mathbf{a},\Phi(u,u'))O(\mathbf{a},\Phi(u,u'),o)b(u)} \tag{6.20}$$

We will use the same induction base as in the proof of Theorem 2. That is, at step $n = 0$, in vectorial form, we have:

$$V_0^*(b) = \max_{\mathbf{a}\in\mathcal{A}} u^{\mathbf{a}} \cdot b \quad , \tag{6.21}$$

where $u^{\mathbf{a}}$ is the a-th column of $U(s,\mathbf{a})$.

We will also define a $|\mathcal{S}| \times |\mathcal{S}|$ matrix, $G^{\mathbf{a},o,\tau}$, such that, for $\mathbf{a} \in \mathcal{A}$, $o \in \mathcal{O}$ and $\tau \in \mathbb{R}_0^+$:

$$[G^{\mathbf{a},o,\tau}]_{i,j} = f^{\mathbf{a}}_{\Phi(s_i,s_j)}(\tau)T(s_j,\mathbf{a},\Phi(s_j,s_i))O(\mathbf{a},\Phi(s_j,s_i),o) \quad .$$

Then, adapting Eq (6.8), we can rewrite the backup at step n as:

$$V_n^*(b) = \max_{\mathbf{a}\in\mathcal{A}} \Big\{ u^{\mathbf{a}} \cdot b + \sum_{o\in\mathcal{O}} \int_0^\infty \mathbf{1}^T G^{\mathbf{a},o,t} b \max_{\mathbf{a}'\in\mathcal{C}(\mathbf{a},o)} \frac{(q_{n+1}^{*,\mathbf{a}'})^T G^{\mathbf{a},o,t} b}{\mathbf{1}^T G^{\mathbf{a},o,t} b} e^{-\lambda t} dt \Big\} \tag{6.22}$$

$$= \max_{\mathbf{a}\in\mathcal{A}} \Big\{ u^{\mathbf{a}} \cdot b + \sum_{o\in\mathcal{O}} \int_0^\infty \max_{\mathbf{a}'\in\mathcal{C}(\mathbf{a},o)} (q_{n+1}^{*,\mathbf{a}'})^T G^{\mathbf{a},o,t} b \; e^{-\lambda t} dt \Big\} \quad . \tag{6.23}$$

Returning, momentarily, to our "flat" notation, and defining $s'_{e,s}$ to be the resulting state after event e in state s (that is, $\Phi(s, s'_{e,s}) = e$), we have that:

$$V_n^*(b) = \max_{\mathbf{a}\in\mathcal{A}} \Big\{ \sum_{s\in S} b(s)U(s,\mathbf{a}) \quad + \sum_{\substack{s\in\mathcal{S},o\in\mathcal{O}\\ \varepsilon\in E(s,\mathbf{a}),}} \int_0^\infty \max_{\mathbf{a}'\in\mathcal{C}(\mathbf{a},o)} q_{n+1}^{*,\mathbf{a}'}(s'_{\varepsilon,s}) f_e^{\mathbf{a}}(t) T(s,\mathbf{a},\varepsilon) O(\mathbf{a},\varepsilon,o) b(s) e^{-\lambda t} dt \Big\} \tag{6.24}$$

$$= \max_{\mathbf{a}\in\mathcal{A}} \Big\{ \sum_{s\in S} b(s)U(s,\mathbf{a}) \quad + \sum_{\substack{s\in\mathcal{S},o\in\mathcal{O}\\ \varepsilon\in E(s,\mathbf{a}),}} \max_{\mathbf{a}'\in\mathcal{C}(\mathbf{a},o)} q_{n+1}^{*,\mathbf{a}'}(s'_{\varepsilon,s}) T(s,\mathbf{a},\varepsilon) O(\mathbf{a},\varepsilon,o) b(s) \int_0^\infty f_e^{\mathbf{a}}(t) e^{-\lambda t} dt \Big\} \tag{6.25}$$

Let $D^{\mathbf{a}}$ be an $|\mathcal{S}| \times |\mathcal{S}|$ matrix such that:

$$[D^{\mathbf{a}}]_{s,s'} = \begin{cases} \mathcal{L}\{f_e^{\mathbf{a}}(t)\} & \text{if } \Phi(s,s') \in E(s,\mathbf{a}), \\ 0 & \text{otherwise.} \end{cases}$$

Then,

$$V_n^*(b) = \max_{\mathbf{a}\in\mathcal{A}} \Big\{ u^{\mathbf{a}} \cdot b + \sum_{o\in\mathcal{O}} \max_{\mathbf{a}'\in\mathcal{C}(\mathbf{a},o)} (q_{n+1}^{*,\mathbf{a}'})^T \left(H^{\mathbf{a},o} \circ D^{\mathbf{a}}\right) b \Big\} \quad , \tag{6.26}$$

where $A \circ B$ is the Hadamard (entry-wise) product of A an B, and, $H^{\mathbf{a},o}$ is defined as in the proof of Theorem 2.

Finally, we have that $V_n^*(b) = \max_{\upsilon_n \in \Upsilon_n} \upsilon_n \cdot b$, with

$$\upsilon_n = u^{\mathbf{a}} + \Big(\sum_{o \in \mathcal{O}} \max_{\mathbf{a}' \in \mathcal{C}(\mathbf{a},o)} (q_{n+1}^{*,\mathbf{a}'})^T \left(H^{\mathbf{a},o} \circ D^{\mathbf{a}} \right) \Big)^T \quad . \tag{6.27}$$

□

This theoretical result enables us to apply algorithms for Event-Driven MPOMDPs, such as CC-Perseus, to this Semi-Markovian framework. Indeed, we note that the only necessary modification to the algorithm presented in Section 6.3.2 would be the substitution of Eq. (6.13) with

$$q_n^{k,\mathbf{a}',\mathbf{a},o} = \Big((q_{n+1}^{k,\mathbf{a}'})^T \left(H^{\mathbf{a},o} \circ D^{\mathbf{a}} \right) \Big)^T \quad . \tag{6.28}$$

We emphasize, however, that this was only possible since we have considered decision steps to occur at every event, including those that are *not* detected; if that wasn't the case, and we wouldn't be able to describe the behavior of the system between decision steps due to the effect of false negative observations and intermediate transitions, analogously to what we have proposed in Section 6.2.3. This is due to the fact, in the limit, an unobserved chain of infinite transitions would involve infinite nested integrals of $f_e^{\mathbf{a}}(t)$ for the respective intermediate events and actions. This also prevents us from extending the run-time belief update proposed in Theorem 3 to these continuous-time models. For this reason, we cannot (yet) execute Event-Driven MPOSMDP plans, and we have not collected empirical results using this framework.

Conceptually, in an Event-Driven MPOSMDP equipped with time, transition, and observation models described directly over *event detections*, and not over *events*, the update (6.20) could be used. In particular, this would require time models of the form $f_{s,s'}^{\mathbf{a},o}$, *i.e.* depending explicitly on the initial state s and the state s' at the time of the next detection, but not on any intermediate undetected events. The problem with such models is that they do not isolate the effect of false negative detections: they depend implicitly on the intermediate states between s and s', which may affect the probability of emitting a given observation. Therefore, these distributions can easily become impossible (or at least highly impractical) to parametrize. We note, however, that for the belief update (6.20), these time models do not need to be integrated, but simply evaluated at τ, so we conjecture that learning approximations to these functions could

suffice, in practice. Obtaining an appropriate execution-time belief update mechanism for Event-Driven MPOSMDPs constitutes one of our most immediate topics for future work.

Finally, we note that the only difference between the proposed Event-Driven MPOSMDP framework, as per Definition 6.5.1 and a conceptual, even more general Multiagent Partially Observable GSMDP is that, in the latter, event times would be allowed to depend on the execution history of the system. Therefore, the application of the Event-Driven MPOSMDP framework to Generalized Semi-Markov domains follows trivially from the considerations of Chapter 5 — that is, it can be used to approximate Generalized Semi-Markov systems, if all of the persistently enabled events in those systems are replaced by unobservable Semi-Markov chains.

6.6 Summary

In this chapter, we have proposed a novel modeling approach for multiagent decision-making under partial observability, based on the MPOMDP framework, which draws from the concepts of asynchrony in Discrete-Event systems to allow a more compact representation of such scenarios than what is typically possible through decision-theoretic frameworks.

We have described how such a model could be formalized and shown that it still retains the essential properties that allow it to be solved through dynamic programming. We have also shown how a common POMDP-solving algorithm could be adapted to function in an event-driven paradigm, and how agents can track belief states at runtime in the presence of false negative observations.

We have supported the efficiency of our framework, by showing that, for simulated environments, our asynchronous models could be solved faster than synchronous alternatives, and the resulting event-driven policy performed better during execution.

Finally, we have also shown how these concepts could be tied to the work presented in Chapter 5 for Generalized Semi-Markovian systems. The proposed Event-Driven MPOSMDP framework constitutes a direct result of our effort of integrating Discrete-Event Systems and Decision Theory, and allows the modeling of fully communicative multi-robot systems, subject to the influence of continuous time, and partial observability.

Chapter 7

A Case Study in Multiagent Surveillance

7.1 Introduction

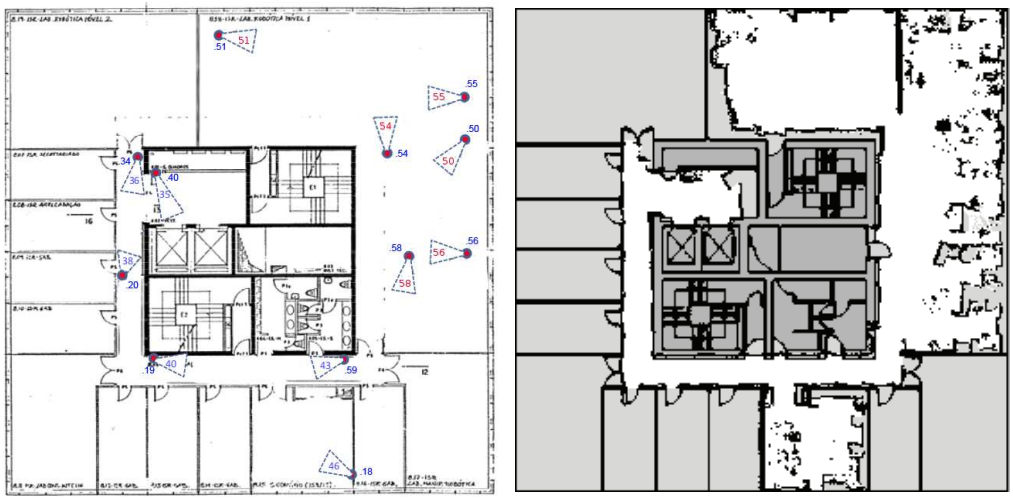
In this chapter, we describe the application of the Event-Driven MPOMDP framework, proposed in Chapter 6, to a decision-making problem involving a real networked robot system of significant dimension, operating in the context of autonomous surveillance. In doing so we will discuss, from a pragmatical standpoint, the decision-theoretic modeling process for this system, and also the organization and functionality of the various software components that are necessary for the deployment of these methods on a real team of robots. Among these software components, we will note those that are direct and novel contributions of this work, and that were developed with the purpose of easing the implementation of generic decision-theoretic controllers on multi-robot systems.

Finally, we will evaluate the performance of our multi-robot system.

7.2 The MAIS+S Testbed

Our case study takes place as part of the *MultiAgent Intelligent Surveillance System* (MAIS+S) project[1]. The main testbed for MAIS+S is a networked surveillance system that combines common stationary sensors (cameras) with mobile autonomous agents (robots). The purpose of this framework is to complement a typical human surveillance

[1]Project reference CMU-PT/SIA/0023/2009. Web: http://gaips.inesc-id.pt/mais-s/project.html



(a) Camera positions / FOVs

(b) Robot operation area

Figure 7.1: The environment of our surveillance framework. (a): The positions and fields-of-view (FOVs) of the stationary cameras in our surveillance system, overlaid on the respective floor plan at ISR-Lisbon; (b): The area that was designated for the operation of our robots, shown in white. The gray shaded areas are off-limits to the robots, and also outside of the coverage of the camera network. The boundaries of this map were obtained experimentally using the onboard laser rangefinders of the robots.

team by providing situational awareness based on video analytics and fused sensor data, while leveraging decision-theoretic solutions, such as those presented in this thesis, to allow autonomous response of the robot team to relevant events, in the absence of human input. This testbed is an extension of the earlier ISRobotNet surveillance framework (Barbosa et al., 2009).

In the following sections, we describe the fundamental aspects of our surveillance framework, and of its event-driven control architecture.

7.2.1 Hardware

The physical testbed for this project, which is installed at the Institute for Systems and Robotics (ISR-Lisbon), is comprised of a network of stationary cameras and a pair of mobile robots, all of which are able to share data in real-time.

A floor plan of the environment for this system is represented in Figure 7.1a, which also shows the camera positions and respective fields-of-view. The system contains twelve AXIS 211/P1344 network cameras (Figure 7.2b), connected to three servers (HP DL120 G6), which are able to process the live feed from the cameras at a maximum of

(a) The robot team

(b) Surveillance cameras

(c) Camera views

Figure 7.2: (a): Our robot team, *Duke* (left) and *Orwell* (right); (b): Examples of the network cameras used in our surveillance system; (c): Typical view from (some of) our surveillance cameras. The bounding boxes shown around people in these images are the output of a set of video processing algorithms, running in real time in our camera servers.

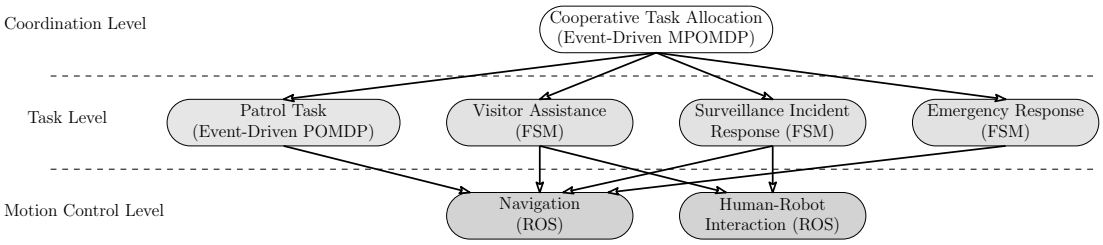


Figure 7.3: The various levels of decision-making involved in our two-robot autonomous surveillance scenario. This hierarchical organization integrates different decision-making formalisms – the cooperative between agents is modeled as an Event-Driven MPOMDP; the tasks that it abstracts as actions were described either as manually designed Finite State Machines (FSMs), or, in the case of the 'Patrol' task, as a single-agent Event-Driven POMDP.

25 frames per second. A set of video processing algorithms running on these servers, which is outside of the scope of this work, is able to detect the position of people in the areas covered by each camera, and also if people are waving directly at any of the cameras (Moreno et al., 2009). Figure 7.2c shows snapshots from typical camera feeds.

The robot team in this framework consists of two Pioneer 3-AT robots, equipped with laser scanners and webcams (Figure 7.2a). Each robot carries an onboard laptop (Sony VAIO VPCS135FA/B), which not only runs its navigation and decision-making algorithms, but also acts as an interface for Human-robot interaction. The robots are connected to the network via their wireless adapters, and are able to roam seamlessly, and with negligible packet loss, across different wireless access points over the area designated for their operation (shown in Figure 7.1b). When operating continuously, each robot has approximately 30 minutes of battery life.

7.2.2 Decision-Making

The operation of our robot team as a part of the surveillance framework requires decision-making at different levels of abstraction. These are graphically represented in Figure 7.3. The cooperative decision-making problem in this scenario lies at the top of this hierarchical organization, and concerns the allocation of *tasks* between the robots, as a response to the discrete detections of the sensor network.

The event-driven methodologies that were studied in Chapter 6 are particularly appropriate to this type of problem. Since agents (robots) and static sensors (cameras) are heterogeneous, and communicate over different media, subject to different delays

and restrictions (wired / wireless network), ensuring their synchronization is not only difficult, but undesirable. Furthermore, for a large network that is supposed to operate over extended periods of time, it is more efficient, in terms of communication usage, to exclusively communicate relevant changes in the system, than it is to perform synchronous communication with mostly constant information, at a sufficiently high rate so as to preserve reactivity to detections.

We cast the problem of multi-robot coordination in our surveillance framework as an Event-Driven MPOMDP[1]. The actions in this model correspond to the abstract tasks that each robot must be capable of performing individually. We will now provide a short description of the requirements and of the design of each of these tasks:

1. **Patrol**: This is the default task for both robots, that should be performed in the absence of events in the surveillance framework. When patrolling, each agent should actively search for ongoing occurrences.

   Since this is naturally a partially observable problem, it was modeled as a single-agent Event-Driven POMDP. The state space for this task contains the position of the robot in a topological graph describing its area of operation (Figure 7.15b), and also the position of a fictitious "target", which is drawn with uniform probability over the topological nodes. The robot is rewarded for observing and "capturing" the target, which is only possible if both the robot and the target are in the same topological node. This induces the behavior of searching every node for the presence of the target, until the robot is sufficiently certain that no target exists. In that case, the robot terminates its patrol round, and a new fictitious target is spawned.

   The actions in this model correspond to movement to an adjacent topological node, plus "capture" and "terminate round" actions. Observations, in turn, correspond to the presence or absence of the occurrence in each node. An important feature of this model is that observations may be emitted by external sources of information – the robot may receive observations signaling that a room is clear from its partner, or from a human security agent, for example. This will naturally cause the robot

[1]Evidently, an Event-Driven MPOSMDP (Section 6.5) would be a more exact representation, since events may have different rates of occurrence over time. However, as we cannot yet execute Event-Driven MPOSMDP policies, we here make the assumption that all events occur at the same rate, which, as it will be shown, does not compromise the operation of the system.

to skip those rooms until its present patrol round is terminated. When both robots execute this task simultaneously, this results in an implicitly cooperative behavior in which each robot patrols only part of the environment (Figure 7.15a). The full state, action, and observation space description of this task can be found in Appendix A.

2. **Assistance Response**: In this task, the robot should assist visitors to ISR, in guiding them to their desired destinations. Visitors may request assistance by waving directly to one of the cameras. When the cameras communicate the position of the waving person to the robot team, a robot should then move there and interact with that person. If both robots simultaneously decide to assist a visitor, the closest robot has priority, and should proceed with the task. The remaining robot should cancel its execution and select a different task. A robot can also decide to assist visitors *without* having received a waving detection from one of the cameras. In this case, the robot should audibly offer to guide any waiting visitors, while scanning its operational area;

3. **Surveillance Incident Response**: This task encapsulates the response to generic *surveillance incidents* – in particular, these can be events in which unauthorized people trespass into restricted areas, in which case a robot should warn the trespasser; or camera failures, which should prompt one of the robots to minimize the uncertainty over the expected field-of-view of the disabled camera, in order to maintain optimal sensor coverage of the environment;

4. **Emergency Response**: A symbolic task representing the response of the robots to emergencies (*e.g.* fires). This is the highest priority task, and should prompt both robots to move to the position of the detected emergency. The task is completed immediately if both robots reach the emergency position, or if one robot reaches and holds that position for 120 seconds. Although emergency detections must be simulated, this task serves the purpose of being an easily recognizable *joint* action.

The "Assistance Response", "Surveillance Incident Response" and "Emergency Response" tasks were implemented as manually specified Finite State Machines (FSMs). The layout of these FSMs can be found in Appendix A.

Tasks may have a higher probability of being successfully completed if they are performed simultaneously by both agents (*e.g.* addressing an emergency situation), while in others it is pointless to engage both robots simultaneously (*e.g.* assisting a visitor).

The state space in our coordinative Event-Driven MPOMDP model is factored into a set of variables which encode the occurrence of each of the events that the robot team is supposed to respond to, according to the aforementioned tasks (visitors requesting assistance, surveillance incidents, and emergencies); and also the condition of each robot, *i.e.* whether it is enabled or disabled, and whether it is busy responding to an event.

Observations, in turn, symbolize the detections of these events, triggered by the surveillance framework as a whole. That is, observations can originate from any of the robots **or** any of the stationary sensors, even though the latter are not formally agents. The robot team selects a joint assignment of tasks (starting a new decision episode) whenever an observation is received. Note that the relationship between events and observations is not a bijection, in the general case – multiple observations can be associated with the same event, in order to model different degrees of certainty with which an event is detected. For example, in our framework, each camera can transmit a detection of a person requesting assistance with either a 'Low' or a 'High' confidence, depending on the outcome of its detection algorithms. These are associated with the same underlying event, but due the fact that each of these detections has different rates of false positives / negatives, the robot team may respond differently to each of them.

The robot team is penalized for each step that an event is awaiting response. Different events carry different penalties according to their priority (emergencies have the highest priority, while assisting visitors has the lowest). This reward strategy implies that, before committing to a low priority event, each robot must consider its partner's availability to respond to high priority situations that may take place in the meantime.

The resulting Event-Driven MPOMDP model has $|\mathcal{S}| = 144$, $|\mathcal{A}| = 16$ and $|\mathcal{O}| = 13$. Due its the size, the specific parameterization of this model (transition, observation probabilities, and rewards) is here omitted, although we encourage the reader to inspect the model definition at: `http://users.isr.ist.utl.pt/~jmessias/PhDthesis` (the patrol task Event-Driven POMDP can also be accessed). The 2-DBN that represents this problem graphically, as well as the full description of its state, action and observation spaces, can be found in Appendix A.

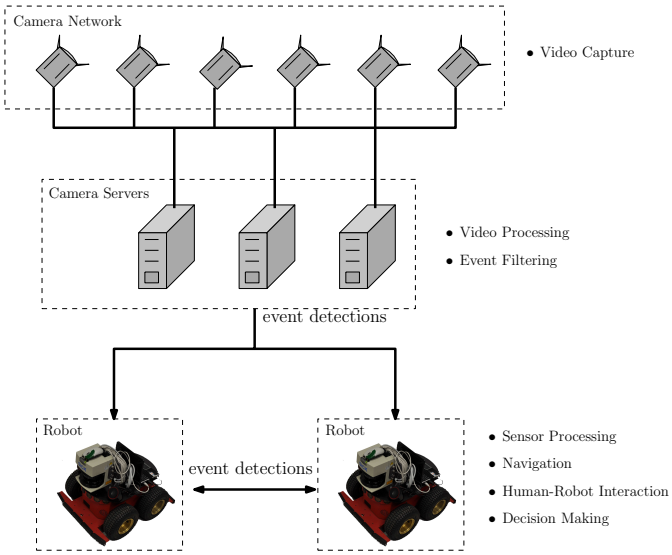


Figure 7.4: An overview of the structure of our surveillance system, and the respective functional distribution.

7.2.3 Software Organization

We used ROS (Quigley et al., 2009) as the middleware for our surveillance system. The inherent asynchronous, distributed execution strategy of ROS is particularly suitable for the deployment of event-driven control strategies, such as the one proposed in this work.

In Figure 7.4, we show an overview of the functional organization of our surveillance framework. We note that all of the decision-making involved in this system runs locally in each robot. Cameras (and their respective servers) act merely as external sources of information, in the form of events. These are then passed to the robots in a format that identifies the type of event, its actual (continuous) position in the environment, and the time associated with the detection. Events are *filtered*, as per the methodology discussed in Section 6.2.4 (Figure 6.3b), which means that detections from different cameras are fused, and subsequent detections from the same camera are tracked, so that the same event is not erroneously transmitted multiple times.

The implementation of the actual decision-making process that is meant to be carried out by the robots, based on these events, is a three-step process (Figure 7.5): first, the Event-Driven MPOMDP model is defined in an appropriate file format. We used OpenMarkov (Arias et al., 2012) to design the model graphically (as per the 2-DBN

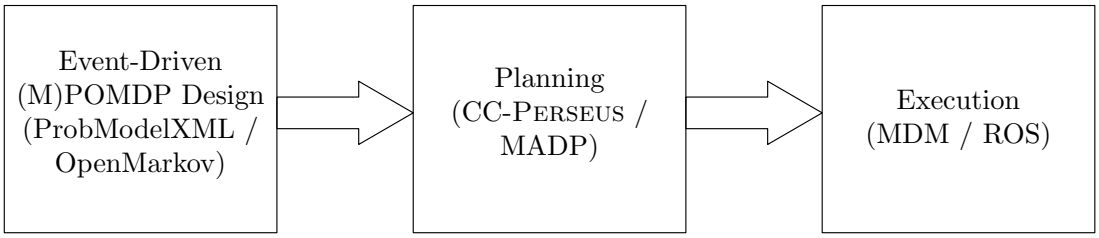


Figure 7.5: The design and implementation of an Event-Driven MPOMDP model.

in Figure A.10); then, we plan over the resulting model using CC-Perseus, through the MADP Toolbox (Spaan and Oliehoek, 2008), to obtain an approximately optimal policy; finally, we execute this policy in each of the mobile robots, by means of the *MDM Library*, a decision-making library that was designed in the context of this work, and which we will describe in Section 7.3. This library handles the semantic grounding of the observations in the model, with respect to the events that are transmitted in the network; and of the actions of the Event-Driven MPOMDP, as the *tasks* that were previously described. We implemented the Finite State Machines describing most actions in the model using the SMACH ROS package[1].

7.3 The *Markov Decision-Making* (MDM) Library

This section describes the "Markov Decision-Making" (MDM) software package and its applications. MDM was developed as part of this thesis work, with the goal of supporting the deployment of DT methodologies to teams of physical autonomous agents.

As we have previously seen throughout this work, modeling a real-world system through an MDP-based framework typically requires the abstraction of relevant characteristics of the environment. Consequently, a plan which results from such a model also provides symbolic, abstract actions, which are not directly applicable to the *control inputs* of a physical autonomous agent (see Section 3.2.3). Therefore, agents must be provided with some means of interpreting the environment through the scope of its associated MDP, and must also be capable of interpreting the actions that the solution to that MDP provides as concrete references to its actuators. This problem is depicted in Figure 7.6.

[1]URL: `http://www.ros.org/wiki/smach`

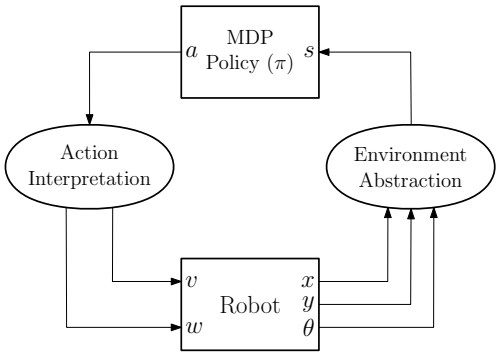


Figure 7.6: An example of the integration of an MDP-based control policy to a robotic agent. The actual control inputs of the robot are its linear and angular velocities (v and w resp.). It is able to sense its location in the environment (the triplet $\langle x, y, \theta \rangle$). The MDP control policy requires the abstraction of this information as a symbolic state s. In response, it provides an adequate symbolic action a, which must be then interpreted by the robot as a sequence of appropriate control inputs.

In most cases, both sides of this abstraction are carried out in an *ad-hoc* approach, tailored to the task at hand and the particular policy that a given DT model is supposed to provide. This results, however, in a large amount of implementation-specific work by the problem designer, which is difficult to reuse in other, similar applications, and is subject to designer errors during deployment, which, in turn, are difficult to track, and may invalidate the DT model of the system and/or its solution.

The purpose of the MDM package is to provide researchers / autonomous systems engineers with an easy-to-use set of tools for the deployment of MDP-based decision-making methods to physical agents. Its features include:

- The ability to easily abstract system/environment characteristics as symbolic states or observations (for discrete MDPs or POMDPs, respectively);
- Supports single agent and multiagent systems;
- Generic callback-based action interpretation allows actions to be defined through ROS-native frameworks (e.g. *actionlib*[1]/SMACH);
- The ability to implement hierarchical MDPs/POMDPs;
- Supports synchronous (fixed-rate) and asynchronous (event-driven) execution strategies;

[1]URL: `http://wiki.ros.org/actionlib`

- Relevant execution information can easily be logged through ROS (actions, states, rewards, transition rates, etc.);
- MDM is based on the MADP Toolbox (Spaan and Oliehoek, 2008). MADP is a toolbox for decision-theoretic research, containing state-of-the-art solution algorithms for multiagent MDP-based models, and is actively maintained and extended by researchers in that field. MDM can potentially implement any model which can be defined through MADP.

MDM is currently available at `http://users.isr.ist.utl.pt/~jmessias/MDM/`. It is developed and maintained specifically as a package for the ROS middleware.

7.3.1 Terminology

In the remainder of this section, we will discuss the software architecture of the MDM Library. We will here overview the basic concepts and terminology of the ROS middleware that are necessary to that end.

In the ROS framework, processes, or *nodes*, run in a distributed, networked fashion. Each node represents an abstract producer or consumer of data. ROS abstracts the actual protocols for communication between nodes. This results in a modular framework where the physical location of each node in the network is irrelevant from the implementation standpoint – nodes running on the same machine communicate in the same way as nodes in different network locations. Nodes can communicate one-to-many / many-to-many, via *topics* (which can be thought of as streams of data), or one-to-one, via *services*. When communicating via topics, nodes can *publish* (send) data to a topic; and they can *subscribe* to a topic, which means that they register to receive all data flowing through it.

By default, nodes are expected to receive data asynchronously. The corresponding processes are woken up as they receive topic data or service calls. The data is processed by means of a suitable *callback* function.

For a more complete overview of ROS, see (Quigley et al., 2009).

7.3.2 MDM Overview

At its core, MDM is organized into a set of *Layers*, which embody the components that are involved in the decision-making loop of a generic MDP-based agent:

- The *System Abstraction Layer* – This is the part of MDM that constantly maps the sensorial information of an agent to the constituent abstract representations of a (PO)MDP model. This layer can be implemented, in turn, by either a *State Layer* or an *Observation Layer* (or both), depending on the observability of the system;

- The *Control Layer* – Contains the implementation of the policy of an agent, and performs essential updates after a decision is taken (e.g. belief updates);

- The *Action Layer* – Interprets (PO)MDP actions. It associates each action with a user-definable callback function.

MDM provides users with the tools to define each of these layers in a way which suits a particular decision-making problem, and manages the interface between these different components within ROS.

An MDM *ensemble* is an instantiation of a System Abstraction Layer, a Control Layer, and an Action Layer which, together, functionally implement an MDP, POMDP, or any other related decision-theoretic model. A basic MDM control loop is shown in Figure 7.7, which embodies a single-agent MDP. The inputs and outputs to each of the various ROS components are also made explicit.

In the following subsections, the operation of each of these core components is described in greater detail.

7.3.2.1 The State Layer

From the perspective of MDM and for the rest of this document, it is assumed that a robotic agent is processing or filtering its sensor data through a set of appropriate ROS nodes, in order to estimate relevant characteristics of the system as a whole (*e.g.* its localization, atmospheric conditions, etc.). This information can be mapped to a space of discrete factored states by a State Layer, if and only if there is no perceptual aliasing and there are no unobservable state factors. The State Layer constitutes one of the two possible System Abstraction Layers in MDM (the other being the Observation Layer).

Functionally, State Layers translate sets of *logical predicates* into factored, integer-valued state representations. For this purpose, MDM makes use of the concurrently

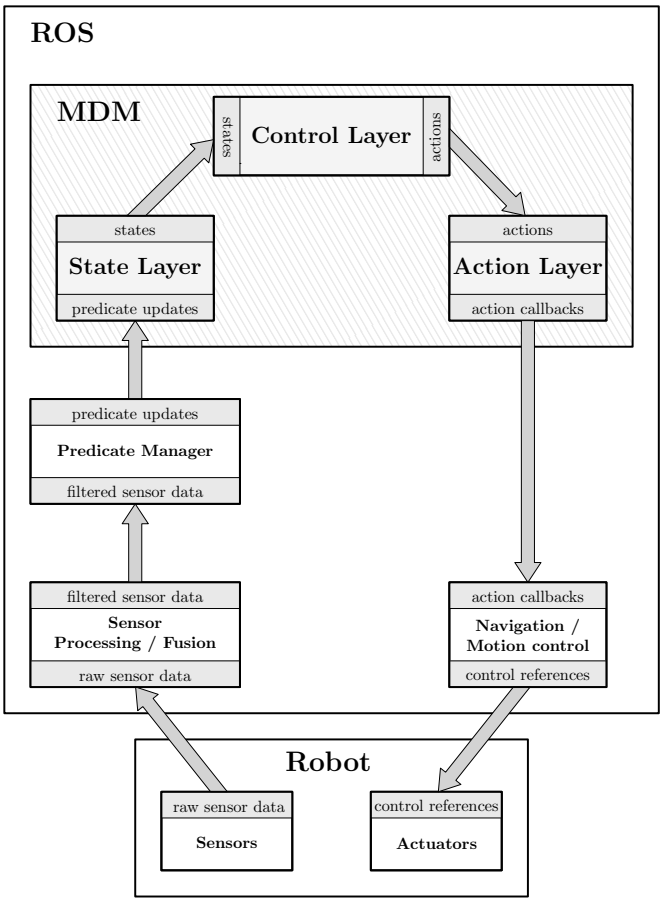


Figure 7.7: The control loop for an MDP-based agent using MDM.

developed *Predicate Manager* ROS package[1]. In its essence, Predicate Manager allows the user to easily define semantics for arbitrary logical conditions, which can either be directly grounded on sensor data, or defined over other predicates through propositional calculus. Predicate Manager operates asynchronously, and outputs a set of *predicate updates* whenever one or more predicates change their logical value (since predicates can depend on each other, multiple predicates can change simultaneously).

From the perspective of ROS/MDM, predicates are seen as named logical-valued structures. More formally, let p represent a predicate and $l_t(p) \in \{0, 1\}$ represent the logical value of p at some time $t \in \mathbb{R}_0^+$. Given a set of predicates $\mathcal{P} = \{p_1, \ldots, p_n\}$, and a factored state description $\mathcal{X} = \{\mathcal{X}_1 \ldots, \mathcal{X}_m\}$, the State Layer establishes a surjective

[1] Also available at `http://users.isr.ist.utl.pt/~jmessias/MDM/`. The Predicate Manager package was developed, and is currently being maintained, by J. Reis (ISR/IST).

map $\mathcal{H}_i : \{0,1\}^k \to \mathcal{X}_i$ for each state factor i, that is, it maps length-k logical vectors onto each discrete set $\mathcal{X}_i$. These logical vectors, in turn, are the values of k predicates in a given subset $\mathcal{P}' = \{p'_1, \ldots, p'_k\} \subseteq \mathcal{P}$. That is, such a vector can be taken at time t as $v_t^{\mathcal{P}'} \in \{0,1\}^k : [v_t^{\mathcal{P}'}]_i = l_t(p'_i)\, \forall i \in \{1, \ldots, k\}$.

Predicates can be mapped onto discrete state factors by a State Layer, in one of the following ways:

- A binary state factor can be defined by binding it directly to the value of a particular predicate. That is, for the i-th state factor and j-th predicate, $x_i|_t = l_t(p_j)\, \forall t \in \mathbb{R}_0^+$.
- An integer-valued factor can be defined by an ordered set of **mutually exclusive** predicates, under the condition that one (and only one) predicate in the set is true at any given time. The index of the true predicate in the set is taken as the integer value of the state factor. Formally, the given (sub)set of predicates $\mathcal{P}' \subseteq \mathcal{P}$ must be such that $l_t(p'_1) \vee l_t(p'_2) \vee \ldots \vee l_t(p'_k) = 1$ and $l_t(p'_u) \wedge l_t(p'_v) = 1 \implies u = v$, $\forall t \in \mathbb{R}_0^+$. Then, for each time t, $x_i|_t = \iota_t$ such that $\iota_t = \min\{k : l_t(p'_k) = 1\}$.

The State Layer always outputs the *joint* value of the state factors that it contains, *i.e.* a single integer value which unambiguously corresponds to an assignment of state factors. This minimizes the amount of information that needs to be passed between modules. However, for multiagent problems, note that this does not mean that the whole joint state of the system needs to be modeled in each State Layer for every agent. If each agent can only access *part* of the state space, for example in a Dec-MDP, then different State Layers can model different subsets of state factors.

7.3.2.2 The Observation Layer

In the partially observable case, sensorial information should be mapped to a space of discrete factored *observations*, by an *Observation Layer*. The resulting decision-making loop, which can be used for POMDP-driven agents, is shown in Figure 7.8.

The most significant difference of this abstraction layer with respect to a fully-observable State Layer is that, while in the latter case states are mapped directly from assignments of persistent predicate values; in an Observation Layer, observations are mapped from instantaneous predicate changes. This is consistent with our definition of

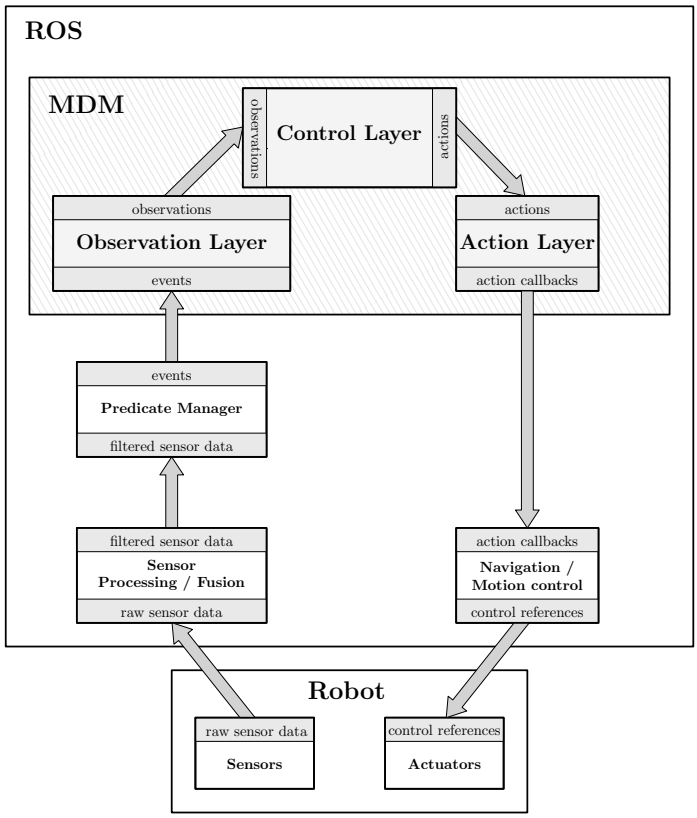


Figure 7.8: The basic control loop for a POMDP-based agent using MDM.

observations as event detections, in Chapter 6. The same rationale is here followed: an *observation* symbolizes a relevant occurrence in the system, typically associated with an underlying (possibly hidden) state transition. This event is captured by the system as a change in some conditional statement. The semantics of events, therefore, are defined over instantaneous conditional changes, as opposed to persistent conditional values.

The Predicate Manager package also makes it possible to define named *p-events* (short for "predicate-based event"). A named p-event is simply a label associated to a change in a predicate, or set of predicates[1]. These can be defined either as propositional formulas over existing predicates, in which case the respective p-event is triggered whenever that formula becomes true; or directly as conditions over other sources of data (*e.g.* a touch sensor signal).

[1]Note that these p-events are *not equivalent* to the "events" that are contemplated by Definition 2.2.2 and that were used throughout this work. In particular, if changes in interdependent predicates are labeled with different p-events, several of these p-events may trigger simultaneously. This means that an "event" (as per Definition 2.2.2) may actually be associated with *a set of* simultaneous p-events.

Observation spaces can also be factored. Observation factors are always associated with at least two p-events, and so their definition is similar to that of integer-valued state factors. Let $\Omega = \{\omega_1, \ldots, \omega_n\}$ be a set of p-events and $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_m\}$ a factored observation space description. An Observation Layer defines a mapping $\mathcal{G}_i : \mathrm{PS}(\Omega) \rightarrow \mathcal{O}_i$, for each observation factor i, that is, it maps from subsets of Ω directly onto each discrete domain $\mathcal{O}_i$. Let $\nu_t(\Omega) \subseteq \Omega \cup \emptyset$ represent the p-events in Ω which have triggered at time t (possibly none). Then, an observation factor i is defined through a set $\Omega' = \{\omega'_1, \ldots, \omega'_k\} \subseteq \Omega$ of ordered *asynchronous p-events* – that is, at most one p-event in Ω' triggers at any given instant ($\max_{t \in \mathbb{R}_0^+} |\nu_t(\Omega')| = 1$) – so that, iff $\nu_t(\Omega') \neq \emptyset$, $o_i|_t = \kappa_t$ with $\kappa_t = \min\{k : e'_k \in \nu_t(\Omega')\}$. Less formally, whenever a p-event is received, the value of the i-th observation factor is the index of the first (and supposedly only) active p-event in the associated p-event set.

Note that, although events are naturally asynchronous, this does not mean that synchronous decision-making under partially observability cannot be implemented through MDM. As it will be seen, only the Control Layer is responsible for defining the execution strategy (synchronous or asynchronous) of a given implementation.

As in the case of the State Layer, an Observation Layer always outputs the *joint* value of the observation factors that it contains.

7.3.2.3 The Control Layer

The Control Layer is responsible for parsing the policy of an agent or team of agents, and providing the appropriate response to an input state or observation, in the form of a symbolic action.

The internal operation of the Control Layer depends on the decision-theoretic framework which is selected by the user as a model for a particular system. For each of these frameworks, the Control Layer functionality is implemented by a suitably defined (multi)agent *controller*. Currently, MDM provides ready-to-use controllers for MDPs and POMDPs operating according to deterministic policies, which are assumed to be computed outside of ROS/MDM (using, for example, MADP).

For POMDPs, the Control Layer can also perform belief updates internally[1]. Con-

[1]Belief updates are performed internally if the POMDP Control Layer is connected to the output of an Observation Layer, which is the default case. Alternatively, POMDP Control Layers can also accept belief states directly, in which case no updates are carried out (see Section 7.3.3.1)

sequently, in such a case, the stochastic model of the problem (its transition and observation probabilities) must typically also be passed as an input to the controller (an exception is discussed in Section 7.3.3). The system model is also used to validate the number of states/actions/observations defined in the System Abstraction and Action Layers.

MDM Control Layers use the MADP Toolbox extensively to support their functionality. Their input files (policies, and the model specification if needed), can be defined in all MADP-compatible file types, and all decision-theoretic frameworks which are supported by MADP can potentially be implemented as MDM Control Layers. The MADP documentation describes the supported file types for the description of MDP-based models, and the corresponding policy representation for each respective framework.

The Control Layer of an agent also defines its real-time execution scheme. This can be either synchronous or asynchronous – synchronous execution forces an agent to take actions at a fixed, pre-defined rate; in asynchronous execution, actions are selected immediately after an input state or observation is received by the controller.

All agent controllers can be remotely started or stopped at run-time through ROS service calls. This allows the execution of an MDM ensemble to be itself abstracted as an "action".

7.3.2.4 The Action Layer

Symbolic actions can be associated with task-specific functions through an MDM Action Layer. In this layer, the user unambiguously associates each action of a given (PO)MDP with a target function (an action *callback*). That callback is triggered whenever a Control Layer publishes its respective action. An Action Layer can also interpret commands from a Control Layer as *joint actions*, and execute them under the scope of a particular agent.

The general purpose of an action callback is to delegate the control of the robotic agent to other ROS modules outside of MDM, operating at a lower level of abstraction. For example, action callbacks can be used to send goal poses to ROS native navigation modules; or to execute scripted sets of commands for human-robot interaction. However, the Action Layer also makes it possible to abstract other MDM ensembles

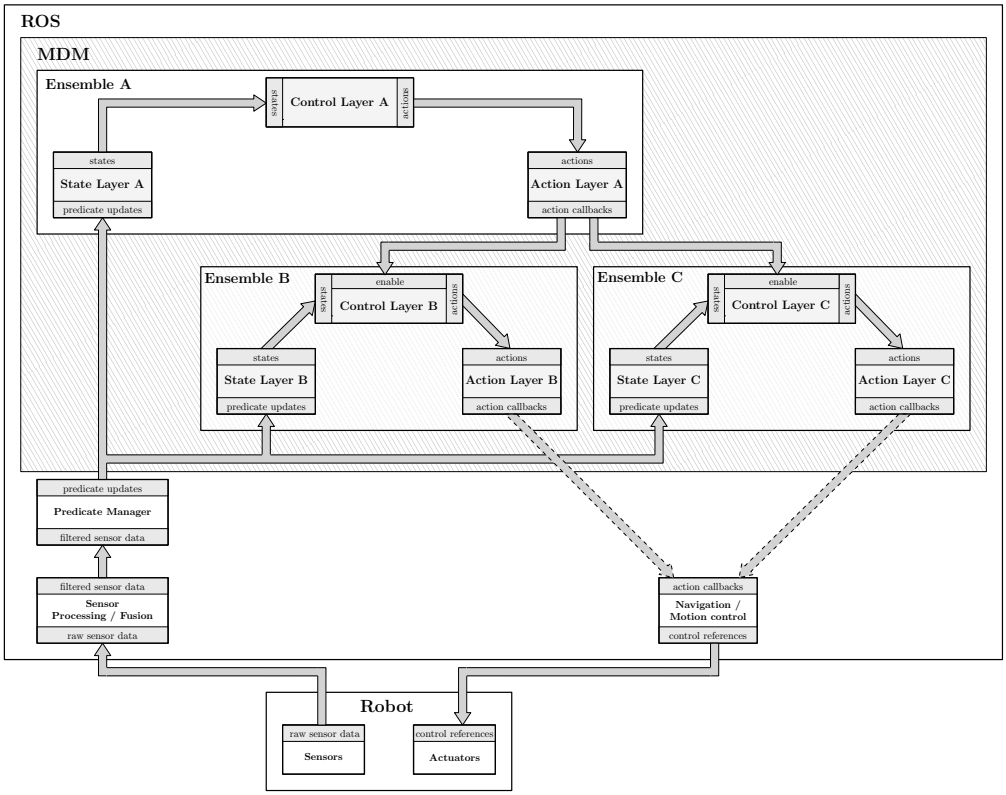


Figure 7.9: An example of the organization of a hierarchical MDP, as seen by MDM. The actions from Action Layer A enable / disable the controllers in ensembles B and C. When a controller is disabled, its respective State Layer still updates its information, but the controller does not relay any actions to its Action Layer.

as actions. This feature allows users to model arbitrarily complex hierarchical dependencies between MDPs/POMDPs (see Figure 7.9 for an example of the resulting node layout and respective dependencies).

The layered organization of MDM, and the "peer-to-peer" networked paradigm of ROS, allow action *execution* and action *selection* to be decoupled across different systems in a network, if that is desired. This can be accomplished by running an agent's Action Layer on a different network location than its respective Control Layer. For mobile robots, this means that the components of their decision-making which can be computationally expensive (sensor processing / abstraction, and stochastic model parsing, for example), can be run off-board. For teams of robots, this also implies that a single Control Layer, implementing a centralized multiagent MDP/POMDP policy, can be readily connected to multiple Action Layers, one for each agent in the system (see

Figure 7.10).

For the implementation of typical topological navigation actions (which are prevalent in applications of decision theory to robotics), MDM includes *Topological Tools*, an auxiliary ROS package (bundled with MDM). This package allows the user to easily define states and observations over a map of the robot's environment, such as those obtained through ROS mapping utilities. It also allows the abstraction of that map as a labeled graph, which in turn makes it possible to easily implement context-dependent navigation actions (*e.g.* "Move Up", "Move Down") in an MDM Action Layer.

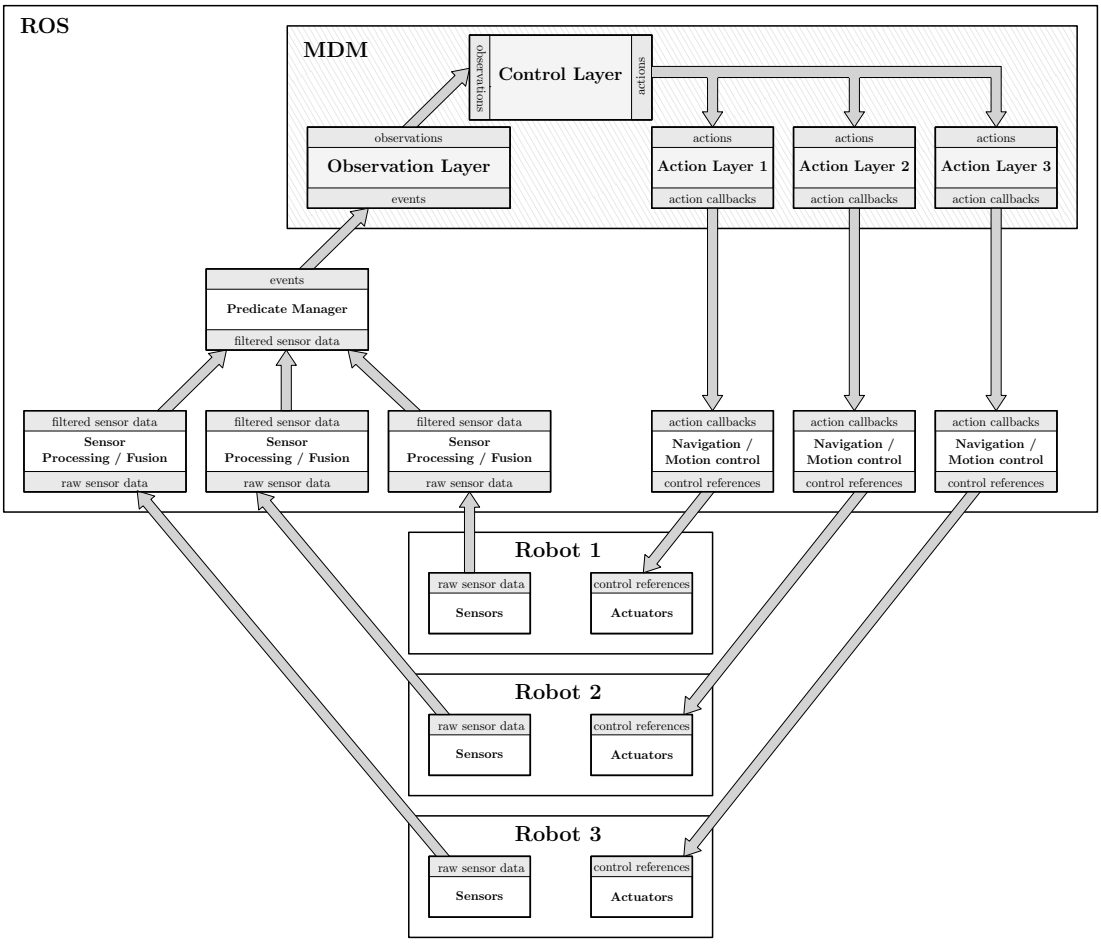


Figure 7.10: An MPOMDP implemented by a single MDM ensemble – in this case, there are multiple Action Layers. Each one interprets the joint actions originating from the centralized controller under the scope of a single agent. There is a single Observation Layer which provides joint observations, and a single Predicate Manager instantiation which fuses sensor data from all agents.

7.3.3 Deploying MDM: Considerations for Specialized Scenarios

The previous sections covered the internal organization of MDM and provided an overview of its implementation in generic scenarios. The present section discusses how MDM can be applied to scenarios with practical requirements which lie outside of the more common deployment schemes which have been presented so far.

7.3.3.1 POMDPs with External Belief States

In some scenarios, the probability distribution over the space state of the system can be handled indirectly, and continuously, by processes which can run independently of the agent's decision-making loop. This may be the case, for example, if a robot is running a passive self-localization algorithm, which outputs a measure of uncertainty over the robot's estimated pose, and if the user wants to use that estimate, at run-time, as a basis for the belief state of an associated POMDP.

In MDM, this execution scheme can be implemented by omitting the Observation Layer of a POMDP agent, and instead passing the estimated belief state directly to its Control Layer. For this purpose, POMDP Control Layers subscribe, by default, to a topic where the belief state can be externally specified. Upon receiving a belief estimate, the Control Layer can output its associated action, according to the provided policy file, either asynchronously or at a fixed temporal rate. A representation of this deployment scheme is shown in Figure 7.11.

There are, however, notable caveats to this approach:

- The description of the state space used by the Control Layer must be known by the belief state estimator. In particular, the notion of *state* that is assumed by the module which responsible for the belief estimation must be the same as that which is modeled in the POMDP. If this is not the case, then the probability distributions originating from the external estimator must first be projected onto the state space of the POMDP, which is not trivial. The system abstraction must be carried out within the belief state estimator;

- Algorithms which produce the belief state estimate directly from sensor data (*e.g.* self-localization) typically operate synchronously, at the same rate as that source data. This means that asynchronous, *event-driven* POMDP Control Layers are not well-defined in this case;

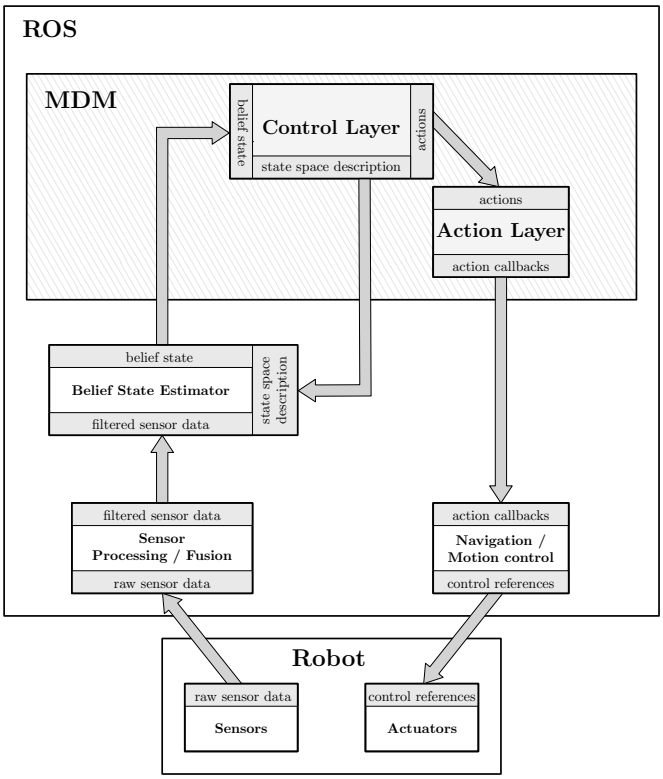


Figure 7.11: A deployment scheme for a POMDP-based agent where belief updates are carried out outside of MDM. The node that is responsible for the estimation of the belief state must have the same state space description as the Control Layer. The MDM ensemble is driven at the same rate as the belief state estimator, so it is implicitly synchronous with sensorial data.

- Planning is still assumed to be carried out prior to execution, and, during planning, the stochastic models of the POMDP are assumed to be an accurate representation of the system dynamics. If the external belief updates do not match the sequences of belief states predicted by the transition and observation models of the POMDP, then the agent can behave unexpectedly at run-time, since its policy was obtained using ill-defined models; if, on the other hand, the external belief updates are consistent with those which are predicted by the POMDP model, then a better option (in terms of the amount of work required for the deployment of the POMDP) is to simply carry out those belief updates internally in the Control Layer, as per the deployment scheme described originally in Section 7.3.2.2.

7.3.3.2 Multiagent Decision-Making with Managed Communication

For multiagent systems, the standard operation of ROS assumes that a single ROS *Master*[1] mediates the connection between the all of the nodes that are distributed across a network. After two nodes are connected, communication between them is peer-to-peer, and managed transparently by ROS, without providing the user the possibility of using custom communication protocols (for example, to limit the amount of network traffic). Therefore, in its default deployment scheme, the ROS Master behaves as a single point of failure for the distributed system, and more advanced communication protocols such as the one proposed in (Reis et al., 2013) cannot be used.

A more robust multiagent deployment scheme combines multiple ROS Masters, with managed communication between them (see Figure 7.12). Typically, each mobile robot in a multi-robot team will operate its own ROS Master. Topics which are meant to be shared between Masters through managed communication must be explicitly selected and configured by the user.

For MDM centralized multiagent controllers, which completely abstract inter-agent communication, using a multimaster deployment scheme simply means that a complete copy of its respective MDM ensemble must be running locally to each Master. Communication should be managed before the System Abstraction Layer, ideally by sharing Predicate Manager topics, so that each agent can maintain a coherent view of the system state, or of the team's observations. Note that, since each agent has local access to the joint policy, it can execute its own actions when given the joint state or observation.

For MDM controllers with explicit communication policies, communication should be managed at the output of the System Abstraction Layer, possibly with feedback from the Control Layer. The rationale in such a case is that states or observations are local to each agent (as opposed to being implicitly shared), and may not contain enough information for an agent to unambiguously determine its own action at the Control Layer. Consequently, the Control Layer should also be capable of fusing system information arriving from different sources.

[1] http://www.ros.org/wiki/Master

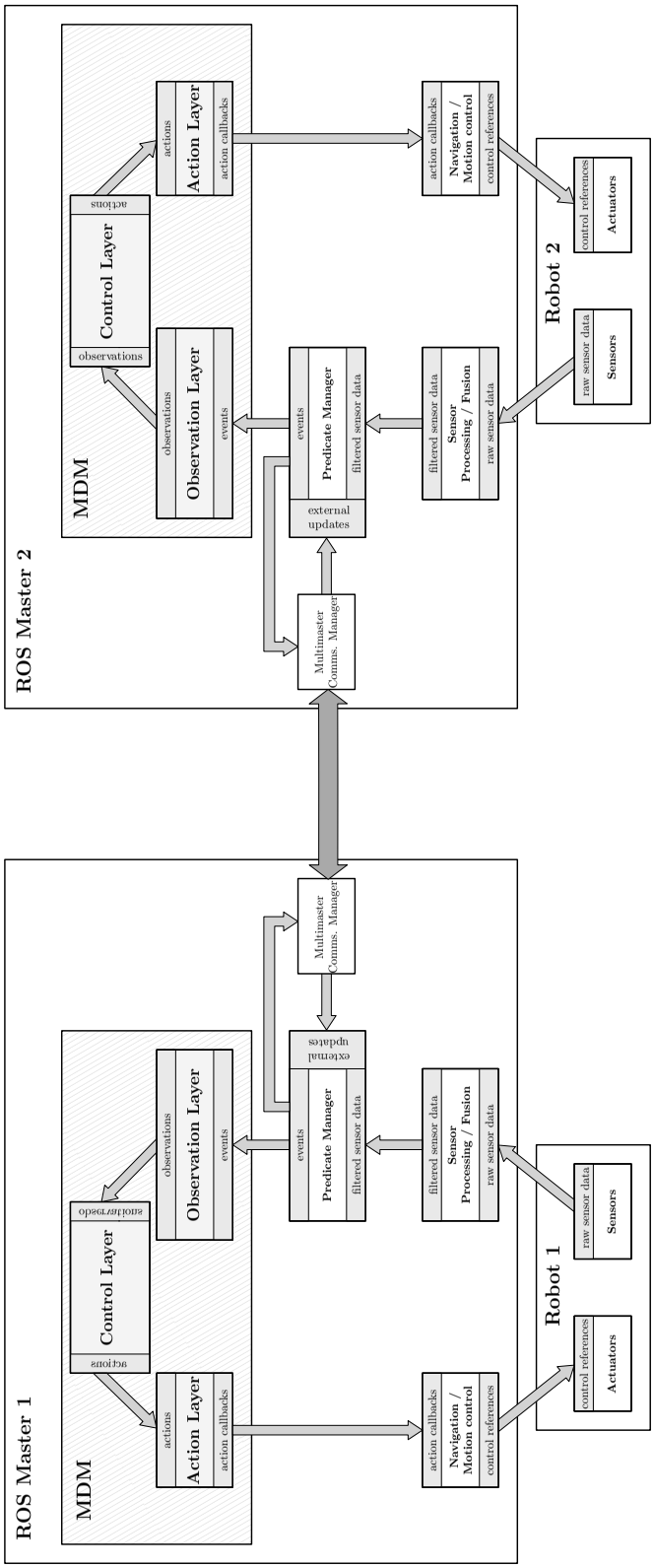


Figure 7.12: A multiagent MDM deployment scheme with multiple ROS Masters, and managed multimaster communication. Predicate Manager topics are explicitly shared between ROS Masters, so each Observation Layer still outputs *joint* observations. This is the approach which was taken in the MAIS+S project, using SocRob Multicast (Reis et al., 2013) to manage the multimaster communication.

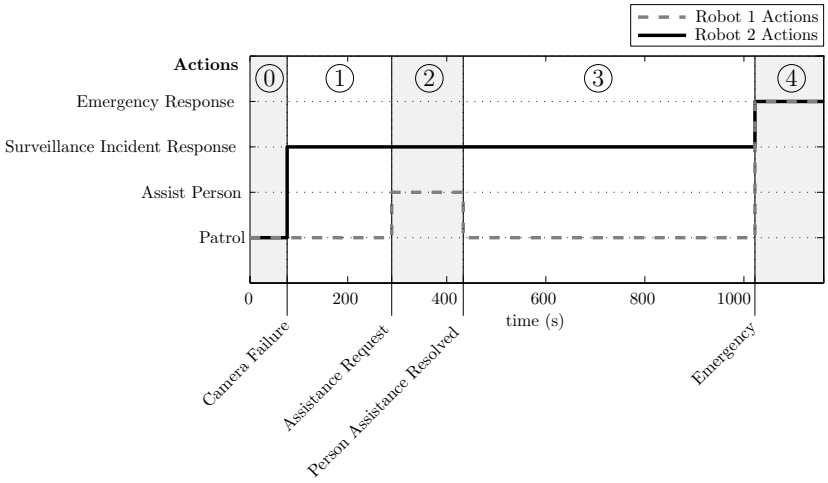


Figure 7.13: A timeline of actions and events in a trial run of our autonomous surveillance system. The steps of the decision-making process are identified at the top, and match the robot paths shown in Figure 7.14. The event detections, shown under the timeline, are experienced by both robots with negligible delay.

7.4 Results

The main results of our multi-robot experiments can be seen, in video format, at `http://users.isr.ist.utl.pt/~jmessias/PhDthesis`. Our results video also provides further description of our experimental setup for multiagent surveillance at ISR-Lisbon. In the remainder of this section, we will provide an outline of these results.

In Figures 7.13 and 7.14, we show the timeline of a trial execution of our Event-Driven MPOMDP policy. In this trial, the detection of a camera failure prompts one the robots to inspect that position, and replace the failed camera by assuming a stationary position while covering its field-of-view. This corresponds to action "Surveillance Incident Response" taken at step 1. Meanwhile, the other robot continues to patrol the environment, in the absence of any other events; In step 2, an assistance request is detected. Since one of the robots is already busy replacing the failed camera, the remaining agent (robot 1) decides to assist, and guides that person to a desired destination. Afterwards, the robot goes back to patrolling the environment until, at step 4, a fire detection is simulated, which causes both robots to abandon their active tasks and address the emergency immediately. The total runtime of this trial ($19m\ 18s$) is limited only by the battery lifetimes of both robots. We used infinite-horizon policies, with a discount rate of $\gamma = 0.95$, since there isn't a fixed limit to the number of steps that

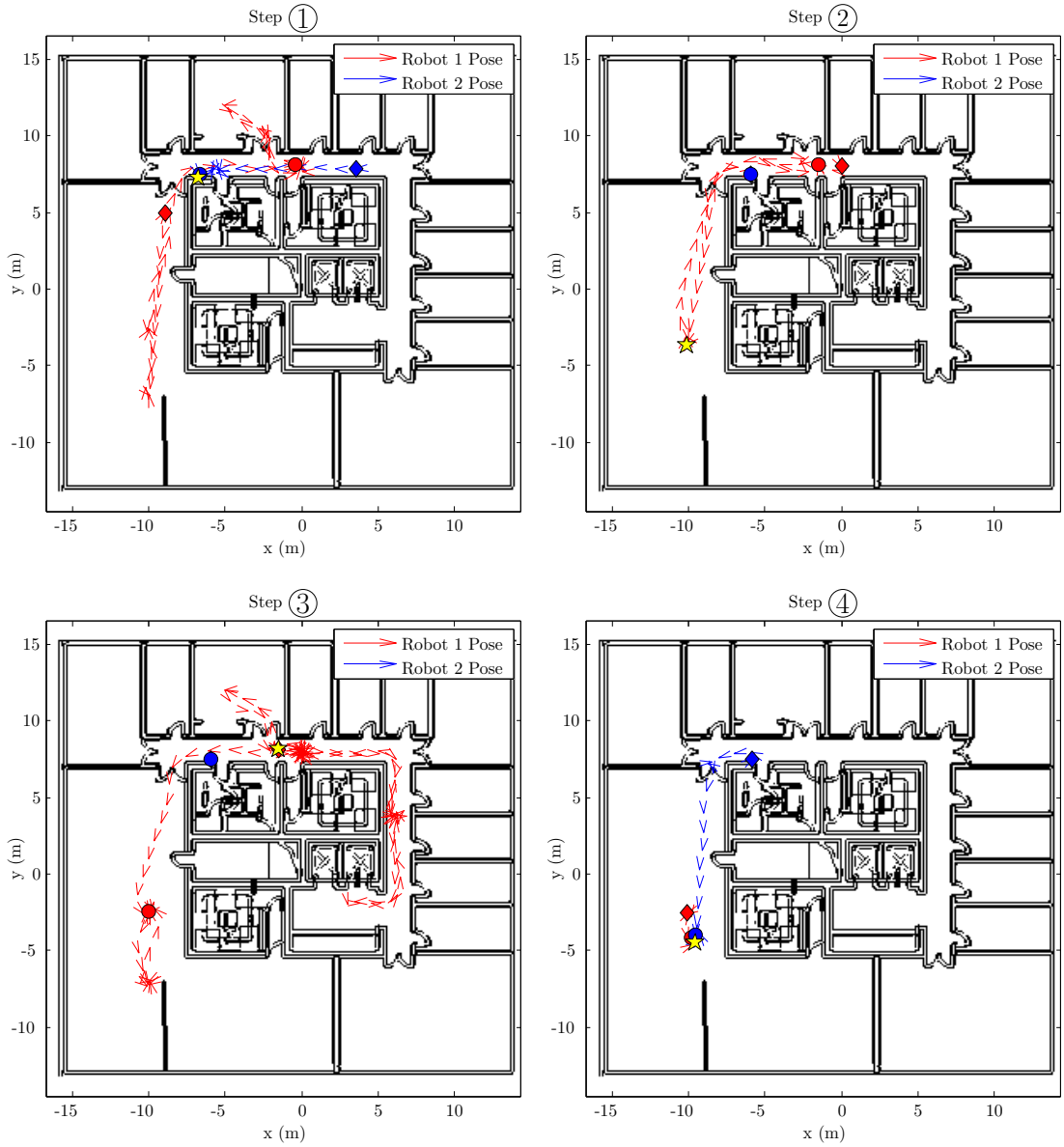


Figure 7.14: The paths traversed by the robots during a trial run, overlaid on the floorplan of their environment at ISR. Each snapshot represents a different decision step, identified at the top, which matches the timeline of actions and events shown in Figure 7.13. Step 0 is omitted. In each step, the initial position of each robot is represented by a diamond marker, and the final position by a circular marker. The position where each event was detected **at the beginning** of each step is also shown as a star-shaped marker. Step 1: Robot 1 patrols, while robot 2 converges to the position of a failed camera; Step 2: Robot 1 assists a person in need of guidance, while robot 2 remains stationary; Step 3: Robot 1 continues to patrol; Step 4: Both robots converge towards the position of a simulated fire.

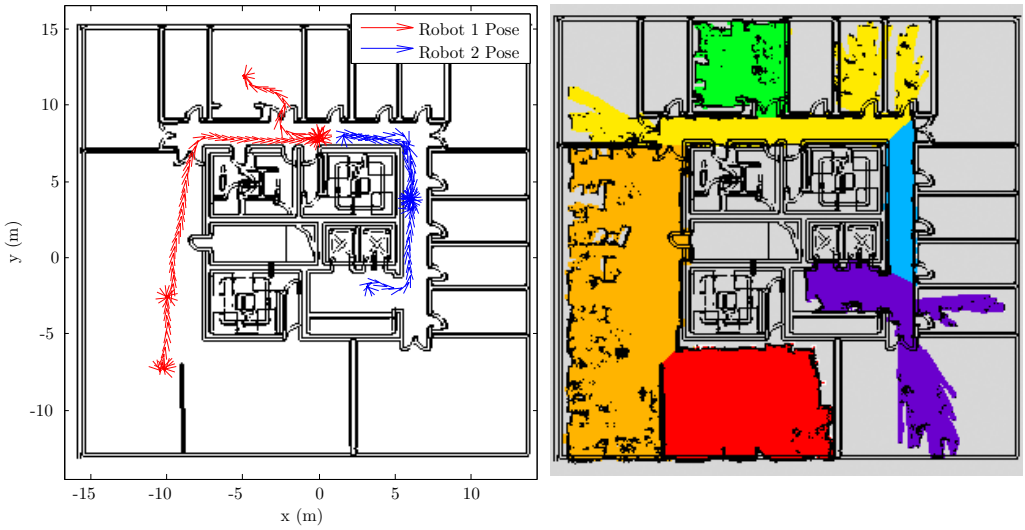
can be taken during operation, and since our run-time belief update (Eq. (6.16)) is, at this time, only applicable to infinite horizon policies, due to the inability to observe how many false negative events occur between each two decisions. In practice, however, a low number of decisions is taken in each run – in this example, only 4 decisions steps are taken. This very low frequency of decision-making steps, while maintaining near-instantaneous reactivity, can only be achieved, in the domain of MDP theory, by asynchronous frameworks such as ours, or related SMDP-based methodologies such as the *Options* framework (Sutton et al., 1999). In contrast to the latter, our framework has the added advantage of allowing decision steps with any real-valued temporal duration, and not just multiples of a fundamental time-step.

Since robots only communicate at event detection instants, an event-driven approach also keeps communication between agents to a strictly necessary minimum. Implicitly, 4 communication episodes were necessary for the coordination of the robot team in this trial run, each of them carrying only an integer value (the observation symbol) and the detection timestamp.

The ability of our robots to patrol their environment cooperatively, searching for events, is showcased in Figure 7.15, over a trial run of $2m\ 35s$. Recall that the patrol task is also an (Event-Driven) POMDP, so their behavior is not hardcoded. Each robot covers only part of the state space (shown in Figure 7.15b) to maximize search efficiency.

7.4.1 Realistic Simulations

We note that the experimental conditions in this real-world environment cannot be held constant, which, coupled with limited battery lifetime of our robots, does not allow for a statistically significant amount of real robot data to be collected, for a quantitative evaluation of the performance of our multi-robot system. Therefore, we have used a ROS-native simulation environment (Stage) as a platform for a systematic, and realistic, evaluation of the performance of our Event-Driven MPOMDP. This simulation considers the physical properties of both robots, and runs the same code for navigation and decision-making as the real multi-robot system. In Figure 7.16, we show a set of results that was obtained by simulating the arrival of visitors to ISR, and the consequent response by the robot team to assist those visitors, in the presence of uncertainty regarding the detection of assistance requests. Visitors always enter the environment and request assistance in same position, but only one visitor can be present at a time. Upon



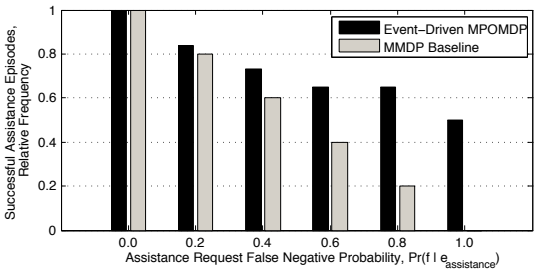
(a) Robot paths during cooperative patrol **(b)** The underlying partition into states

Figure 7.15: The behavior of our robot team when patrolling their environment cooperatively. (a) The paths covered by the robots after $2m$ $35s$ of execution. In this time, the robots covered all the relevant areas of their floor. A new round would trigger in the absence of other events. (a) The state space description for the patrol task consisted of a topological map where each node abstracts a relevant (reachable) area of the environment. Each of these nodes is here represented in a different color.

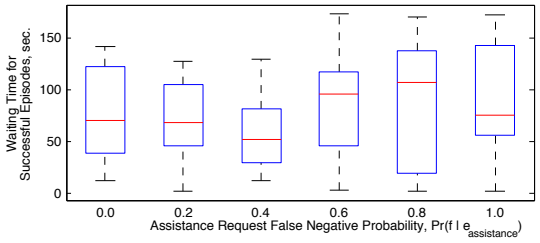
arriving, the visitor waits for a maximum of 3 minutes. A fixed camera overlooking the arrival position is expected to detect an event $e_{assistance}$, signaling that the visitor is waiting for guidance. If none of the robots show up during that period, the visitor leaves, unattended, and a new arrival time is sampled from an exponential distribution with $\lambda = 1/120$.

In Figure 7.16a, we show the frequency of successfully attended visitors, as a function of the probability of false negatives in their detection, $\Pr(o_f \mid e_{assistance})$. For each data point, we ran the realistic simulation for 4 hours (simulation time), to an average sample size of 45 arrivals. If no event detections are received for a sufficiently long time ($160s$), a "timeout" event triggers instead, forcing a new decision episode to take place, and belief update as per Eq. (6.16). This means that the likelihood that a visitor is waiting increases with time, even if $e_{assistance}$ is completely unobservable (see Figure 7.16c). If this likelihood is high, the robots will still trigger their "assistance response" behavior, and search for visitors at their arrival position. Therefore, the Event-Driven MPOMDP policy provides a higher rate of successfully attended visitors than a fully-observable

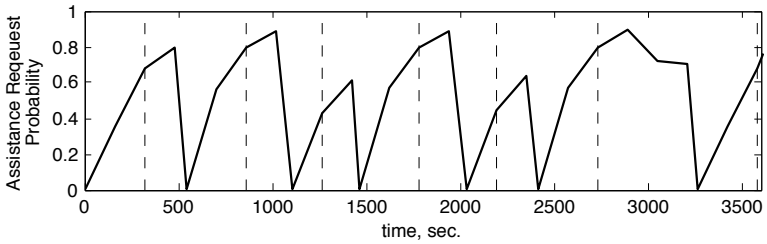
alternative, particularly (and as expected) as the probability of false negative detections increases. In reality, the probability of false negative detections of an assistance request by our surveillance cameras is 0.5, so this shows a clear advantage of a partially-observable formulation to this problem. In Figure 7.16b, we also show, as a box-plot, the waiting times of the visitors that were successfully attended, and we can see that there is no clear dependency between the rate of false negatives and (successul) service time.



(a) Assisted visitors vs. $\Pr(o_f \mid e_{assistance})$



(b) Waiting times for attended visitors



(c) Run-time belief propagation with $\Pr(o_f \mid e_{assistance}) = 1$

Figure 7.16: Results for a realistically simulated "visitor assistance" experiment. (a) The relative frequency of successfully attended visitors versus the probability of not detecting their arrival. The Event-Driven MPOMDP provides higher solution quality than the fully observable, theoretical baseline. (a) The waiting times of successfully attended visitors, which are not correlated with the false negative detection probability. (c) The likelihood that a person is waiting for assistance, in a sample run, and for 1 hour of execution, when arrivals are fully unobservable ($\Pr(o_f \mid e_{assistance}) = 1$). This was obtained by marginalizing the result of Eq (6.16). The instants in which one of the robots decided to search for visitors are marked.

7.5 Summary

In this chapter, we have supported the applicability of our Event-Driven MPOMDP methods to real multi-robot systems, by presenting the results of its implementation in a full scale, autonomous multiagent surveillance framework.

In doing so, we have described the software components that are essential to our implementation, and which include original contributions of this thesis work. The MDM Library, which is the most significant of these original software contributions, aims at facilitating the deployment of general DT methods to teams of robots.

Our multi-robot system, operating at ISR-Lisbon, was shown to cooperate by using an Event-Driven MPOMDP policy, with a low (average) frequency of decisions. Our approach requires minimal communication, while retaining solution quality and reactivity. Through realistic simulation, we saw that our event-driven system is robust to missed event detections.

Chapter 8

Conclusions

To conclude this thesis, we will review the main contributions of this work, and discuss potential directions for future research.

8.1 Contributions

In this work, we have studied the problem of decision-making under uncertainty for teams of real robots. We will now overview our most significant contributions towards that end:

Examining the limitations of Decision Theory, from the point-of-view of Robotics. At their core, DT methods are grounded on abstract, mathematical frameworks. Although they elegantly address the problems of planning and learning for autonomous agents, these methods are rarely directly applicable to scenarios involving physical environments and agents. The problems involved in the integration of symbolic DT methods, in systems with physical agents, are often overlooked, or approached in an *ad-hoc* manner. This has been the main motivation for the development of this thesis work. In Chapter 3, we have provided comprehensive, step-by-step guidelines for the process of designing and implementing a DT control policy for teams of robots. In doing so, we have also identified the most significant obstacles to this process.

Developing methods to manage communication in MPOMDPs. The complexity of DT methodologies that assume costly or unavailable communication limits their applicability to real multiagent systems. A popular approach to circumvent this issue is to try to separate the problems of planning over actions, and planning over

communication instances. That is, communication is assumed to be free during planning, and minimized *a posteriori*, if needed. Previously to this work, the only available methodologies to manage communication in MPOMDPs operated exclusively at run-time (Roth et al., 2005a). In Chapter 4, we have developed methods to obtain communication policies for MPOMDPs prior to execution. We have shown that our methods allow MPOMDP agents to act, given a joint value function, but based only on locally available information, when possible; and also allow agents to determine what additional information is necessary, when needed. We saw that, when using our methods to manage communication in scenarios with sparse interactions, the number of communication episodes is significantly reduced, and the run-time performance of the multiagent team is preserved.

Analyzing and implementing decision-making for teams of robots as an event-driven process. One of the most significant limitation of common multiagent DT methods is that they assume synchronized perception and action loops among all agents. In a team of (possibly heterogeneous) physical agents, synchronization is undesirable, since it induces loss of reactivity, and / or increases the horizon of the decision-making problem and communication frequency. In Chapter 5, we have explored the application of the GSMDP framework to a team of robots, by interpreting multiagent decision-making as an event-driven, possibly non-Markovian process. We have highlighted the limitations of the GSMDP framework, namely: that it assumes full observability over states and events; and that its approximation as a fully Markovian CTMDP is not always possible for robotics problems, due to the prevalence of events that follow temporal distributions that are not amenable to Phase-Type approximations. Even so, we have empirically demonstrated that event-driven DT policies are not only feasible for practical, cooperative robotics applications, but that they outperform synchronous MDP-based alternatives. This constitutes, to our knowledge, the first reported application of the GSMDP framework to a real multi-robot system.

Introducing frameworks for multiagent decision-making driven by partially observable events. In Chapter 6, we have taken a novel interpretation of MPOMDP dynamics, that considers the asynchronous and uncertain perception of the environment by the multiagent team. For fully Markovian systems, this interpretation was embodied in our Event-Driven MPOMDP framework. We have studied the implications of observing events, as opposed to states, on decision-making. In particular, we

have shown how, in systems driven by partially observable events, with free communication, the total number of observations to be considered grows *linearly* in the number of agents, as opposed to *exponentially* as is the case in synchronous MPOMDPs. This allows Event-Driven MPOMDP models to scale to larger domains than synchronous MPOMDP counterparts. Moreover, the Event-Driven MPOMDP framework explicitly considers the effects of false positive and false negative event detections. We have proved that such a model still retains the essential properties of a standard POMDP that allow their solution to be computationally efficient, and we have extended a standard POMDP point based-solver to operate in the event-driven setting. Finally, we have introduced an extended version of our framework for Semi-Markovian domains. The Event-Driven MPOSMDP framework allows the methodologies of fully observable GSMDPs to be applied to domains with partially observable events. Our event-driven frameworks were developed, from their onset, with the explicit purpose of providing suitable models for decision-making in teams of robots.

Supporting the application of DT methods to general Robotics domains. In Chapter 7, we described the application of our Event-Driven MPOMDP framework to a multi-robot, networked surveillance system. This implementation has not only provided evidence of the applicability of our methodologies to real teams of robots, but it has also resulted in the development of software tools with the purpose of aiding the deployment of DT frameworks and methods to multi-robot systems.

In Table 8.1, we organize the most important topics of the contributions of this work, along with the relevant properties of the case studies in which they were evaluated, and the respective publications in which these contributions were introduced, when applicable.

8.2 Future Work

Perhaps the most important omission in this thesis has been regarding the application of reinforcement learning methods to obtain DT policies directly from the interaction of robotic agents with their environment. One of the issues that was identified in Chapter 3, pertaining to the implementation of model-based MDP frameworks in real environments, was their reliance on the knowledge of the stochastic model parameters,

Chapter	Contributions	Case-Study Properties	Publications
3	Examined the limitations of DT for multi-robot systems	Discrete MPOMDP - Partially Observable - Fixed Time-Step (Unrestricted Comms.)	—
4	A method to determine offline comm. policies for MPOMDPs	Discrete MPOMDP - Partially Observable - Fixed Time-Step (Minimal Comms.)	(Messias et al., 2011)
5	Applied GSMDPs to real robot team	GSMDP / MMDPs - Fully Observable - Asynchronous	(Messias et al., 2013a)
6	Extended Event-Driven paradigm to Partially Observable settings	Event-Driven MPOMDP - Partially Observable - Asynchronous	(Messias et al., 2013b) (Messias et al., 2013c)
7	Real-world Event-Driven MPOMDP application & software	Event-Driven MPOMDP - Partially Observable - Asynchronous	—

Table 8.1: A summary of the contributions of this work, their respective chapters, and resulting international publications.

and their inflexibility with respect to changes in those parameters. Reinforcement learning, and especially *off-policy* learning methods, could be used to address that problem, allowing efficient decision-making in complex scenarios, in which explicitly modeling the dependencies between all of the involved variables is not feasible. The main reason for the omission of the application of reinforcement learning methods in this work is that we have attempted to remain abstracted from particular solution methods for the DT models that we have considered. Rather, we have focused on the structure and properties of the models themselves. Most of the advances presented in this work have followed directly from questioning the assumptions of DT frameworks, and whether they are valid for problems involving robots or other physical agents. Our conjecture was that, once a suitable framework was defined that embodied the ideal characteristics for multi-robot decision-making (such as the Event-Driven PO(S)MDP framework(s)), then, appropriate methods for planning and/or learning could be defined over that framework. One of our most immediate directions for future research is, therefore, the development of reinforcement learning methodologies for the modeling frameworks that we have proposed in this work. We are aware, however, that multiagent reinforcement learning is still

an open problem, particularly for partially observable environments, where established solutions that can be scaled to realistically-sized domains are still lacking.

Regarding our proposed Event-Driven MPOSMDP framework, and as we have noted in Section 6.5, we have still to collect empirical results, since we have not yet defined an appropriate belief update mechanism that can be applied, at run-time, by an agent controlled by an Event-Driven MPOSMDP policy. This constitutes a short-term step to be taken in our future research.

Another topic that can be explored is the integration of the communication reduction methods presented in Chapter 4, with the event-driven frameworks that were studied and proposed in Chapters 5 and 6. The former approach was developed specifically for synchronous decision-making, as we had not yet considered the alternative of having event-driven dynamics; as such, there are important obstacles to this integration. The most prominent of these is that, as we saw in Section 6.2.5, asynchronous models cannot be strictly described by Dynamic Bayesian Networks without introducing new state factor variables that indirectly correlate all others. Therefore, belief state propagation cannot be decentralized using, for example, the Factored Frontier algorithm, which means that joint communication is always necessary to update belief states. However, it remains to be seen whether any alternative mechanism, that does not require the propagation of factored belief states, could still exploit the structure of a joint value function to avoid extraneous communication.

As we have noted in Chapter 5, GSMDPs are difficult to solve in their natural form: the possibly non-Markovian state transitions contemplated by these models imply that Dynamic Programming methods cannot be directly applied. The solution that we have studied in Chapter 5, of approximating GSMDPs as SMDPs, by replacing non-Markovian temporal distributions with fully-Markovian Phase-Type distributions (proposed in (Younes and Simmons, 2004)), is not always applicable, as we had noted, due to the fact that many events in Robotics applications are governed by distributions with low coefficient of variation, which require unreasonably many phases for an approximation. As future research, we will investigate the use of Bilateral Phase-Type distributions (Ahn and Ramaswami, 2005), which can potentially approximate a broader class of non-Markovian events. Furthermore, we will also study the applicability, to teams of robots, of alternative algorithms that can operate directly on GSMDPs such as that of Rachelson et al. 2008.

Appendix A

Supporting Material

In this appendix, we present auxiliary material regarding the definition of the states, actions and observations of the various DT models that were used as the case-studies of this thesis. The problem files describing all of the models in this Appendix can be found at `http://users.isr.ist.utl.pt/~jmessias/PhDthesis`. Here, we omit the stochastic model parametrizations (transition, observation and reward functions), due to their size, although we encourage the reader to inspect the model definition at the online repository.

A.1 Robotic Soccer Case-Study

Here, we describe the state, action and observation spaces for our Robotic Soccer case-study, which was cast as an MPOMDP in Chapter 3, and as a GSMDP and comparable MMDPs in Chapter 5.

A.1.1 Partially Observable Formulation (Chapter 3)

Figure A.1 represents the factored state space of the Robotic Soccer MPOMDP. Its action and observation spaces are shown in Figure A.2.

$$\text{Role Assignment: } \mathcal{X}_1 = \begin{cases} \text{Attacker – Supporter} \\ \text{Supporter – Attacker} \end{cases}$$

$$\text{Supporter State: } \mathcal{X}_2 = \begin{cases} \text{Ready to Receive Pass} \\ \text{Blocked by Obstacles} \\ \text{Not Ready to Receive Pass} \end{cases}$$

$$\text{Attacker State: } \mathcal{X}_3 = \begin{cases} \text{Shooting Opportunity} \\ \text{Near Goal – Cleared to Move} \\ \text{Near Goal – Blocked by Obstacles} \\ \text{Opponent's Half – Cleared to Move} \\ \text{Opponent's Half – Blocked by Obstacles} \\ \text{Own Half – Cleared to Move} \\ \text{Own Half – Blocked by Obstacles} \\ \text{Without the Ball} \end{cases}$$

Figure A.1: State space description for the MPOMDP instantiation of our robotic soccer case study.

$$\mathcal{A}_1 = \mathcal{A}_2 = \begin{cases} \text{Dribble} \\ \text{Shoot} \\ \text{Pass} \\ \text{Recover} \\ \text{PrepareForPass} \\ \text{FindClearance} \end{cases} \qquad \mathcal{O}_1 = \mathcal{O}_2 = \begin{cases} \text{Ready} \\ \text{HasBallNearGoal} \\ \text{HasBallOppHalf} \\ \text{HasBallOwnHalf} \\ \text{Blocked} \\ \text{Closest} \\ \text{Second Closest} \end{cases}$$

Figure A.2: Action space description, left, and observation space description, right, for the MPOMDP instantiation of our robotic soccer case study.

A.1.2 Fully Observable Formulation (Chapter 5)

The state space for our Robotic Soccer GSMDP is shown in Figure A.3. The respective action and observation spaces are represented in Figure A.4. Note that the synchronous MMDPs that were used as a baseline for our comparative results have the same action and observation spaces, and its state space contains the same state factors $\mathcal{X}_{\{1,2,3,4\}}$ (that is, the "Phase Variable" state factor is excluded).

$$\text{Role Assignment: } \mathcal{X}_1 = \begin{cases} \text{Attacker – Supporter} \\ \text{Supporter – Attacker} \end{cases}$$

$$\text{Supporter State: } \mathcal{X}_2 = \begin{cases} \text{Ready to Receive Pass} \\ \text{Blocked by Obstacles} \\ \text{Not Ready to Receive Pass} \end{cases}$$

$$\text{Attacker State: } \mathcal{X}_3 = \begin{cases} \text{Shooting Opportunity} \\ \text{Near Goal – Cleared to Move} \\ \text{Near Goal – Blocked by Obstacles} \\ \text{Far From Goal – Cleared to Move} \\ \text{Far From Goal – Blocked by Obstacles} \\ \text{Without the Ball} \\ \text{Not Seeing the Ball} \end{cases}$$

$$\text{Phase Variable: } \mathcal{X}_4 = \begin{cases} \text{Phase 0} \\ \text{Phase 1} \\ \text{Phase 2} \end{cases}$$

Figure A.3: State space description for the GSMDP instantiation of our robotic soccer case study.

$$\mathcal{A}_1 = \mathcal{A}_2 = \begin{cases} \text{Dribble} \\ \text{Shoot} \\ \text{Pass} \\ \text{Recover} \\ \text{PrepareForPass} \\ \text{FindClearance} \end{cases}$$

Figure A.4: Action space description for the GSMDP and MMDP instantiations of our robotic soccer case study.

A.2 Synchronous MPOMDP Case-Studies (Chapter 4)

In this section, we describe the state, action and observation spaces for the simulated decision-making problems that were used as case-studies for our communication reduction methodologies.

$$\text{Left Room: } \mathcal{X}_L = \begin{Bmatrix} \text{L1} \\ \text{L2} \end{Bmatrix}$$
$$\text{Right Room: } \mathcal{X}_R = \begin{Bmatrix} \text{R1} \\ \text{R2} \end{Bmatrix}$$

Figure A.5: State space description for the *Relay-Small* MPOMDP.

$$\mathcal{A}_1 = \mathcal{A}_2 = \begin{Bmatrix} \text{Shuffle} \\ \text{Sense} \\ \text{Exchange} \end{Bmatrix} \qquad \mathcal{O}_1 = \mathcal{O}_2 = \begin{Bmatrix} \text{Door} \\ \text{NoDoor} \\ \text{Idle} \end{Bmatrix}$$

Figure A.6: Action space description, left, and observation space description, right, for the *Relay-Small* MPOMDP.

$$\text{Top Room: } \mathcal{X}_1 = \begin{Bmatrix} \text{L1} \\ \text{CL1} \\ \text{CR1} \\ \text{R1} \end{Bmatrix}$$
$$\text{Agent 1 Package: } \mathcal{X}_2 = \begin{Bmatrix} \text{NoPackage} \\ \text{Package} \end{Bmatrix}$$
$$\text{Bottom Room: } \mathcal{X}_3 = \begin{Bmatrix} \text{L2} \\ \text{CL2} \\ \text{CR2} \\ \text{R2} \end{Bmatrix}$$
$$\text{Agent 2 Package: } \mathcal{X}_4 = \begin{Bmatrix} \text{NoPackage} \\ \text{Package} \end{Bmatrix}$$

Figure A.7: State space description for the *Relay-Large* MPOMDP.

$$\mathcal{A}_1 = \mathcal{A}_2 = \begin{Bmatrix} \text{East} \\ \text{West} \\ \text{Interact} \end{Bmatrix} \qquad \mathcal{O}_1 = \mathcal{O}_2 = \begin{Bmatrix} \text{Corner - NoPackage} \\ \text{Corridor - NoPackage} \\ \text{Corridor - Package} \\ \text{Corner - Package} \end{Bmatrix}$$

Figure A.8: Action space description, left, and observation space description, right, for the *Relay-Large* MPOMDP.

$$\mathcal{X}_1 = \mathcal{X}_2 = \{1, \ldots, 7\}$$

$$\mathcal{A}_1 = \mathcal{A}_2 = \left\{ \begin{array}{c} \text{North} \\ \text{East} \\ \text{South} \\ \text{West} \end{array} \right. \qquad \mathcal{O}_1 = \mathcal{O}_2 = \left\{ \begin{array}{c} \text{Left} \\ \text{Right} \\ \text{Both} \\ \text{Nothing} \end{array} \right.$$

Figure A.9: State, action, and observation space description for the *OneDoor* MPOMDP.

A.3 Multiagent Surveillance Case-Study (Chapter 7)

In this section, we describe the decision-making models that were used in our multiagent surveillance case-study.

A.3.1 Event-Driven (M)POMDP Descriptions

Here, the state, action and observation spaces for the Event-Driven (M)POMDPs that were referred in Section 7.2.2 are described.

A.3.1.1 Coordinative (Top-Level) Event-Driven MPOMDP

The graphical model for the Coordinative Event-Driven MPOMDP that was used to allocate tasks in our multi-robot team is shown in Figure A.10. Note that this graphical model was designed according to the method described in Section 6.2.5, that is, by introducing an additional state variable (the "Event Prior" variable) that models the mutual exclusivity of every other variable in the state description. The actual factored state space description, excluding this virtual variable (since it can be marginalized), is represented in Figure A.11. Action and observation spaces are described in Figure A.12.

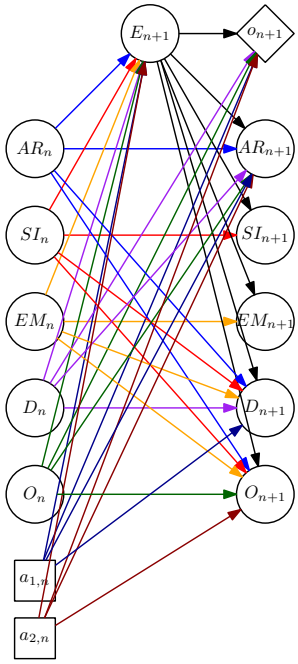


Figure A.10: The 2-DBN for our Coordinative Event-Driven MPOMDP, which assigns tasks to each robot. Outgoing connections from the same node at time n are represented with the same color, for better visibility. The represented state factor variables are: "Assistance Requested" (AR'); "Surveillance Incident" (SI); "Emergency" (EM); "Duke Status" (D); "Orwell Status" (O); "Event Prior" (E).

$$\text{Assistance Requested: } \mathcal{X}_1 = \begin{cases} \text{No} \\ \text{Duke Assisting} \\ \text{Orwell Assisting} \\ \text{Yes} \end{cases}$$

$$\text{Surveillance Incident: } \mathcal{X}_2 = \begin{cases} \text{No} \\ \text{Yes} \end{cases}$$

$$\text{Emergency: } \mathcal{X}_3 = \begin{cases} \text{No} \\ \text{Yes} \end{cases}$$

$$\text{Duke Status: } \mathcal{X}_4 = \begin{cases} \text{Disabled} \\ \text{Idle} \\ \text{Busy} \end{cases}$$

$$\text{Orwell Status: } \mathcal{X}_5 = \begin{cases} \text{Disabled} \\ \text{Idle} \\ \text{Busy} \end{cases}$$

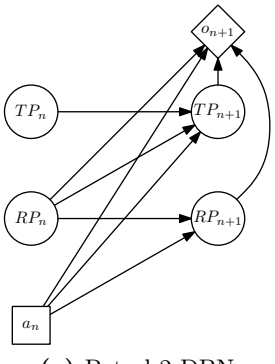
Figure A.11: State space description for our Coordinative Event-Driven MPOMDP.

$$\mathcal{A}_1 = \mathcal{A}_2 = \left\{ \begin{array}{c} \text{Patrol} \\ \text{Assist Person} \\ \text{Surveillance Incident Resp.} \\ \text{Emergency Response} \end{array} \right. \qquad \mathcal{O} = \left\{ \begin{array}{c} \text{False Negative} \\ \text{Timeout} \\ \text{Waving - Low Confidence} \\ \text{Waving - High Confidence} \\ \text{Surveillance Incident} \\ \text{Emergency} \\ \text{Person Assistance Resolved} \\ \text{Surveillance Incident Resolved} \\ \text{Emergency Resolved} \\ \text{Duke Task Cancelled} \\ \text{Orwell Task Cancelled} \\ \text{Duke On / Off} \\ \text{Orwell On / Off} \end{array} \right.$$

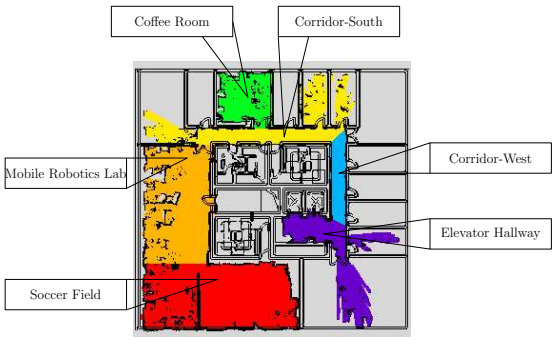
Figure A.12: Action space description, left, and observation space description, right, for the Coordinative Event-Driven MPOMDP.

A.3.1.2 Patrol Task Event-Driven POMDP

The "Patrol" task Event-Driven POMDP was implemented as the graphical model shown in A.13a. Here, we did not use the method of Section 6.2.5, since the state variables are already mutually exclusive **when conditioned on the actions of the agent**. That is, for the "Capture Target" and "Spawn Target" actions, only the "Target Position" variable will change. For all other actions, only the "Robot Position" variable will change. In Figure A.14 we show the factored state description for this problem, and in Figure A.15 we describe its action and observation spaces.



(a) Patrol 2-DBN



(b) State Space Labels

Figure A.13: (a): The 2-DBN for the "Patrol" Event-Driven POMDP. The represented state factor variables are: "Robot Position" (RP); "Target Position" (TP). (b): The labels associated to our topological abstraction of the area of operation.

$$\text{Robot Position: } \mathcal{X}_1 = \begin{cases} \text{Elevator Hallway} \\ \text{Corridor-West} \\ \text{Coffee Room} \\ \text{Corridor-South} \\ \text{Mobile Robotics Lab} \\ \text{Soccer Field} \end{cases}$$

$$\text{Target Position: } \mathcal{X}_2 = \begin{cases} \text{No Target} \\ \text{Elevator Hallway} \\ \text{Corridor-West} \\ \text{Coffee Room} \\ \text{Corridor-South} \\ \text{Mobile Robotics Lab} \\ \text{Soccer Field} \end{cases}$$

Figure A.14: State space description for our "Patrol" task Event-Driven POMDP. See Figure A.13b for the semantic grounding of these labels.

$$\mathcal{A} = \begin{cases} \text{Move Down} \\ \text{Move Right} \\ \text{Move Up} \\ \text{Move Left} \\ \text{Capture Target} \\ \text{Spawn Target} \end{cases} \qquad \mathcal{O} = \begin{cases} \text{Elevator Hallway Clear} \\ \text{Corridor-West Clear} \\ \text{Coffee Room Clear} \\ \text{Corridor-South Clear} \\ \text{Mobile Robotics Lab Clear} \\ \text{Soccer Field Clear} \\ \text{Found Target} \end{cases}$$

Figure A.15: Action space description, left, and observation space description, right, for the "Patrol" task Event-Driven POMDP.

A.3.2 Finite State Machines

In this section, we present the Finite State Machines (FSMs) that were used for the "Assistance Response", "Surveillance Incident Response" and "Emergency Response" tasks. In these graphs, nodes represent states of the FSM and labeled arrows represent transitions following the respective event. We note that, since these FSMs were implemented with the SMACH ROS package, some of the terminology over events is specific to that software: "preempted" events imply that the execution of a given procedure was overridden by another call; "aborted / cancelled" events signify that the robot failed to complete the procedure. All of the presented FSMs are initialized in a "Monitoring" state, where they remain until they receive an "acting" service request, which corresponds to the selection of that action by the coordinative Event-Driven MPOMDP.

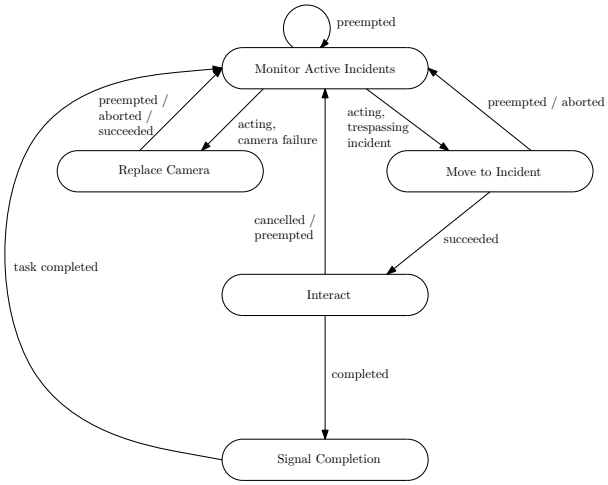


Figure A.17: The FSM for the "Surveillance Incident Response" task.

In Figure A.16, we show the FSM for the "Emergency Response" task. The robot is expected to move to the position of the latest detected emergency, and wait for the other robot for a fixed amount of time.

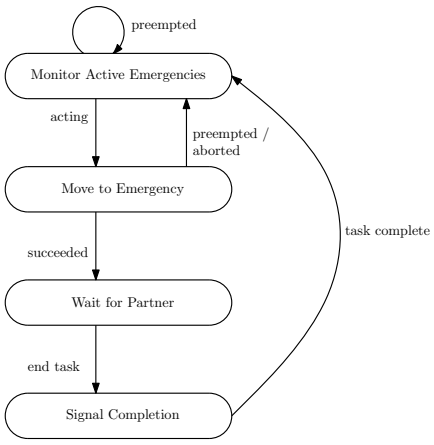


Figure A.16: The FSM for the "Emergency Response" task.

The "Surveillance Incident Response" FSM is shown in Figure A.17. Depending on the detected incident, the robot can either replace a failed camera or move to the position of a person trespassing in a restricted area. In the latter case, it should interact with that person to conclude the task.

Finally, we show the "Assistance Response" FSM is shown in Figure A.18. When a "assistance request" event is detected, the robot should first confirm that it is the

closest agent that is attempting to respond to that event. If so, it should move to the position of the requesting person and interact, by offering guidance to a desired location. Robots can also decide to offer assistance to visitors without having received an assistance request event (which corresponds to the "acting, no detection" transition).

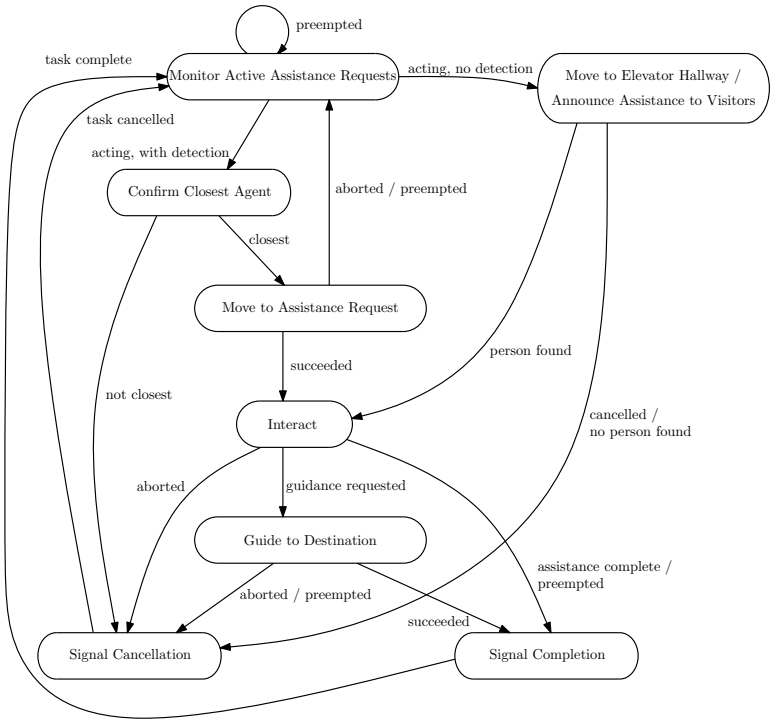


Figure A.18: The FSM for the "Assistance Response" task.

Appendix B

Implementation Examples for the MDM Library

In this appendix, we present implementation examples for the Markov Decision Making (MDM) Library (see Section 7.3). In particular, we show how to implement the State, Observation, Control, and Action Layers that can be used to create an MDM *ensemble* in order to deploy a given DT control strategy.

These `C++` examples assume that the reader is familiar with the implementation of simple ROS nodes in that programming language, and with basic ROS concepts and terminology (namely *topics*, *services*, *parameters* and *namespaces*).

B.1 Implementing a State Layer

The following example demonstrates how a State Layer ROS node can be implemented in `C++`, for a small scenario in the context of the surveillance network described in Chapter 7. There are two state factors in the problem: the first is an integer-valued state factor describing the topological location of a robot in its environment; the second is a binary variable representing whether or not there is a person waiting for assistance.

```
#include <ros/ros.h>

#include <markov_decision_making/StateLayer.h>

using namespace ros;
using namespace markov_decision_making;

int main (int argc, char** argv)
{
  init (argc, argv, "state_layer_example");

  StateLayer robot_mls;
  robot_mls.addStateFactor (StateDep()
                            .add ("IsInElevatorHallway")
                            .add ("IsInWestCorridor")
                            .add ("IsInSouthCorridor")
                            .add ("IsInCoffeeRoom")
                            .add ("IsInLRM")
                            .add ("IsInSoccerField"));
  robot_mls.addStateFactor (StateDep()
                            .add ("PersonIsWaiting"));
  spin();

  return 0;
}
```

We will now analyze the code (besides the bare minimum ROS node initialization / support):

```
StateLayer robot_mls;
```

This declares our State Layer, which will be silently connecting to Predicate Manager topics (`~/predicate_map` and `~/predicate_updates`), but will *not* be spinning its own thread. Note that the `spin()` function is still handled externally, and it is only called *after* state factor variables are declared. The State Layer will be publishing its state information to the `~/state` topic.

```
robot_mls.addStateFactor (StateDep()
                          .add ("IsInElevatorHallway")
                          .add ("IsInWestCorridor")
                          .add ("IsInSouthCorridor")
                          .add ("IsInCoffeeRoom")
                          .add ("IsInLRM")
                          .add ("IsInSoccerField"));
```

This adds an integer state factor to the State Layer, and binds its value to a set of mutually exclusive predicates which describe whether or not the robot is in a particular topological position. State factors must be added in the order that the user wants them to appear in the factored state description - that is, this code snippet will be considered as state factor $\mathcal{X}_1$. Likewise, predicates are added as dependencies in the order that they should be assigned to state factor values – `"IsInElevatorHallway"` will correspond to the first value of this state factor. The `StateDep()` class is used to easily register a chain of predicates as dependencies. Note that, following the `C++` standard and to maintain consistency with the internal operation of MDM, **all indexes start at 0**. This means that the domain of this state factor is $\mathcal{X}_1 = \{0, \ldots, 5\}$.

```
robot_mls.addStateFactor (StateDep()
                            .add ("PersonIsWaiting"));
```

This binds the second state factor to the predicate `"PersonIsWaiting"`, which means that $x_2 = 1$ iff the predicate is true.

```
  spin();
```

This spins this node's thread in a loop, during which the State Layer will be running and listening for predicate updates. Currently, the state description cannot be changed after the `spin()` function is called. If the state description contained in a State Layer does not match what is expected by an associated Control Layer, a warning will be thrown.

B.2 Implementing an Observation Layer

An example of the implementation of a simple Observation Layer in ROS will now be analyzed. In this example, there is a single observation factor, defined over events which are associated to a robot's task of patrolling its environment for fires.

```
#include <markov_decision_making/ObservationLayer.h>

using namespace ros;
using namespace markov_decision_making;

int main (int argc, char** argv)
{
  init (argc, argv, "observation_layer_example");

  ObservationLayer ol;
  ol.addObservationFactor (ObservationDep()
                           .add ("Elevator Hallway Clear")
                           .add ("West Corridor Clear")
                           .add ("Coffee Room Clear")
                           .add ("South Corridor Clear")
                           .add ("LRM Clear")
                           .add ("Soccer Field Clear")
                           .add ("Found Fire"));

  spin();

  return 0;
}
```

Breaking down the code:

```
ObservationLayer ol;
```

This declares an Observation Layer which silently subscribes to event topics coming from Predicate Manager (`~/event_updates` and `~/event_map`). The Observation Layer isn't started until the `spin()` function is called. Before that is done, however, the observation space description must be provided.

```
ol.addObservationFactor (ObservationDep()
                         .add ("Elevator Hallway Clear")
                         .add ("West Corridor Clear")
                         .add ("Coffee Room Clear")
                         .add ("South Corridor Clear")
                         .add ("LRM Clear")
                         .add ("Soccer Field Clear")
                         .add ("Found Fire"));
```

This creates our observation factor and associates it to a set of named events (which will be flowing in from `~/event_updates`). As before, indexes start at 0 – the output

of this Observation Layer is 0 when the event `"Elevator Hallway Clear"` is received.

While spinning, the resulting observation is published to the `~/observation` topic, whenever one of the associated events is caught. The Observation Layer also listens in the `~/observation_metadata` topic for incoming observation space descriptions from associated Control Layers, for validation purposes.

B.3 Implementing a Control Layer

The following examples show how generic Control Layers for MDPs and POMDPs can be implemented. First, for an asynchronous (event-driven) MDP:

```
#include <string.h>

#include <ros/ros.h>

#include <markov_decision_making/ControllerEventMDP.h>

using namespace std;
using namespace ros;
using namespace markov_decision_making;

int main (int argc, char** argv)
{
  init (argc, argv, "mdp_control_layer_example");

  if (argc < 4) {
    ROS_ERROR_STREAM ("Usage: mdp_control_layer_example"
                      << "<number of states>"
                      << "<number of actions>"
                      << "<path to MDP Q-table>");
    abort();
  }

  size_t nr_states = atoi(argv[1]);
  size_t nr_actions = atoi(argv[2]);
  string q_table_file = argv[3];

  ControllerEventMDP controller (nr_states,
                                 nr_actions,
                                 q_table_file);

  spin();
```

```
  return 0;
}
```

The `ControllerEventMDP` class implements an asynchronous controller for an MDP agent. Note that it requires, as an input, the $Q-$value function associated with the desired policy. The MDP stochastic models are not required, only its domain sizes (number of states and actions). However, if the MDP itself is defined in a MADP-compatible file format, it can be passed as an input instead (see the MDM API for alternate constructors). In that case, the model will be parsed by MADP, and the following options can be set as parameters in the node's private namespace:

- `is_sparse` (boolean): when set to `true`, the internal representation of the transition and observation functions of the model uses sparse (`boost::uBLAS`) matrices. Typically, for large models, this results in faster parsing and less memory usage at run-time.

- `cache_flat_models` (boolean): when set to `true`, even if the model is defined in a factored format (for example, if the model is written in the *ProbModelXML* format), the "flat" (non-factored) version of the transition and observation functions will be calculated and stored in memory.

MDP controllers will subscribe to the `~/state` topic and publish the associated action to the `~/action` topic. Additionally, if the MDP model is provided, the controller will publish the immediate reward after an action selection to the `~/reward` topic.

In contrast, the following implementation describes an asynchronous POMDP controller:

```
#include <string.h>

#include <ros/ros.h>

#include <markov_decision_making/ControllerEventPOMDP.h>



using namespace std;
using namespace ros;
using namespace markov_decision_making;
```

```
int main (int argc, char** argv)
{
  init (argc, argv, "pomdp_control_layer_example");

  if (argc < 3) {
    ROS_ERROR_STREAM ("Usage: pomdp_control_layer_example"
                      << "<path to problem file>"
                      << "<path to POMDP value function>");
    abort();
  }

  string problem_file = argv[1];
  string value_function_file = argv[2];

  ControllerEventPOMDP controller (problem_file,
                                   value_function_file);

  spin();

  return 0;
}
```

Note that, for POMDP controllers operating according to the scheme shown in Figure 7.8, the problem file must be passed to the constructor, so that it is able to handle belief updates at run-time. POMDP controllers receive observations through the `~/observation` topic. Additionally, they subscribe to `/~initial_state_distribution`, which can be used to set the initial belief of the POMDP. As outputs, POMDP controllers publish actions on the `/~action`; the belief state at run-time to `/~current_belief`; and the immediate (expected) reward to `~/reward`.

B.4 Implementing an Action Layer

The following example shows how a simple MDM Action Layer can be implemented. The present Action Layer interprets the high-level actions *Patrol*, *Assistance Response*, *Trespassing Response* and *Emergency Response*. The former of these is itself a POMDP (this is an example of hierarchical control), while the remaining actions are carried out by finite-state controllers defined using the SMACH package.

```
#include <boost/bind.hpp>

#include <ros/ros.h>
#include <std_srvs/Empty.h>

#include <markov_decision_making/ActionLayer.h>

using namespace ros;
using namespace markov_decision_making;

class Actions
{
public:
  Actions () :
    patrol_stop_client_
    (nh_.serviceClient<std_srvs::Empty>("patrol_POMDP/stop")),
    patrol_reset_client_
    (nh_.serviceClient<std_srvs::Empty>("patrol_POMDP/reset")),
    assistance_SMACH_client_
    (nh_.serviceClient<std_srvs::Empty>("assistance_SMACH/act")),
    trespassing_SMACH_client_
    (nh_.serviceClient<std_srvs::Empty>("trespassing_SMACH/act")),
    emergency_SMACH_client_
    (nh_.serviceClient<std_srvs::Empty>("emergency_SMACH/act")){}

  void patrolPOMDP() {
    std_srvs::Empty e;
    patrol_reset_client_.call (e);
  }

  void assistanceSMACH() {
    std_srvs::Empty e;
    patrol_stop_client_.call (e);
    assistance_SMACH_client_.call (e);
  }

  void trespassingSMACH() {
    std_srvs::Empty e;
    patrol_stop_client_.call (e);
    trespassing_SMACH_client_.call (e);
  }

  void emergencySMACH() {
    std_srvs::Empty e;
```

```
    patrol_stop_client_.call (e);
    emergency_SMACH_client_.call (e);
  }
private:
  NodeHandle nh_;
  ServiceClient patrol_stop_client_;
  ServiceClient patrol_reset_client_;
  ServiceClient assistance_SMACH_client_;
  ServiceClient trespassing_SMACH_client_;
  ServiceClient emergency_SMACH_client_;
};

int main (int argc, char** argv)
{
  init (argc, argv, "action_layer_example");

  Actions am;
  ActionLayer al;
  //Patrol Action
  al.addAction (boost::bind (&Actions::patrolPOMDP, &am));
  //Assistance Response Action
  al.addAction (boost::bind (&Actions::assistanceSMACH, &am));
  //Trespassing Response Action
  al.addAction (boost::bind (&Actions::trespassingSMACH, &am));
  //Emergency Response Action
  al.addAction (boost::bind (&Actions::emergencySMACH, &am));

  spin();

  return 0;
}
```

The most important aspect of this example is how actions are bound to functions in the action layer:

```
Actions am;
ActionLayer al;
//Patrol Action
al.addAction (boost::bind (&Actions::patrolPOMDP, &am));
//Assistance Response Action
al.addAction (boost::bind (&Actions::assistanceSMACH, &am));
//Trespassing Response Action
al.addAction (boost::bind (&Actions::trespassingSMACH, &am));
//Emergency Response Action
al.addAction (boost::bind (&Actions::emergencySMACH, &am));
```

This will create an Action Layer with four associated actions, where each of them is implemented by a method in the auxiliary `Actions` class. Note that the latter class is not a part of MDM – it is simply a design choice, a generic way of containing all action implementations in the same object. The `addAction(.)` function accepts `boost::function` pointers (to functions with no arguments / return values), so using `bind` on the methods of our auxiliary class directly returns the desired type. Those methods will be immediately called when their respective action is received through the `~/action` topic. For example, receiving action 0 will trigger the `Actions::patrolPOMDP()` method.

In this example, all action-bound functions hand over the control of the agent to other modules through ROS service calls. This is also a design choice – using services lets the Action Layer know when and if its client modules receive a request for execution, so it allows for a more secure control of the program. The execution on the client side is outside of the scope of the Action Layer (and of this example). Each SMACH finite-state controller is triggered by calling an `<>\act` service in its respective namespace, which can advertised by a SMACH *Service State*[1]. The execution of a lower-level MDP/POMDP can also be controlled via service requests: Control Layers automatically advertise services to *stop*,*start* or *reset* their execution (the latter essentially stops and starts the controller from its initial conditions). In this particular implementation, the lower-level POMDP (`patrol_POMDP`) is controlled by calling its `stop`/`reset` services.

The downside to the service-based approach is that a client which should *not* be running in a given context may also need an explicit request to stop its execution. This is true, in particular, for the lower-level MDPs/POMDPs.

[1] see: http://www.ros.org/wiki/smach/Tutorials/ServiceState

B.5 Software Location and Documentation

MDM is kept up-to-date at `http://users.isr.ist.utl.pt/~jmessias/MDM/`, where installation instructions can also be found.

With the MDM package installed, the `rosdoc` tool will generate a local copy of its documentation. The MDM `C++` API can then be consulted, and its respective documentation includes further detail on the practical implementation of each MDM module.

References

J. Ahn and J. Hornberger. Involving patients in the cadaveric kidney transplant allocation process: A decision-theoretic perspective. *Management Science*, pages 629–641, 1996.

S. Ahn and V. Ramaswami. Bilateral phase type distributions. *Stochastic models*, 21 (2-3):239–259, 2005.

M. Arias, F. Díez, M. Palacios-Alonso, M. Yebra, and J. Fernández. POMDPs in OpenMarkov and ProbModelXML. In *Seventh Annual Workshop on Multiagent Sequential Decision Making Under Uncertainty (MSDM-2012)*, page 1, 2012.

H. Asama. *Distributed Autonomous Robotic Systems 8*. Springer, 2009.

Y. Aviv and A. Pazgal. A partially observed Markov decision process for dynamic pricing. *Management Science*, pages 1400–1416, 2005.

M. Barbosa, A. Bernardino, D. Figueira, J. Gaspar, N. Gonçalves, P. U. Lima, P. Moreno, A. Pahliani, J. Santos-Victor, M. T. J. Spaan, and J. Sequeira. ISRobotNet: A testbed for sensor and robot network systems. In *Proc. of International Conference on Intelligent Robots and Systems*, pages 2827–2833. IEEE, 2009.

N. Bäuerle and U. Rieder. *Markov Decision Processes with applications to finance*. Springer Verlag, 2011.

R. Becker, S. Zilberstein, and C. Goldman. Solving transition independent decentralized Markov decision processes. *Journal of Artificial Intelligence Research*, 22:423—-455, 2004.

F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing multi-agent systems with JADE*, volume 7. Wiley. com, 2007.

R. Bellman. A Markovian decision process. Technical report, DTIC Document, 1957a.

R. Bellman. *Dynamic Programming*. Princeton University Press, 1957b.

D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, 2002.

D. P. Bertsekas and S. E. Shreve. *Stochastic optimal control: The discrete time case*, volume 139. Academic Press New York, 1978.

C. Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, pages 195–210. Morgan Kaufmann Publishers Inc., 1996.

C. Boutilier and D. Poole. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1168–1175. Citeseer, 1996.

C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian networks. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, pages 115–123. San Francisco, California, 1996.

M. Bowling and M. Veloso. Simultaneous adversarial multi-robot learning. In *Proc. Int. Joint Conf. on Artificial Intelligence*, volume 3, pages 699–704, 2003.

X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proc. of Uncertainty in Artificial Intelligence*, 1998.

S. J. Bradtke and M. O. Duff. Reinforcement learning methods for continuous-time Markov decision problems. *Advances in neural information processing systems*, 7: 393–400, 1995.

E. Brunskill, L. Kaelbling, T. Lozano-Pérez, and N. Roy. Planning in partially-observable switching-mode continuous domains. *Annals of Mathematics and Artificial Intelligence*, 58(3):185–216, 2010.

M. Cabasino, A. Giua, and C. Seatzu. Diagnosis using labeled Petri Nets with silent or undistinguishable fault events. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 43(2):345–355, 2013.

J. Capitán, M. T. J. Spaan, L. Merino, and A. Ollero. Decentralized multi-robot cooperation with auctioned POMDPs. *International Journal of Robotics Research*, 32(6): 650–671, 2013.

A. R. Cassandra. *Exact and approximate algorithms for partially observable Markov decision processes*. PhD thesis, Brown University, Providence, RI, USA, 1998a.

A. R. Cassandra. A survey of POMDP applications. In *Working Notes of AAAI 1998 Fall Symposium on Planning with Partially Observable Markov Decision Processes*, pages 17–24, 1998b.

C. Cassandras and S. Lafortune. *Introduction to discrete event systems*. Kluwer academic publishers, 1999.

H. Chao and A. Manne. Oil stockpiles and import reductions: A dynamic programming approach. *Operations research*, pages 632–651, 1983.

H. Cheng. *Algorithms for partially observable Markov decision processes*. PhD thesis, University of British Columbia, 1988.

H. Costelha and P. Lima. Modelling, analysis and execution of robotic tasks using Petri Nets. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1449–1454. IEEE, 2007.

B. Damas and P. Lima. Stochastic discrete event model of a multi-robot team playing an adversarial game. In *5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles-IAV2004*, 2004.

J. de Cani. A dynamic programming algorithm for embedded Markov chains when the planning horizon is at infinity. *Management Science*, pages 716–733, 1964.

C. Derman, G. Lieberman, and S. Ross. A stochastic sequential allocation model. *Operations Research*, pages 1120–1130, 1975.

REFERENCES

M. B. Dias, R. Zlot, N. Kalra, and A. Stentz. Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94(7):1257–1270, 2006.

S. Dreyfus. A note on an industrial replacement process. *OR*, 8(4):190–193, 1957.

R. Duda, P. Hart, and D. Stork. *Pattern classification*. John Wiley & Sons, 2001.

J. Duncan and L. Scholnick. Interrupt and opportunistic replacement strategies for systems of deteriorating components. *Operational Research Quarterly*, pages 271–283, 1973.

F. Díez, M. Palacios, and M. Arias. MDPs in medicine: Opportunities and challenges. In *Decision Making in Partially Observable, Uncertain Worlds: Exploring Insights from Multiple Communities*, 2011. Workshop at IJCAI11.

J. Eckles. Optimum maintenance with incomplete information. *Operations Research*, pages 1058–1067, 1968.

H. Ellis, M. Jiang, and R. Corotis. Inspection, maintenance, and repair with partial observability. *Journal of Infrastructure Systems*, 1(2):92–99, 1995.

R. Emery-Montemerlo, G. Gordon, J. Schneider, and S. Thrun. Game theoretic control for robot teams. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1163–1169. IEEE, 2005.

T. Fabian, J. Fisher, M. Sasieni, and A. Yardeni. Purchasing raw material on a fluctuating market. *Operations Research*, pages 107–122, 1959.

X. G. Fang and G. Havas. On the worst-case complexity of integer gaussian elimination. In *Proceedings of the 1997 international symposium on Symbolic and algebraic computation*, pages 28–31. ACM, 1997.

E. Feinberg and A. Shwartz. *Handbook of Markov decision processes: methods and applications*, volume 40. Springer Netherlands, 2002.

F. Fernández and L. E. Parker. Learning in large cooperative multi-robot domains. *International Journal of Robotics and Automation (Special issue on Computational Intelligence Techniques in Cooperative Robots)*, 16(4):217–226, 2001.

S. Fleten and T. Kristoffersen. Short-term hydropower production planning by stochastic programming. *Computers & Operations Research*, 35(8):2656–2671, 2008.

V. K. Garg. An algebraic approach to modeling probabilistic discrete event systems. In *Proceedings of the 31st IEEE Conference on Decision and Control*, pages 2348–2353. IEEE, 1992.

B. P. Gerkey and M. J. Mataric. Multi-robot task allocation: Analyzing the complexity and optimality of key architectures. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 3, pages 3862–3868. IEEE, 2003.

P. W. Glynn. A GSMP formalism for discrete event systems. *Proceedings of the IEEE*, 77(1):14–23, 1989.

P. Gmytrasiewicz and P. Doshi. A framework for sequential planning in multiagent settings. *Journal of Artificial Intelligence Research*, 24(1):49–79, 2005.

C. V. Goldman and S. Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research*, 22: 143–174, 2004.

R. Grupen and J. Coelho. Acquiring state from control dynamics to learn grasping policies for robot hands. *Advanced Robotics*, 16(5):427–443, 2002.

X. Guo and O. Hernández-Lerma. *Continuous-time Markov decision processes: theory and applications*, volume 62. Springer, 2009.

E. Hansen and Z. Feng. Dynamic programming for POMDPs using a factored state representation. In *Proceedings of the Fifth International Conference on AI Planning Systems*, pages 130–139, 2000.

E. Hansen, D. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *Proceedings of the National Conference on Artificial Intelligence*, pages 709–715. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2004.

M. Hauskrecht and H. Fraser. Planning treatment of ischemic heart disease with partially observable Markov decision processes. *Artificial Intelligence in Medicine*, 18(3): 221–244, 2000.

D. Herrero-Perez and H. Martinez-Barbera. Petri Nets based coordination of flexible autonomous guided vehicles in flexible manufacturing systems. In *IEEE International Conference on Emerging Technologies and Factory Automation*, pages 508–515. IEEE, 2008.

J. Hoey and P. Poupart. Solving POMDPs with continuous or large discrete observation spaces. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2005.

J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Proc. of Uncertainty in Artificial Intelligence*, 1999.

R. Howard. Semi-Markovian decision processes. *Bulletin de l'Institut International de Statistique*, 40(2):625–652, 1963.

R. A. Howard. *Dynamic programming and Markov processes*. New York: John Wiley & Sons, Inc, 1960.

K. Hsiao, L. Kaelbling, and T. Lozano-Perez. Grasping POMDPs. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4685–4692. IEEE, 2007.

D. Hsu, W. Lee, and N. Rong. A point-based POMDP planner for target tracking. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2644–2650. IEEE, 2008.

E. Ignall and P. Kolesar. Optimal dispatching of an infinite-capacity shuttle: control at a single terminal. *Operations Research*, pages 1008–1024, 1974.

T. Jaakkola, S. P. Singh, and M. I. Jordan. Reinforcement learning algorithm for partially observable Markov decision problems. In *Advances in Neural Information Processing Systems 7*, pages 345–352. MIT Press, 1995.

A. Jazwinski. *Stochastic processes and filtering theory*, volume 64. Academic Press, Inc., 1970.

W. Jewell. Markov-renewal programming. ii: Infinite return models, example. *Operations Research*, pages 949–971, 1963.

L. Kaelbling, M. Littman, and A. Moore. Reinforcement learning: A survey. *Arxiv preprint cs/9605103*, 1996.

L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

R. Kaplan. Optimal investigation strategies with imperfect information. *Journal of Accounting Research*, pages 32–43, 1969.

J. Kober, B. Mohler, and J. Peters. Learning perceptual coupling for motor primitives. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 834–839. IEEE, 2008.

S. Koenig and R. Simmons. Xavier: A robot navigation architecture based on partially observable Markov decision process models. *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, pages 91–122, 1998.

A. Kristensen. A survey of Markov decision programming techniques applied to the animal replacement problem. *European Review of Agricultural Economics*, 21(1):73, 1994.

A. Kumar and S. Zilberstein. Anytime planning for decentralized POMDPs using expectation maximization. In *Uncertainty in Artificial Intelligence*, pages 294–301, 2010.

H. Kurniawati, D. Hsu, and W. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems*, pages 65–72, 2008.

D. Lane. A partially observable model of decision making by fishermen. *Operations Research*, pages 240–254, 1989.

S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society - Series B (Methodological)*, pages 157–224, 1988.

M. Lawford and W. M. Wonham. Supervisory control of probabilistic discrete event systems. In *Proceedings of the 36th Midwest Symposium on Circuits and Systems*, pages 327–331. IEEE, 1993.

T. Lemaire, R. Alami, and S. Lacroix. A distributed tasks allocation scheme in multi-UAV context. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 4, pages 3622–3627. IEEE, 2004.

Z. Lim, D. Hsu, and W. Lee. Monte Carlo Value Iteration with macro-actions. *Journal of Web Semantics*, 2:2, 2004.

P. U. Lima and L. M. Custodio. Multi-robot systems. In *Innovations in robot mobility and control*, pages 1–64. Springer, 2005.

J. Lindqvist. Operation of a hydrothermal electric system: A multistage decision process. *Power Apparatus and Systems, Part III. Transactions of the American Institute of Electrical Engineers*, 81(3):1–6, 1962.

J. Little. The use of storage water in a hydroelectric system. *Journal of the Operations Research Society of America*, pages 187–197, 1955.

D. Low. Optimal dynamic pricing policies for an M/M/s queue. *Operations Research*, pages 545–561, 1974.

S. Mahadevan. Partially observable semi-Markov decision processes: Theory and applications in engineering and cognitive science. In *AAAI: Fall Symposium on Planning with Partially Observable Markov Decision Processes*, 1998.

S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial intelligence*, 55(2-3):311–365, 1992.

S. Mahadevan and N. Khaleeli. Robust mobile robot navigation using partially-observable semi-Markov decision processes. Technical report, 1999.

R. T. Maheswaran, M. Tambe, E. Bowring, J. P. Pearce, and P. Varakantham. Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In *Proc. of Int. Conference on Autonomous Agents and Multi Agent Systems*, volume 1, pages 310–317. IEEE Computer Society, 2004.

M. Mataric. Reward functions for accelerated learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, volume 189. Morgan Kauffman, New Brunswick, NJ, 1994.

M. Matarić. Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4 (1):73–83, 1997.

D. A. McAllester and S. Singh. Approximate planning for factored POMDPs using belief state simplification. In *Proc. of Uncertainty in Artificial Intelligence*, 1999.

J. Messias, M. T. J. Spaan, and P. U. Lima. Efficient offline communication policies for factored multiagent POMDPs. In *Proceedings of the 25th Annual Conference on Neural Information Processing Systems (NIPS '11)*, pages 1917–1925, 2011.

J. Messias, M. T. J. Spaan, and P. U. Lima. Gsmdps for multi-robot sequential decision-making. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence*, 2013a.

J. Messias, M. T. J. Spaan, and P. U. Lima. Multiagent pomdps with asynchronous execution. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-13) - Extended Abstract*, 2013b.

J. Messias, M. T. J. Spaan, and P. U. Lima. Asynchronous Execution in Multiagent POMDPs: Reasoning over Partially-Observable Events. *Artificial Intelligence (Special Issue on AI and Robotics)* ***Submitted. Pending Review.***, 2013c.

S. Miller, Z. Harris, and E. Chong. A POMDP framework for coordinated guidance of autonomous UAVs for multitarget tracking. *EURASIP Journal on Advances in Signal Processing*, 2009:2, 2009.

W. T. Miller, F. H. Glanz, and L. G. Kraft. CMAC: An associative neural network alternative to backpropagation. *Proceedings of the IEEE*, 78(10):1561–1567, 1990.

G. Monahan. A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Management Science*, pages 1–16, 1982.

P. Moreno, A. Bernardino, and J. Santos-Victor. Waving detection using the local temporal consistency of flow-based features for real-time applications. In *Image Analysis and Recognition*, pages 886–895. Springer, 2009.

T. Murata. Petri Nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

K. Murphy and Y. Weiss. The factored frontier algorithm for approximate inference in DBNs. In *Proc. of Uncertainty in Artificial Intelligence*, pages 378–385. Morgan Kaufmann Publishers Inc., 2001.

K. P. Murphy. *Dynamic Bayesian networks: representation, inference and learning*. PhD thesis, University of California, 2002.

R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *Proc. of the National Conference on Artificial Intelligence*, pages 133–139, 2005.

G. Neto. *Planning, learning and control under uncertainty based on discrete event systems and reinforcement learning*. PhD thesis, Instituto Superior Técnico, Universidade Ténica de Lisboa, 2010.

G. Neto, H. Costelha, and P. Lima. Topological navigation in configuration space applied to soccer robots. In *RoboCup 2003: Robot Soccer World Cup VII*, pages 551–558. Springer, 2004.

A. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang. Autonomous inverted helicopter flight via reinforcement learning. *Experimental Robotics IX*, pages 363–372, 2006.

B. Ng, C. Meyers, K. Boakye, and J. Nitao. Towards applying interactive POMDPs to real-world adversary modeling. In *Twenty-Second IAAI Conference*, 2010.

I. Nourbakhsh, R. Powers, and S. Birchfield. Dervish: an office-navigating robot. *AI magazine*, 16(2):53, 1995.

M. Ohnishi, H. Kawai, and H. Mine. An optimal inspection and replacement policy under incomplete state information. *European journal of operational research*, 27(1): 117–128, 1986.

F. Oliehoek and A. Visser. A hierarchical model for decentralized fighting of large scale urban fires. In *AAMAS'06 Workshop on Hierarchical Autonomous Agents and Multi-Agent Systems*, pages 14–21, 2006.

F. A. Oliehoek. Factored Dec-POMDPs: exploiting locality of interaction. In *Value-based planning for teams of agents in stochastic partially observable environments*, chapter 2. PhD thesis Frans A. Oliehoek, University of Amsterdam, 2010.

F. A. Oliehoek, M. T. J. Spaan, and N. Vlassis. Dec-POMDPs with delayed communication. In *Multi-agent Sequential Decision Making in Uncertain Domains*, 2007. Workshop at AAMAS07.

F. A. Oliehoek, M. T. J. Spaan, and N. Vlassis. Optimal and approximate q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research*, 32 (1):289–353, 2008a.

F. A. Oliehoek, M. T. J. Spaan, S. Whiteson, and N. Vlassis. Exploiting locality of interaction in factored Dec-POMDPs. In *Proc. of Int. Conference on Autonomous Agents and Multi Agent Systems*, 2008b.

F. A. Oliehoek, S. Whiteson, and M. T. J. Spaan. Approximate solutions for factored Dec-POMDPs with many agents. In *Proc. of Int. Conference on Autonomous Agents and Multi Agent Systems*, pages 563–570. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

D. Onstad and R. Rabbinge. Dynamic programming and the computation of economic injury levels for crop disease control. *Agricultural Systems*, 18(4):207–226, 1985.

T. Osogami and M. Harchol-Balter. Closed form solutions for mapping general distributions to quasi-minimal PH distributions. *Performance Evaluation*, 63(6):524–552, 2006.

J. Pajarinen and J. Peltonen. Efficient planning for factored infinite-horizon Dec-POMDPs. In *Proc. of the International Joint Conference on Artificial Intelligence,(to appear)*, 2011.

C. Papadimitriou and J. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of operations research*, pages 441–450, 1987.

S. Paquet, L. Tobin, and B. Chaib-Draa. An online POMDP algorithm for complex multiagent environments. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 970–977. ACM, 2005.

R. Parr. *Hierarchical control and learning for Markov decision processes*. PhD thesis, Citeseer, 1998.

R. Patrascu, P. Poupart, D. Schuurmans, C. Boutilier, and C. Guestrin. Greedy linear value-approximation for factored Markov decision processes. In *Eighteenth national conference on Artificial intelligence*, pages 285–291, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence. ISBN 0-262-51129-0.

J. Pavón and J. Gómez-Sanz. Agent oriented software engineering with INGENIAS. In *Multi-Agent Systems and Applications III*, pages 394–403. Springer, 2003.

J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

J. Pineau and S. Thrun. An integrated approach to hierarchy and abstraction for POMDPs. Technical Report CMU-RI-TR-02-21, Robotics Institute, Carnegie Mellon University, 2002.

J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *International Joint Conference on Artificial Intelligence*, volume 18, pages 1025–1032. Citeseer, 2003.

J. Porta, N. Vlassis, M. T. J. Spaan, and P. Poupart. Point-based value iteration for continuous POMDPs. *The Journal of Machine Learning Research*, 7:2329–2367, 2006.

P. Poupart. *Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes*. PhD thesis, University of Toronto, 2005.

P. Poupart and C. Boutilier. Value-directed belief state approximation for POMDPs. In *Proc. of Uncertainty in Artificial Intelligence*, volume 130, 2000.

G. Pritchard, A. Philpott, and P. Neame. Hydroelectric reservoir optimization in a pool market. *Mathematical programming*, 103(3):445–461, 2005.

M. L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, Inc., 1994.

D. V. Pynadath and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, 16:389–423, 2002.

M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, 2009.

M. M. Quottrup, T. Bak, and R. Zamanabadi. Multi-robot planning: A timed automata approach. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 5, pages 4417–4422. IEEE, 2004.

E. Rachelson, G. Quesnel, F. Garcia, and P. Fabiani. Approximate policy iteration for generalized semi-Markov decision processes: an improved algorithm. In *European Workshop on Reinforcement Learning*, 2008.

J. C. G. Reis, P. U. Lima, and J. Garcia. Efficient distributed communications for multi-robot systems. In *Proceedings of the 17th RoboCup International Symposium*, 2013.

M. Riedmiller, T. Gabel, R. Hafner, and S. Lange. Reinforcement learning for robot soccer. *Autonomous Robots*, 27(1):55–73, 2009.

H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.

P. Rong and M. Pedram. Extending the lifetime of a network of battery-powered mobile devices by remote processing: a Markovian decision-based approach. In *Proceedings of the 40th annual Design Automation Conference*, pages 906–911. ACM, 2003.

J. Rosell, N. Munoz, and A. Gambin. Robot tasks sequence planning using Petri Nets. In *Proceedings of the IEEE International Symposium on Assembly and Task Planning*, pages 24–29. IEEE, 2003.

M. Roth, R. Simmons, and M. Veloso. Decentralized communication strategies for coordinated multi-agent policies. In *Multi-Robot Systems: From Swarms to Intelligent Automata*, volume IV. Kluwer Academic Publishers, 2005a.

M. Roth, R. Simmons, and M. Veloso. Reasoning about joint beliefs for execution-time communication decisions. In *Proc. of Int. Conference on Autonomous Agents and Multi Agent Systems*, pages 786–793. ACM, 2005b.

M. Roth, R. Simmons, and M. Veloso. Exploiting factored representations for decentralized execution in multi-agent teams. In *Proc. of Int. Conference on Autonomous Agents and Multi Agent Systems*, 2007.

A. Schaefer, M. Bailey, S. Shechter, and M. Roberts. Modeling medical treatment using Markov decision processes. *Operations Research and Health Care*, pages 593–612, 2005.

S. Seuken and S. Zilberstein. Memory-bounded dynamic programming for Dec-POMDPs. In *International Joint Conference on Artificial Intelligence*, pages 2009–2016, 2007.

S. Shechter, M. Bailey, A. Schaefer, and M. Roberts. The optimal time to initiate hiv therapy under ordered health states. *Operations Research*, 56(1):20, 2008.

Y. Shoham and K. Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2009.

R. Simmons and S. Koenig. Probabilistic robot navigation in partially observable environments. In *International Joint Conference on Artificial Intelligence*, volume 14, pages 1080–1087, 1995.

E. Sondik. *The optimal control of partially observable Markov processes*. PhD thesis, Stanford, 1971.

F. Sonnenberg and J. Beck. Markov models in medical decision making. *Medical decision making*, 13(4):322, 1993.

M. Spaan and F. Groen. Team coordination among robotic soccer players. In G. Kaminka, P. Lima, and R. Rojas, editors, *RoboCup 2002: Robot Soccer World Cup VI*, volume 2752 of *Lecture Notes in Computer Science*, pages 409–416. Springer Berlin / Heidelberg, 2003.

M. T. J. Spaan and F. S. Melo. Interaction-driven Markov games for decentralized multiagent planning under uncertainty. In *Proc. of Int. Conference on Autonomous Agents and Multi Agent Systems*, 2008.

M. T. J. Spaan and F. A. Oliehoek. The MultiAgent Decision Process toolbox: Software for decision-theoretic planning in multiagent-systems. In *AAMAS'08 Workshop on Multiagent Sequential Decision Making in Uncertain Domains (MSDM-2008)*. IFAAMAS, 2008.

M. T. J. Spaan and N. Vlassis. A point-based POMDP algorithm for robot planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 3, pages 2399–2404. IEEE, 2004.

M. T. J. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24(1):195–220, 2005.

M. T. J. Spaan, F. A. Oliehoek, and N. Vlassis. Multiagent planning under uncertainty with stochastic communication delays. In *Proc. of Int. Conf. on Automated Planning and Scheduling*, pages 338–345, 2008.

M. T. J. Spaan, F. A. Oliehoek, and C. Amato. Scaling up optimal heuristic search in Dec-POMDPs via incremental expansion. In *Proc. of International Joint Conference on Artificial Intelligence*, pages 2027–2032, 2011.

M. Sridharan, J. Wyatt, and R. Dearden. Planning to see: A hierarchical approach to planning visual actions on a robot using POMDPs. *Artificial Intelligence*, 174: 704–725, 2010.

S. Stidham and R. Weber. A survey of Markov decision models for control of networks of queues. *Queueing Systems*, 13(1):291–314, 1993.

R. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163, 1991.

R. S. Sutton. *Temporal credit assignment in reinforcement learning*. PhD thesis, University of Massachusetts Amherst, 1984. AAI8410337.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 28. Cambridge Univ Press, 1998.

R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1): 181–211, 1999.

D. Szer and F. Charpillet. An optimal best-first search algorithm for solving infinite horizon Dec-POMDPs. *Machine Learning: ECML 2005*, pages 389–399, 2005.

N. Tao, J. Baxter, and L. Weaver. A multi-agent, policy-gradient approach to network routing. In *In: Proc. of the 18th Int. Conf. on Machine Learning*, pages 553–560. Morgan Kaufmann, 2001.

G. Tesauro. Practical issues in temporal difference learning. *Machine learning*, 8(3): 257–277, 1992.

G. Theocharous and S. Mahadevan. Approximate planning with hierarchical partially observable Markov decision processes for robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2002.

G. Theocharous, S. Mahadevan, and L. P. Kaelbling. Spatial and temporal abstractions in POMDPs applied to robot navigation. Technical Report MIT-CSAIL-TR-2005-058, Computer Science and Artificial Inteligence Laboratory, MIT, 2005.

S. Thrun. Learning to play the game of chess. *Advances in Neural Information Processing Systems*, pages 1069–1076, 1995.

S. Thrun. Monte carlo POMDPs. *Advances in neural information processing systems*, 12:1064–1070, 2000.

J. N. Tsitsiklis and B. Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22(1):59–94, 1996.

W. T. B. Uther and M. M. Veloso. Tree based discretization for continuous state space reinforcement learning. In *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 769–774, 1998.

N. Vlassis, M. Toussaint, G. Kontes, and S. Piperidis. Learning model-free robot control by a Monte Carlo EM algorithm. *Autonomous Robots*, 27(2):123–130, 2009.

D. Wang and B. Adams. Optimization of real-time reservoir operations with Markov decision processes. *Water Resources Research*, 22(3):345–352, 1986.

D. Wang and X. Cao. Event-based optimization for POMDPs and its application in portfolio management. In *Proceedings of the 18th IFAC World Congress*, pages 3228–3233, 2011.

C. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

C. White, E. Wilson, and A. Weaver. Decision aid development for use in ambulatory health care settings. *Operations Research*, pages 446–463, 1982.

C. C. White. Procedures for the solution of a finite-horizon, partially observed, semi-Markov optimization problem. *Operations Research*, pages 348–358, 1976.

C. C. White. Partially observed Markov decision processes: a survey. *Annals of Operations Research*, 32, 1991.

D. White. Real applications of Markov decision processes. *Interfaces*, pages 73–83, 1985.

D. White. Further real applications of Markov decision processes. *Interfaces*, pages 55–61, 1988.

D. White. A survey of applications of Markov decision processes. *Journal of the Operational Research Society*, 44(11):1073–1096, 1993.

D. White and J. Norman. Control of cash reserves. *OR*, pages 309–328, 1965.

J. Williams and S. Young. Partially observable Markov decision processes for spoken dialog systems. *Computer Speech & Language*, 21(2):393–422, 2007.

I. Witten. An adaptive optimal controller for discrete-time Markov environments. *Information and Control*, 34(4):286–295, 1977.

S. Witwicki. *Abstracting Influences for Efficient Multiagent Coordination Under Uncertainty*. PhD thesis, University of Michigan, 2012.

S. Witwicki, F. S. Melo, J. Capitán, and M. T. J. Spaan. A flexible approach to modeling unpredictable events in MDPs. In *Int. Conf. on Automated Planning and Scheduling*, 2013.

F. Wu and X. Chen. Solving large-scale and sparse-reward Dec-POMDPs with correlation-MDPs. In U. Visser, F. Ribeiro, T. Ohashi, and F. Dellaert, editors, *RoboCup 2007: Robot Soccer World Cup XI*, volume 5001 of *Lecture Notes in Computer Science*, pages 208–219. Springer Berlin / Heidelberg, 2008.

F. Wu, S. Zilberstein, and X. Chen. Multi-agent online planning with communication. In *Int. Conf. on Automated Planning and Scheduling*, 2009.

T. Yamasaki and T. Ushio. Decentralized supervisory control of discrete event systems based on reinforcement learning. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 88(11):3045–3050, 2005.

H. Younes. Planning and execution with phase transitions. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1030–1035, 2005.

H. L. S. Younes and R. G. Simmons. Solving generalized semi-Markov decision processes using continuous phase-type distributions. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 742–747. San Jose, California. AAAI Press., 2004.

S. Young, M. Gasic, S. Keizer, F. Mairesse, J. Schatzmann, B. Thomson, and K. Yu. The hidden information state model: A practical framework for POMDP-based spoken dialogue management. *Computer Speech & Language*, 24(2):150–174, 2010.

E. Zhou, M. Fu, and S. Marcus. A density projection approach to dimension reduction for continuous-state POMDPs. In *47th IEEE Conference on Decision and Control*, pages 5576–5581. IEEE, 2008.