INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Cooperative Active Perception using POMDPs

Tiago Santos Veiga

Dissertação para a obtenção do Grau de Mestre em

Engenharia Electrotécnica e de Computadores

Júri:

Presidente:	Professor Carlos Jorge Ferreira Silvestre
Orientador:	Doutor Matthijs Theodor Jan Spaan
Co-Orientador:	Professor Pedro Manuel Urbano Almeida Lima
Vogal:	Professor Mário Alexandre Teles de Figueiredo

Maio de 2010

Agradecimentos

Antes de mais, um agradecimento devido aos meus orientadores, ao Matthijs por me ter apresentado e guiado neste mundo novo da incerteza, no fundo a verdadeira representação da vida. Ao Professor Pedro Lima pelo apoio de fundo que deu, e por ser um grande exemplo para todos aqueles que se iniciam na área da robótica.

Acima de tudo à minha família, por ser um suporte tão grande na minha vida. Aos meus pais por todo o esforço, dedicação e esperança que depositaram na minha educação e por serem exemplos de vida, tudo se deve a eles. À minha irmã, quase a minha segunda mãe, mas sempre pronta a ajudar e a apoiar. Aos meus avós por todo o ensinamento com a sua experiência de vida, e as suas histórias sempre interessantes.

Aos meus amigos e companheiros de vida. Aos amigos de Sesimbra, desde o tempo dos jogos na Torrinha até hoje sempre presentes para animar esta vida. Aos amigos e colegas do Técnico, mostram como pessoas simples podem fazer amizades fortes. E aos meus companheiros de aventura erásmica, por aqueles meses fantásticos e intensos em que se abriram tantos horizontes.

À Marta, simplesmente pelos nossos caminhos se terem cruzado, por muito ter crescido, aprendido e partilhado, pela ajuda essencial, por ser importante.

No fundo, agradeço a todos os que fizeram e fazem parte da minha vida, os que ficaram guardados no coração, por tudo o que aprendi convosco, tudo o que me alegrei, tudo o que sofri, é chegada a hora de fazer jus a todos vós, quer tenham ficado ou partido, nunca vos esquecerei.

Resumo

Nesta tese é apresentada uma nova abordagem a problemas de percepção activa cooperativa em sistemas robóticos em rede. O objectivo é realizar planeamento de tarefas num contexto onde múltiplos sensores cooperam de modo a desenvolver tarefas com sucesso, dadas as limitações nos vários tipos de sensores.

Percepção activa cooperativa é o problema de percepção activa (um agente considera os efeitos das suas acções nos seus sensores) envolvendo múltiplos sensores e múltiplos decisores cooperativos. Aqui, é considerado o caso em que um sensor móvel coopera com uma rede de sensores fixos de modo a reduzir a incerteza no sistema.

É proposta uma abordagem à percepção activa cooperativa baseada na teoria da decisão. Em particular, são propostos Processos de Decisão de Markov Parcialmente Observáveis (POMDP) como a base para lidar com tais problemas. Os POMDPs fornecem uma maneira elegante de modelar problemas de tomada de decisão sobre incerteza, oferecendo um suporte matemático muito forte, modelando explicitamente os erros de detecção e de actuação no sistema. Devido a esta capacidade, os POMDPs mantêm uma crença sobre todos os estados, permitindo o acompanhamento de características.

É mostrado como modelar um problema de percepção activa cooperativa sobre a base de POMDP, e apresentados resultados promissores em cenários reais, em particular o modelo é aplicado num sistema classificador que pondera, de entre vários eventos detectados no ambiente, quais são de interesse e classificá-los, provando que o modelo desenvolvido aplica-se na realidade neste tipo de problemas.

Palavras chave: Sistemas Robóticos em Rede, Processos de Decisão de Markov Parcialmente Observáveis, Planeamento sobre incerteza, Percepção Activa Cooperativa, Sistemas de Classificação.

Abstract

In this thesis we present a new approach to cooperative active perception problems in networked robot systems. Our goal is to perform task planning in a context where multiple sensors of different kinds cooperate in order to successfully develop tasks, given the limitations in every kind of sensors.

Cooperative active perception is the problem of active perception involving multiple sensors and multiple cooperating decision makers. Active perception means that an agent considers the effects of its actions on its sensors. Here, we consider the case in which a mobile sensor cooperates with a network of fixed sensors in order to reduce uncertainty in the system.

We propose a decision-theoretic approach to cooperative active perception. In particular, we propose Partially Observable Markov Decision Processes (POMDP) as the framework for such problems. POMDPs provide an elegant way to model problems of decision-making under uncertainty, offering a strong mathematical background by explicitly modeling the imperfect sensing and actuation capabilities of the overall system. Due to this capability, POMDP keep a track on the belief over states, allowing for feature tracking.

We show how to model a cooperative active perception problem under the POMDP framework, and we present promising results in real scenarios. In particular we apply our model in a classifier system which is able to reason, from several events detected in an environment, which are of interest, and classify them, proving that our model actually applies in those kind of problems.

Keywords: Networked Robot System, Partially Observable Markov Decision Processes, Planning under Uncertainty, Cooperative Active Perception, Classifier System.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Technology research has improved and extended the areas of robotic applications over the last years. Although robots used to be confined to industrial environments or to research labs, nowadays they are leaving into more human-inhabited environments. The development of better artificial intelligence algorithms and the increase in computational power played a very important role in this evolution. Therefore, robots have been tested in new and more complex applications, making it possible to create systems which use mobile robots and artificial intelligence to do tasks autonomously, making its own decisions according to sensing information, in contrast to industrial robots, which were not initially designed to take decisions, but to perform repetitive tasks. Examples of mobile robotic applications which may be developed to help human activity or even replace it are a semi-autonomous robot for rescue operations [2], an autonomous vacuum cleaner [3], an office delivery robot [4] or an interactive museum tour-guide robot [5].

Additional technological developments, such as in communications, with reduction of time-delays, lead to the emergence of a new research field, Network Robot Systems (NRS), in which the robots are no longer seen as stand-alone agents, but rather as an active part of a system which behaves as a whole. In this framework, each element sees the others as a part of themselves. For instance, a robot may establish a connection to a sensor to which is not physically connected, but perceives its data as it was observed by the robot itself. A NRS may also be seen as a connected graph, that is, if we consider each element of the network as a node and equally each connection point as nodes, every element is capable of receiving and/or sending information to another element in the network. Figure 1.1 represents a diagram which shows such property, with a multitude of sensors and robots spread over the environment connected through wired or wireless networks. The focus on such systems is not to act depending on what we think others are doing and observing, but rather to act depending on how others tell they are acting and observing, and that is why communications are an important topic in such subjects. Therefore, it is also important to ensure there is enough communication capacity, such as bandwidth so the information is able to be received at any point in the network. In [6] NRS is defined as any distributed system which consists of a multitude of networked robots and other devices and which, as a whole, is capable of interacting with the environment through the use of perception and action for the performance of tasks. In turn, in [7] the author restricts this definition to interactions with human beings. It divides a NRS in the following elements:

- Physical embodiment: a NRS is considered to have at least one physical robot which incorporates hardware and software;

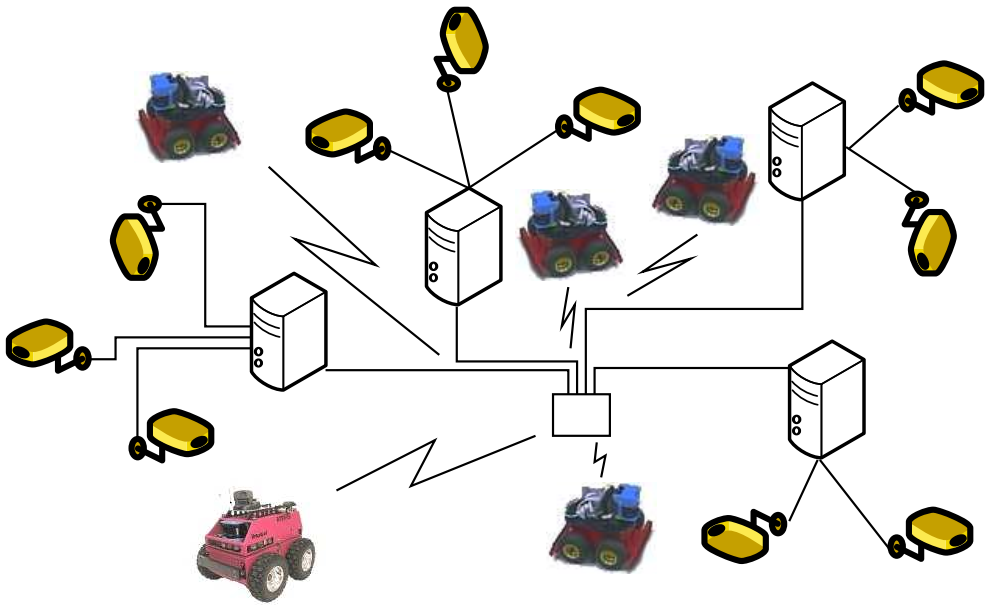


Figure 1.1: Diagram of an NRS, showing how the system distributes over the environment, and all elements of the system interact between each other.

- Autonomous capabilities: robots on the system must have autonomous capabilities;
- Network-based cooperation: all the elements on the system (robots, environment sensors and humans) must communicate and cooperate through a network;
- Environment sensors and actuators: the environment must include a series of sensors and actuators, such as vision cameras, laser range finders or speakers, besides the sensors of the robots;
- Human-robot interaction: the system must have a human-robot related activity.

Initially, technology development on NRS was focused on robotic teleoperation, where the interaction is based on a human supervisor sending commands and receiving feedback through a network to control robots. Nowadays, a different subclass of Networked Robots is the focus of attention: Autonomous NRS. In such systems, robots and sensors exchange data via the network, allowing for the sensor network to extend the effective sensing range of the robots, allowing them to communicate with each other over long distances. NRS implies the integration of several fields: robotics, perception, ubiquitous computing, artificial intelligence and network communications. Some of the hot topics in the design of NRS are cooperative localization and navigation, cooperative environment perception, cooperative map building, task allocation, cooperative task execution, human-robot interaction, network teleoperation and communications.

The main contributions which lead to the increasing interest and development of NRS came from sensor network and cooperative robotics communities. Better information about the environment and better cooperation between all elements on a NRS are essential for better task completion.

With such developments on research, many applications of NRS are being pointed as viable in the next years, including those mention in [6]:

- Elderly care, where a NRS can assist elderly people in their personal homes or retirement homes, with the goal to provide physical and cognitive support, to facilitate communication with and monitoring by remote relatives and care givers, and to detect and respond to emergencies;
- Security applications and intelligent buildings, where a NRS can track and classify the behavior of people,

hazardous situations or threatening acts can be detected, and actions can be decided to maintain a safe status;

- Networked service robot systems, or applications such as trash collection, delivery and logistics, both in public spaces (e.g., city streets) and in private workplaces (e.g., factory floors).
- Flexible automation and collaborative manufacturing, where heterogeneous, cooperative robotic systems are expected to set forth the future of networked control in industrial settings.
- Wide-scale environmental monitoring, by deploying autonomous sensor networks with the ability to self-deploy, self-reconfigure, and self-repair, capable of monitoring in a largely unsupervised way large environments for pollutions, environmental threats, and other hazardous situations.
- Cooperative search and rescue, where NRS actively search for people or objects, and eventually support rescue personnel in dealing with emergencies.

For successful task completion it is very important to take good decisions, specially due to the uncertainty that typically characterizes NRS environments. Planning has been an important research topic in robotics area, specially planning under uncertainty. Those kind of problems brings theory close to reality, since in the real world any agent acts under uncertainty. That is an important issue in robotics, how should a robot act to perform its assigned task as well as possible taking into account that it does not have exact knowledge of the world, but instead it plans in a model of the world, which is always an abstraction of the real world, and therefore does not model all the real effects. Uncertainty, in robotics, has two sources, both uncertainty in motion and in sensing. For instance, when the robot moves it might have to deal with unexpected issues, such as wheel slipping. Another major source of uncertainty is its sensors, which are often noisy and have a limited view of the environment.

The development towards ubiquitous robotics [8] extended this research topic not only for robot behavior but for planning of the system behavior as a whole. Networked robots integrated with networked sensors in a ubiquitous environment to form ubiquitous robots, that is, the concept of robot is extended to every device to which it is connected through the network, each device sees all the others as an extension of themselves, although not physically connected.

As such, planning in NRS covers a number of possible applications and decisions. No longer planning sticks to the problem of driving robots, finding the shortest path or autonomous localization, among others, but rather research has opened to more applications and possible behaviors of the system. It may range from dispatching robots to cover detected events (e.g., a fire or an intruder) so as to help the system to handle them, improving its accuracy or resolution. The latter is an instance of active cooperative perception (ACP). Cooperative perception refers to the fusion between different sensory information with as goal maximizing the amount and quality of perceptual information available to the system. These sensors may be spatially distributed as they can either be static or mobile sensors. In turn, active perception means that a particular sensory agent considers the effects of its actions on its sensors and therefore may do actions in order to improve its sensory performance. Combining the two concepts, cooperative active perception is the problem of active perception involving multiple sensors and multiple cooperating decision makers.

Particularly, in robotics, the problem of active cooperative perception might be referred as a robot (or a team of robots) cooperating with a sensor network in order to reduce uncertainty in situational awareness. Cooperative as the information from the sensor network and from robot's sensors is fused in order to improve situational awareness on the system, providing more and better information for the robot to choose its actions, and active as the robot chooses its actions based on the effects they have on its sensors. The robot may select sensory actions, for instance pointing a pan-and-tilt camera or choosing to execute an expensive vision

algorithm, or may influence a robot's path planning, by choosing the most informative route to get to a location.

Dealing with ACP in NRS is a challenging problem, as it goes much further than deal with uncertainty associated with the robot. Everything in the environment has a particular uncertainty associated, being a robot with uncertainty on localization and on the effect of its actions, or uncertainty associated with errors in the sensors. The system must act in a way to decrease uncertainty and general awareness in the system, ideally optimizing the available resources.

In this thesis, we will propose a decision-theoretic approach to cooperative active perception. In particular, we will propose Partially Observable Markov Process Decision (POMDPs) [9] as a framework for cooperative active perception providing an elegant way to model problems of decision-making under uncertainty and to model the interaction between an active sensor and the environment. POMDPs offer a strong mathematical framework which allows to model directly uncertainties associated with the system, as encoding the goal of each particular task. Based on a model of the environment, we may compute policies that tell the active sensor how to act, based on the observations the system receives. Performance of the system may be measured by accuracy or confidence, through the belief of the POMDP.

Mobile sensors, besides considering the quality of observations, must consider the effects of its actions on its observations. We apply such techniques, where fixed sensors and an active mobile sensor cooperate through a network in order to perform tasks. Particularly, we are interested in an environment where we want to detect events of interest and classify them. That is, we are interested to know if a particular feature (or a number of features) is present in the system, and create a classification system which identifies the feature with low uncertainty. To do that, mobile and fixed sensors need to cooperate to improve situational awareness of the system. For instance, some fixed sensors might be more accurate than others, and in that case, if a fixed sensor notices a possible event, it might ask a mobile sensor (typically mounted on a robot) to investigate in the location where the event was detected to improve certainty. However, each robot typically has an associated cost of moving, and the system needs to trade-off between the increasing of observability of the system and the cost of movement. An example of a cooperative active perception problem of this kind is a surveillance task where the event of interest is to detect a particular person using cameras running a face detection vision algorithm.

1.2 Related Work

Although NRS is still a young research field, there is already some notable work done in the area. Namely, some groups have already developed some major projects in order to explore this new concept from which we may mention:

- URUS [10];
- Japan Network Robot Project [11];
- PEIS Ecology [12];
- DUSTBOT [13];
- AWARE [14].

The general objective of the URUS project is the development of new ways of cooperation between network robots and human beings and/or the environment in urban areas, in order to achieve efficiently tasks that in the other way can be very complex, time consuming or too costly [10]. The final experimental goal is to

test the networked robot in two main applications areas: guiding and transportation of people and goods, and surveillance tasks. A networked robot is developed to integrates cooperative mobile robots, intelligent sensors (video cameras, acoustic sensors,etc), intelligent devices (PDA, mobile telephones, etc) and communications. The main scientific challenges to deal in here are navigation and motion coordination among robots; cooperative environment perception; cooperative map building and updating; task negotiation within cooperative systems; human robot interaction; and wireless communication strategies between users (mobile phones, PDAs), the environment (cameras, acoustic sensors, etc.), and the robots.

Japan's NRS project aims to develop user-friendly interaction between humans and networked environments. There is a strong emphasis on human-robot interaction, for instance, by using RFID (Radio-Frequency Identification) to uniquely identify each person and therefore the robot knows exactly with who it is interacting with. There are four main aspects to this project: communication among robots, the networking platform, ubiquitous sensing, and human-robot interaction based on NRS. As an example, a museum guide robot network using ubiquitous sensors [11] has been developed and tested with major success.

PEIS Ecology Project [12], or Ecology of Physically Embedded Intelligent Systems, combines vision from autonomous robotics with ambient intelligence into a new concept. This new approach takes an ecological viewpoint in which robots, humans and devices in the environment are seen as parts of the same ecosystem, engaged in a symbiotic relationship toward the achievement of common goals. The main idea is to take advantage of the new concept to build assistive, personal and service robots.

The DustBot project is aimed at designing, developing, testing and demonstrating a system for improving the management of urban hygiene (cleaning and removing rubbish from the ground) based on a network of autonomous and cooperating robots, embedded in an ambient intelligence infrastructure [13]. Mobile robots and wireless sensors networks are integrated in a networked robot to provide system functionalities.

Finally, the AWARE project aims at the development of a platform that enables the cooperation of unmanned vehicles with ground wireless sensor-actuator networks comprising static and mobile nodes [14]. This project extends the pre-existing work on wireless sensor networks to include mobile nodes in the network in order to overcome limitations of static networks. These platforms offer self-deployment, self-configuration and self-repairing features, which may be very relevant in conditions such as disasters, when the existing infrastructure is destroyed.

Decision-theoretic techniques have been used in the areas of active and cooperative sensing. [15] considers the problem of sensor-aware path planning for a robot in a NRS, focusing on robot navigation and localization, specifically on improving robot localization using the observations from a camera network, but taking into account the specifics of its local sensors. This is an instance of an ACP, as local sensors on the robot cooperate with a camera network in order to perform navigation tasks, adapting the path of the robot to different priorities (reaching the target as fast as possible, reaching the target with the lowest localization uncertainty possible, etc). Therefore, although shows a different applications for ACP in NRS, it focus on a different purpose than our work.

A classification problem has been introduced in the active sensing framework using POMDPs [16], explicitly including sensing actions. In this problem, sensing actions do not change the state of the process, that only occurs when a classification occurs. This may be seen as a similar problem to ours, as the system balances the cost of acquiring additional information with the cost of misclassification. However, this classification system considers only the sensors of the agent, not extending to the NRS framework.

[17] shows another cooperative setting where a networked distributed sensors cooperate in order to choose its gaze direction to target tracking. The sensors must coordinate where they look in order to improve target detection and tracking, but only considers a network of static sensors.

In turn, mobile target tracking is considered in [18], which unifies target searching and target following with a POMDP solution. It does not consider the robot in the context of a network but rather acting alone, using only its local sensors, and do not include any kind of sensing actions. It relies on moving actions to perform in such a way to gather enough information for target localization.

Although not explicitly modeled as a POMDP, [19] studies and presents a decision-theoretic approach to cooperative sensor planning between multiple autonomous vehicles in the context of a military mission.

Other applications for POMDPs have been reported in the literature, namely in problems where dealing with uncertainty is crucial. [20] applies POMDPs to autonomous mobile robots robust navigation and localization. [21] also uses POMDPs for a robot navigation task, integrating visual and sensorial information from the robot for localization improvement. Also in the area of robot navigation, [22] use POMDPs for navigation and localization of mobile robots in a problem with uncertainty where they must find persons in a health care facility as fast as possible, without knowledge of their initial position and [23] studies and develops POMDP models for assistance for persons with dementia, aiding them to complete activities of daily living. Generally, as state of the art algorithms for solving POMDP are becoming more efficient and less computational expensive, the applicability of models are becoming wider. [24] goes through some of the possible areas where POMDPs may apply, from industrial to business applications.

1.3 Objectives

This project's goal is to study planning in NRS, namely to study the problem of ACP in NRS. In order to do that, we will see how to develop a Networked System which integrates a multitude of active sensors in a cooperative way, and how to model an ACP problem in such environments. We want to prove that we are able to model such problems with a decision-theoretic approach, namely we want to use POMDPs as the framework to model ACP problems. We will highlight the advantages of using POMDPs for decision-making in NRS, namely how such framework is useful to tackle uncertainty.

In order to prove that our models work in the real world, we apply our studies in experimental problems. Particularly, we will go for a surveillance task where robots (acting as mobile sensors) and a sensor network (fixed sensors) cooperate in order to detect events of interest. The scenario for our experiments will be the Mobile Robotics Lab (LRM - Laboratório de Robótica Móvel), located in the 8th floor at the Institute for Systems and Robotics (ISR) at Instituto Superior Técnico.

Therefore, the expected outcomes of this work are:

- To develop a POMDP model which tackles the problem of ACP in NRS. Such model should be as general as possible in order to easily adapt it to any ACP problem;
- Show how to develop a Networked System, and how to integrate all elements in the system.
- Perform a series of experiments where a robot and a sensor network cooperate in order to perform an ACP task, in particular in our experiments, to detect events of interest.

1.4 Thesis Overview

The present thesis is divided into 4 chapters, aside from the introduction, and is organized as follows:

Chapter 2, Background, provides a small introduction on planning under uncertainty and goes through the POMDP theory, with a first look over MDPs, and a final look over the algorithm used for solving the POMDP.

Chapter 3, POMDPs for Cooperative Active Perception, introduces how we define our POMDP model for cooperative active perception, goes through states, observations, transitions, actions and rewards.

Chapter 4, Experiments, describe all the experiments made with our model. In this chapter we include a modular description of the experimental setup.

Chapter 5, Conclusions and Future Work, closes this thesis and provides guidelines for future work.

Chapter 2

Background

In this chapter,we introduce the problem of planning under uncertainty. After a brief review on the problems which arise when making decisions in an uncertain environment, we study the background on Markov Decision Processes and how can an agent interact with a stochastic environment using this framework.

Furthermore,we will see what are the changes when the agent has partially observability over the environment, how we may represent it with factored representations and explain the Perseus algorithm, used during this thesis.

2.1 Planning under uncertainty

A planning problem in artificial intelligence is usually described as: given a description of the current state of some system, a set of actions that can be performed on the system and a description of a goal set of states for the system, find a sequence of actions that can be performed to transform the system into one of the goal states [25]. The objective is to build intelligent agents which are able to compute a plan to perform an assigned task as well as possible, autonomously. An agent is a general concept, which may include from robots to intelligent computer programs, and is defined as a system which observes the environment and takes actions towards achieving goals. Figure 2.1 shows the operation of a general agent interacting with its environment. The agent lives in an environment, and is able to observe by its sensors the actual state of the environment, i.e., the description of the world at a given moment. The agent chooses and executes actions in order to change the state of the environment such that in a long term it reaches a goal state. It influences the environment by acting and can detect the effect of its actions by sensing.

However, when applying techniques in real-world problems, inevitably, the agent must deal with uncertainty because there is no perfect models of the world, every model is an approximation to reality, and there is always unpredicted external influences on the system. Uncertainty may be divided in two different types: acting and sensing uncertainty. Acting uncertainty is related to the unexpected outcomes that might come from an action, for instance, when driving a robot it might not end up in the desired location, due to wheel slip or if an unexpected obstacle appears on the way. In the other hand, sensing uncertainty arises from sensor noise and sensors might have a limited view of the environment. Uncertainty is usually captured in problems using probabilistic models, assigning a given probability for each of the possible outcomes. If we have either deterministic actions or perfect observation of the environment then naturally we are sure of the actual state. However, when combined, acting and sensing uncertainty lead to uncertainty in the actual knowledge of the environment and in the actual state.

While classical planners focused exclusively on the task of reaching a goal state by computing plans in

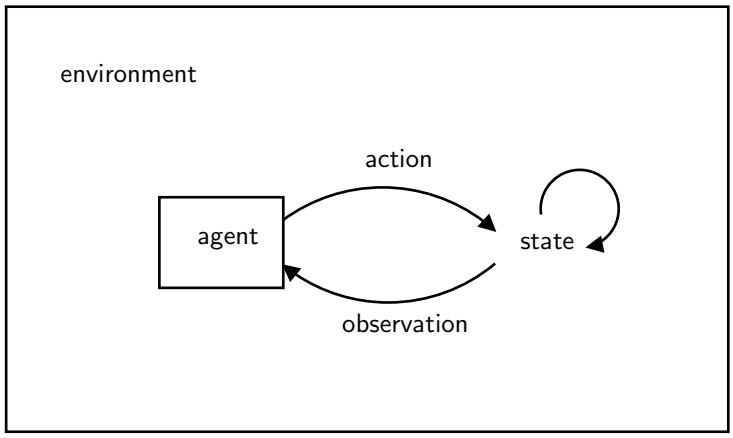


Figure 2.1: Diagram of an agent. The agent live in an environment which is at a particular state, it receives information about the state and acts in the environment according to the information received.

deterministic environments, decision theory weights a probability distribution over the possible outcomes of an action in any state with a given preference function over outcomes, defining a utility function such that the preferred plan has a higher utility. Therefore, both techniques have been combined in order to model and solve problems where actions have uncertain effects and there isn't a well defined goal but rather a preference between states. Also, in a different approach, Markov Decision Processes (MDPs) have been developed as a different framework for planning under uncertainty. It defines preference over states by giving a reward for doing an action at a state, and focus on finding the plan which returns the highest long-term cumulative reward. In a more general concept, the MDP framework is generalized to a partially observable environment in order to tackle with sensor uncertainty.

2.2 Markov Decision Processes

Markov decision processes (MDPs) [26] [27] provides a rich framework to model the interaction of an agent with a stochastic environment where it is confronted with a number of possible actions to take, having to take decisions.

A MDP can be specified as a tuple $(\ S,\ A,\ T,\ R,\ \gamma)$ where:

- S is a finite set of environment states that can be reliable identified by the agent;
- A is a finite set of actions available to the agent;
- $T : S \times A \rightarrow \Pi(S)$ is a state transition model of the environment. It gives for each state $s \in S$ and action $a \in A$ a probability distribution over world states, representing the probability that the resulting state is s' when the the world is at a state s and a given action a is taken: $T(s', a, s) = p(s'|s, a)$;
- $R : S \times A \rightarrow \Re$ is a reward function that specifies the expected immediate reward $R(s, a)$ received by the agent for taking an action a in a state s.
- γ is the discount factor, which is used to weight rewards received over time. Therefore, rewards received in a distant future are less valued.

The state is the way the world currently exists, describing the system, which consists of both the agent and the environment, at a certain point in time, and should contain all the information needed for decision making. In order for the process to be Markovian, the current state must provide enough information in order to predict the next state. Given any current state and an action to take, in an MDP the future state depends

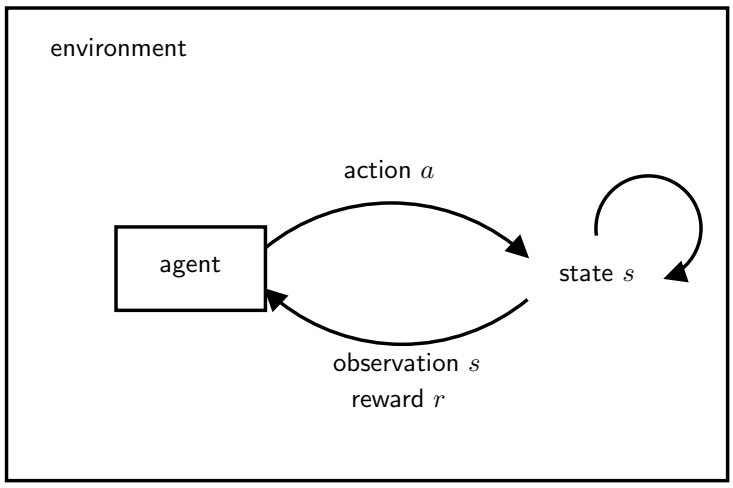


Figure 2.2: Diagram of a MDP agent interacting with the environment

only on those, forgetting about all the previous states, that is, mathematically, given the actual state and action, the future state is conditionally independent of all previous states and actions.

Actions are the set of every possible alternative choices the agent can choose to make. The goal in MDPs is to find the best action to take on each state.

State transitions models how the environment responds to the interaction of the agent. The state of the system can be changed every time the agent takes as action, however there can be uncertainty about the effects of this action. This is captured by transitions, which are probabilistic, specifying a set of resulting states and the probability that each state results.

In order to automate the decision making, there should be any measure on how good or bad is the combination of being in some state and taking some action. This is encoded by the reward function, giving this combination some numerical value. Good actions receive positive rewards while bad actions get a negative reward. This is also useful to define priorities when the agent can execute different tasks, as it can be given different rewards for each task. In the long term, a given plan performance is measured by how much reward the agent is expected to obtain by following that plan. Figure 2.2 shows the diagram of an MDP agent interacting with the environment.

More formally, at any time step the environment is in a state $s \in S$, the agent takes an action $a \in A$ and receives a determinist reward $R(s, a)$ from the environment, while the environment state changes to a new state s' according to a known stochastic transition model $p(s'|s, a)$. Since MDP possess the Markov property, the next state s_{t+1} only depends on the previous state s_t and action a_t:

$$p(s_{t+1}|s_t, s_{t-1}, \ldots, s_0, a_t, a_{t-1}, \ldots, a_0) = p(s_{t+1}|s_t, a_t) \tag{2.1}$$

An agent's goal is to act in order to maximize some measure of long-run reward. Such performance measure is usually called a criterion of optimality. One possible criterion is the finite-horizon model, in which the performance of the agent is measured by the expected cumulative reward after acting for a finite number of steps, usually known as the planning horizon:

$$E\left[\sum_{t=0}^{h} R_t\right] \tag{2.2}$$

where $E[.]$ denotes the expectation operator, h is the planning horizon and R_t is the reward received at timestep t.

Another optimality criterion introduces the notion of discount rate, which models the concept that rewards received earlier in the agent's lifetime are more valuable than later rewards. In this model, the optimality is measured by the expected discounted cumulative rewards over horizon length:

$$E\left[\sum_{t=0}^{h}\gamma^t R_t\right] \tag{2.3}$$

where γ is the discount rate, $0 \leq \gamma \leq 1$. However, for some other problems, for instance when h is unknown, we may use an infinite-horizon model:

$$E\left[\sum_{t=0}^{\infty}\gamma^t R_t\right] \tag{2.4}$$

In this case, the discount factor, besides ensuring that a reward received in the near future is more valuable than a reward receives in the far future, is crucial to ensure that this performance measure has a finite sum. In the following we will assume an optimality criterion based on the expected discounted cumulative reward, and though we use a notation with a horizon length, those notions are general, even if $h = \infty$.

A policy $\pi : S \rightarrow A$ is a mapping between states and actions. More exactly, it indicates what action $a \in A$ should an agent take when the world is at state $s \in S$. From every possible policies we may distinguish the optimal policy $\pi^* : S \rightarrow A$, which indicates for every state which is the optimal action that the agent should execute, given that in the future it continues to follow π^*. That is, the optimal policy is the policy which in the long run maximizes the considered performance measure. Each policy has associated with it a value function $V^\pi : S \rightarrow \Re$, which maps a state $s \in S$ to a real value. For each policy the value function returns the expected cumulative reward the agent receives when starts from state s and follows the policy π.

Using the criterion of optimality (2.3), the value function of state s, $V^\pi(s)$ is:

$$V^\pi(s) = E\left[\sum_{t=0}^{h}\gamma^t R_t\right], \tag{2.5}$$

Such value function may be rewritten as:

$$V^\pi(s) = R(s,\pi(s)) + E\left[\sum_{t=1}^{h}\gamma^t R(s_t,\pi(s_t))\right]. \tag{2.6}$$

The expectation operator averages over the stochastic transition model, which leads to the following recursion, known as the Bellman recursion:

$$V^\pi(s) = R(s,\pi(s)) + \gamma\sum_{s'\in S} p(s'|s,\pi(s))V^\pi(s') \tag{2.7}$$

The policy π corresponding to a particular value function can be easily extracted by the equation:

$$\pi(s) = \underset{a\in A}{\operatorname{argmax}}\left[R(s,a) + \gamma\sum_{s'\in S} p(s'|s,a)V(s')\right] \tag{2.8}$$

which instructs the agent to take the action which maximizes not only the immediate reward, but which as part of a long run policy maximizes the sum of immediate reward and future discounted rewards.

2.2.1 Value Iteration

An optimal policy π^* refers to the one which maximizes the considered performance measure, which is

incorporated in the value function. Therefore, the optimal value function is denoted $V^*(s)$ and is obtained by the called Bellman equation:

$$V^*(s) = \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^*(s') \right] \tag{2.9}$$

This forms a system of equations for each state s, which solved returns the optimal value function, and from it an optimal policy using (2.8).

Solving this system explicitly involves a large request for computational resources and might not be feasible. Therefore, a successive approximation technique as been introduced called value iteration, which turns (2.9) into an update, computing the optimal value function of each state by iterating over successive steps into the future. First the value function is initialized at the optimal value function V_0^*, defined as the maximum reward the agent gets when it can take only one action:

$$V_0^*(s) = \max_{a \in A} R(s,a) \tag{2.10}$$

The value function is than iterated over time, following the rule:

$$V_{n+1}^*(s) = \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V_n^*(s') \right] \tag{2.11}$$

This operation is known as a Bellman backup [28], denoted as H_{MDP}, and converges to the fixed point V^*. We may rewrite (2.11) as:

$$V_{n+1}^* = H_{MDP} V_n^* \tag{2.12}$$

It is iterated until the value function has converged, i.e., until the largest state value update during one step iteration is below a certain threshold.

The algorithm for value iteration solving of MDPs is shown in Algorithm 1.

Algorithm 1 Value iteration

Require: A stopping threshold ϵ
Ensure: An approximate to optimal value function V^* with Bellman error less than ϵ
  Set $V(s) = 0$ for all $s \in S$
  repeat
    for all $s \in S$ **do**
      $V'(s) \leftarrow V(s)$
      $V(s) \leftarrow \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^*(s') \right]$
    end for
  until $\max_{s \in S} |V'(s) - V(s)| \leq \epsilon$

2.3 Partially Observable Markov Decision Processes

MDPs are able to model uncertainty on state transitions, but assume that the agent has completely knowledge about the environment at every point in time. However, that can restrict the applicability of the framework, as the agent might not be able to gather such information. A partially observable Markov decision process (POMDP) [29] extends the MDP setting to deal with uncertainty in state observation. In general, partially observability results when the agent can only sense a limited part of the environment and multiple states give the same sensor reading or when its sensors are noisy and different observations of the same state

can result in different sensor readings. Figure 2.3 depicts a POMDP agent interacting with the environment, note the difference to Figure 2.2: instead of receiving the actual state from the environment, the agent receives an observation, which is captured by a probabilistic model.

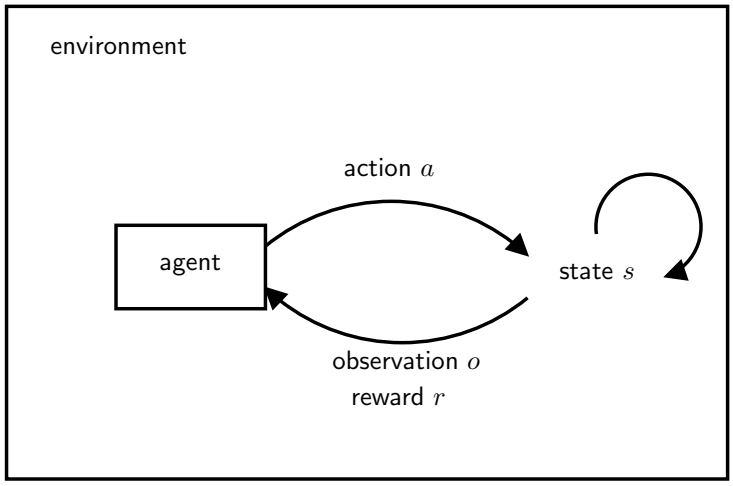


Figure 2.3: Diagram of a POMDP agent interacting with the environment

The POMDP model acts like the fully observable MDP model: the agent takes an action a in state s, the environment transitions to state s' according to $p(s'|s,a)$ and the agent receives an immediate reward $R(s,a)$. However, due to the partially observable property the agent perceives an observation $o \in O$, that may be conditional on its action, which provides information about the state s' through a known stochastic observation model $p(o|s',a)$.

In order for an agent to choose its actions successfully it is important to maintain an history, or provide some memory to the system, of the agent's life, as observations in partially observable environment do not provide unique state identification, but without leaving the Markovian framework. A straightforward implementation of memory would be to simply store the sequence of actions and observations received. However, such implementation may grow indefinitely over time, turning it impractical for large problems. It has been proved that by maintaining a probability distribution over all of the states we may keep the same information as saving the complete history. A probability distribution over S is a belief vector $b(s)$, denoting the probability that the environment is in state s. Since the size of the set of states S is finite, the belief vector is finite with size $|S|-1$, where $|S|$ is the size of the state space, and so the physical memory needed to keep a belief is minimal, since it reduces to store $|S|-1$ numbers. Given the transition and observation model the agent summarizes all information about its past using a belief vector transforming the POMDP to a belief-state MDP. The belief b forms a Markovian signal for the planning task, as maintaining a belief over the states only requires knowledge of the previous belief, the action taken and the current observation. The belief is updated every time the agent takes an action a and receives an observation o by the Bayes' rule:

$$b_a^o(s') = p(s'|o,a,b) = \frac{p(o|s',a)}{p(o|b,a)} \sum_{s \in S} p(s'|s,a) b(s) \tag{2.13}$$

where $p(o|a,b) = \sum_{s' \in S} p(o|s',a) \sum_{s \in S} p(s'|s,a) b(s)$ is a normalizing constant.

Formally, a belief-state MDP is a completely observable continuous-space MDP, defined as a tuple $(B,\ A,\ r,\ \tau,\ \gamma)$ where B is the set of belief states, A is the set of actions, r is the reward function constructed from the MDP reward taking expectations according to the belief state and γ is the discount factor:

$$r(b,a) = \sum_{s \in S} b(s) R(s,a) \tag{2.14}$$

and τ is the belief state transition function:

$$\tau(b,a,b') = p(b'|b,a) = \sum_{o \in O} p(b'|b,a,o) p(o|b,a) \tag{2.15}$$

where $p(b'|b,a,o) = 1$ if $SE(b,a,o) = b'$ or $p(b'|b,a,o) = 0$ if $SE(b,a,o) \neq b'$. SE is the state estimator, which updates the belief state as defined in (2.13).

2.3.1 Value Function

The goal, like in the fully observable MDP, is to compute an optimal plan such that the agent successfully completes its tasks. The way to compute optimal plans is similar to the one used in the MDP framework, however MDP policies maps states to actions, while POMDP policies maps beliefs to actions: $\pi : B \rightarrow A$. While a policy $\pi(s)$ is a function in a discrete domain, since we are considering only discrete MDPs and POMDPs, a policy $\pi(b)$ is a function over a continuous set of probability distributions over S. A policy $\pi(b)$ is also characterized by a value function $V^{\pi}(b)$ which is defined as the expected future discounted reward $V^{\pi}(b)$ the agent can gather by following π starting from belief b with the planning horizon h:

$$V^{\pi}(b) = E_{\pi}\left[\sum_{t=0}^{h} \gamma^t r(b_t, \pi(b_t)) \middle| b_0 = b \right], \tag{2.16}$$

An optimal policy π^* is the one which maximizes V^{π}. For each belief b it specifies the optimal action to execute at the current timestep in order to maximize the expected cumulative reward. The optimal value function satisfies the Bellman optimality equation, $V^* = H_{POMDP} V^*$:

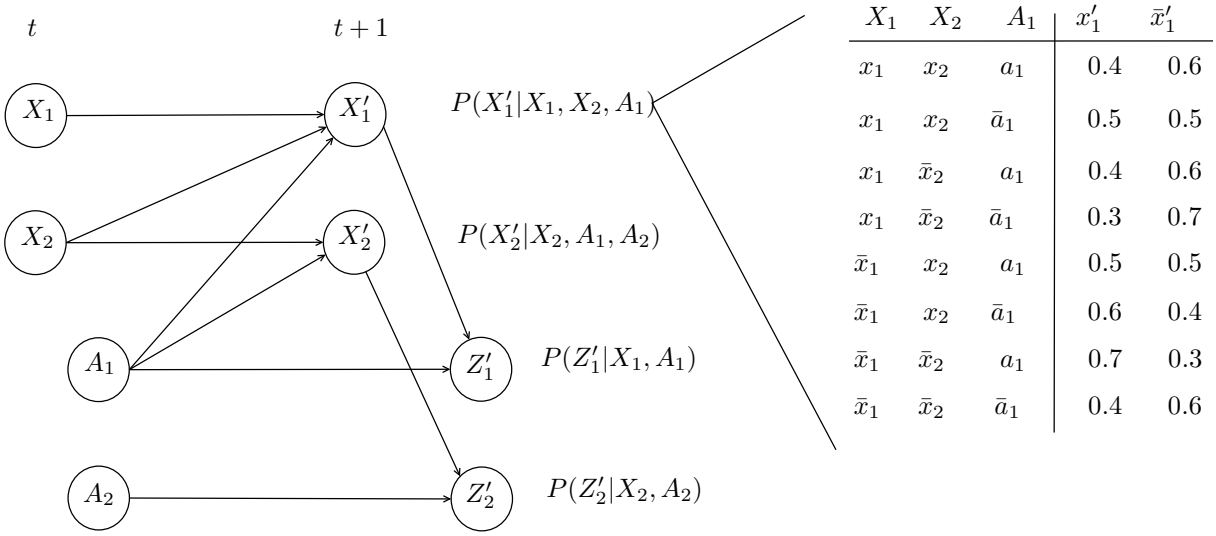
$$\begin{aligned} V^*(b) &= \max_{a \in A} \left[r(b,a) + \gamma \sum_{b'} \tau(b,a,b') V^*(b') \right] \\ &= \max_{a \in A} \left[r(b,a) + \gamma \sum_{o \in O} p(o|b,a) V^*(b_o^a) \right] \end{aligned} \tag{2.17}$$

The solution is optimal when (2.17) holds for every belief b in the space defined by the state space S.

The optimal value function V^* can be approximated by iterating (2.17) a number of stages. V^* will be piecewise linear and convex for problems with finite planning horizon [30]. So, in each time step n, the value function can be parameterized by a finite set of vectors $\{\alpha_n^i\}, i = 1, \ldots, |V_n|$, and an action $a(\alpha_n^i) \in A$ is associated with each vector, which is the maximizing vector for the current step. The belief space is divided into regions, each one defined by each vector. Given a set of vector $\{\alpha_n^i\}_{i=1}^{|V_n|}$ at stage n, the value of a belief b is given by:

$$V_n(b) = \max_{\{\alpha_n^i\}_i} b \cdot \alpha_n^i \tag{2.18}$$

The gradient of the value function at b is given by the vector $\alpha_n^b = \operatorname{argmax}_{\{\alpha_n^i\}_i} b \cdot \alpha_n^i$, and the policy at b is given by $\pi(b) = a(\alpha_n^b)$.



X_1	X_2	A_1	x'_1	$\bar{x}'_1$
x_1	x_2	a_1	0.4	0.6
x_1	x_2	$\bar{a}_1$	0.5	0.5
x_1	$\bar{x}_2$	a_1	0.4	0.6
x_1	$\bar{x}_2$	$\bar{a}_1$	0.3	0.7
$\bar{x}_1$	x_2	a_1	0.5	0.5
$\bar{x}_1$	x_2	$\bar{a}_1$	0.6	0.4
$\bar{x}_1$	$\bar{x}_2$	a_1	0.7	0.3
$\bar{x}_1$	$\bar{x}_2$	$\bar{a}_1$	0.4	0.6

Figure 2.4: Example of a Dynamic Bayesian Network which encodes the states and transitions of a MDP problem [1].

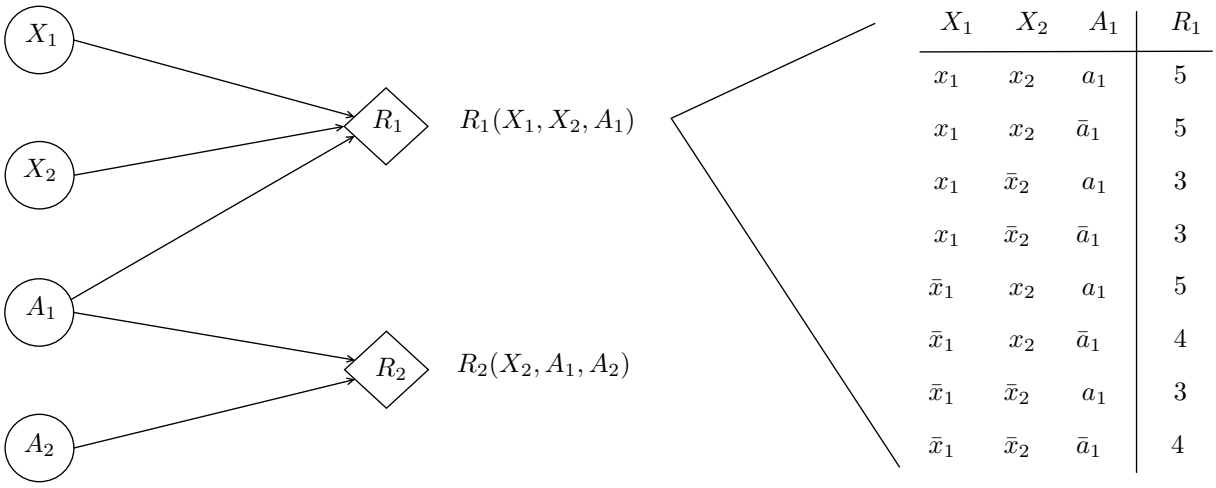
2.4 Problem representation

The definition of the state space in a Markovian problem is of extreme importance, since it defines how we model a description of the world. Usually it is considered a state space S consisting of a discrete and finite set of states, without assuming any structure. However, we may consider for each particular problem a number of different features which characterize the environment. If each feature is represented by a variable S_i, the state of the system may be seen as a cross-product of all state variables: $S = S_1 \times S_2 \times \ldots \times S_k$ (assuming an environment with k features). In the same way, the action space A and the observation space O may also be decomposed such that each action a and observation o correspond to a joint-instantiation of variables. Such representation is called a factored representation [1], which allows for the transition, observation and reward functions to be defined in terms of state variables, actions variables and observations variables (instead of simply states, actions and observations). This also prevents the need to explicitly define all states in the problem, which grow exponentially with the number of variables.

To compactly represent transition and observation functions usually they are represented as a dynamic Bayesian network (DBN) [31] [32]. DBNs take advantage and exploits conditional independence, which refers to the fact that some variables are probabilistically independent of each other when other variables values are held fixed.

Transitions and observation probabilities ($P(s'|s,a)$ and $P(o|s,a)$) are expressed in a DBN, by an acyclic directed graph. The graph is explicitly divided in two parts, corresponding each one to a timestep, being one immediately posterior to the other. Nodes represent state, action and observation variables and edges probabilistic dependencies. Dependencies go either from the previous to the current timestep or from the current to the same timestep. Each node in the posterior timestep has a conditional probability table (CPT) which defines the conditional probability distribution $P(X'_i|parents(X'_i))$. This tables defines which are the parents of each node, that is, of which variables does each node depend at the next timestep. Therefore, the transition function of the model may be factorized in a product of smaller conditional distributions. In the example of Figure 2.4, if we consider the state as a whole the transition function is represented as $P(X'_1, X'_2|X_1, X_2, A_1, A_2)$. However, by using a factored representation the transition is reduced to the product of the transitions of variables X_1 and X_2: $P(X'_1, X'_2|X_1, X_2, A_1, A_2) = P(X'_1|X_1, X_2, A_1) \cdot P(X'_2|X_2, A_1, A_2)$. The same procedure is used for the observation function : $P(Z'_1, Z'_2|X'_1, X'_2, A_1, A_2) = P(Z'_1|X'_1, A_1) \cdot P(Z'_2|X'_2, A_2)$.

The reward function may also take advantage of a factorized representation and be compactly repre-



X_1	X_2	A_1	R_1
x_1	x_2	a_1	5
x_1	x_2	$\bar{a}_1$	5
x_1	$\bar{x}_2$	a_1	3
x_1	$\bar{x}_2$	$\bar{a}_1$	3
$\bar{x}_1$	x_2	a_1	5
$\bar{x}_1$	x_2	$\bar{a}_1$	4
$\bar{x}_1$	$\bar{x}_2$	a_1	3
$\bar{x}_1$	$\bar{x}_2$	$\bar{a}_1$	4

Figure 2.5: Example of how to encode the reward function of a MDP in a Dynamic Bayesian Network [1].

sented exploring the concept of additive separability [33]. For example, the reward function in Figure 2.5 $R(X_1, X_2, A_1, A_2)$ may be represented as the sum of two smaller reward functions, each one depending on a smaller subset of variables: $R(X_1, X_2, A_1, A_2) = R_1(X_1, X_2, A_1) + R_2(X_2, A_1, A_2)$.

The tables presented in the figures may be represented in an even more compact way, through decision trees (DT) [34] and algebraic decision diagrams (ADD) [35]. DTs keep in its leaves the value of a function for a particular value of its variables. To find a value, one should follow the corresponding branch in the tree. DTs and ADDs take advantage of context-specific independence [36], which is a concept that refers to the fact that some variables are independent for some specific contexts (e.g., for some specific values), leading to an even bigger compression of the function representation. For instance, in Figure 2.6 the reward is 5 when X_1 and X_2 are true, independently of the value of A_1. Therefore, the DT may be represented with 5 leaves, instead of the 8 leaves in the CPT. ADDs are even more compact, allowing for branch merging, which in the example leads to a reduction for a representation with only 3 leaves.

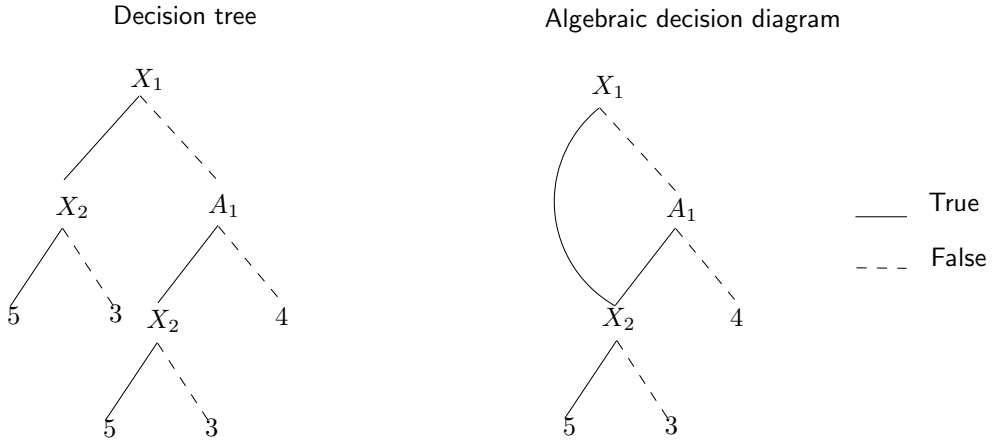


Figure 2.6: Decision Tree and Algebraic Decision Diagram for the reward function presented in Figure 2.5 [1].

2.5 Perseus

A number of exact value iteration techniques have been developed, however such techniques are not suitable for large POMDPs problems, as the size value functions might grow exponentially as they are computed exactly.

Another set of techniques compute an approximate solution for the value function, as they consider only a finite set of belief points. In this thesis we used the PERSEUS method [37], in particular the Symbolic Perseus [1]. Symbolic Perseus uses the original Perseus algorithm with Algebraic Decision Diagrams (ADDs) [35] as a way to compactly represent α-vectors. The symbolic version of Perseus uses a factored representation of transition, observation and reward functions, allowing in many cases to enumerate state, action and observations space as a joint instantiation of state, action and observation variables.

Algorithm 2 Perseus backup stage:$V_{n+1} = \widetilde{H}_{PERSEUS}V_n$

$V_{n+1} \leftarrow \emptyset,\ \widetilde{B} \leftarrow B$
Sample belief b uniformly at random from $\widetilde{B}$ and compute the new α-vector for that point
if $b \cdot \alpha \leq V_n(b)$ **then**
  $\alpha = \text{argmax}_{\{\alpha_n^k\}_k}\, b \cdot \alpha_n^k$
end if
$V_{n+1} \leftarrow V_{n+1} \cup \{\alpha\}$
$\widetilde{B} = \{b \in B : V_{n+1}(b) \leq V_n(b)\}$
if $\widetilde{B} \neq \emptyset$ **then**
  Goto 2
end if
Return V_{n+1}

Perseus is a randomized version of approximate point-based iteration algorithms. Point-based as it acts in a set of belief points, obtained by letting the agent randomly explore the environment and collect a set B of reachable points, as even with large arbitrary action and observation sequences it is unlikely that most POMDP problems reach most of the points in the belief simplex. The backup operator for approximate methods is denoted $\widetilde{H}$. And randomized as it tries to update as much belief points as possible from the chosen set by randomly choosing one point, compute the new α-vector for this point and look over all the set points, which have the value function improved by the new vector. This method does not require backing up all belief points in the set.

Perseus algorithm starts with a V^0 and performs a number of backup stages until meet some criterion. Algorithm 2 describes the procedure in the backup stage of Perseus.

Perseus backup stage receives the set B of belief points and initializes $\widetilde{B}$ to B, randomly selects one point and computes the new α-vectors for that point. If the new vector does not increase the value of the selected point, it is discarded,and the point remains with the maximizing vector for the previous stage, otherwise the vector is inserted in the value function and all other points in the set which are improved by this vector are discarded from the $\widetilde{B}$. This is repeated until $\widetilde{B}$ is empty and the new set of α-vectors representing the new value function is returned.

Chapter 3

POMDPs for Cooperative Active Perception

In Chapter 2 we introduced the problem of planning under uncertainty, we went over the MDP framework and its generalization for a partially observable environment. In our work, we intend to use POMDPs as the framework to model and solve a surveillance task, where mobile and fixed sensors cooperate to detect and classify events of interest. In this chapter, we will first explain how this problem copes with cooperative active perception and how interesting may it be to real world applications. Then, we explain how to tackle such kind of task in the POMDP framework.

3.1 Problem definition

Within a cooperative active perception problem two conditions are mandatory: an agent must consider the effect of its actions on its sensors (active) and the system fuses between different sensory information (cooperative). The POMDP framework immediately captures the notion of active perception, as the model includes transition and observation models. These probabilistic models capture the possible different effects an action has on the agent and on its sensors and influence planning decisions. Cooperation, in turn, is obtained by the way as the model is built, that is, the model must include information in some way which informs that different sensing sources are available and may cooperate in the system.

In order to formalize such kind of problems, we propose to apply the POMDP framework in a particular problem, namely to create a classifier system which decides whether a detected event is one of interest. An event of interest may vary on each application, but in general we may define it as a particular occurrence in the system, among every possible ones. For instance, a straightforward application of such problems is a surveillance task. Imagine an environment in which we have a space (a warehouse, for example) with a valuable content. In this context, it is of the highest interest to keep it safe from thieves, and as such, the event of interest is to detect suspicious persons. If the surveillance is to be performed exclusively by the autonomous system, we may classify as suspicious any person movement detected in the area. On the other side, if in addition there is still a human guard, or if we want the workers to be able to move freely in the area, the system might have to distinguish between suspicious and non suspicious movements. This may be done by a network of cameras around the area, which besides detecting persons walking in the area, may run an additional feature algorithm in order to be able to distinguish if that person is allowed to be there or not. Features may vary depending on each case, as examples we may refer face detectors, or if workers or guards have a particular uniform the feature may be a particular symbol, or the uniform color. In this example, an

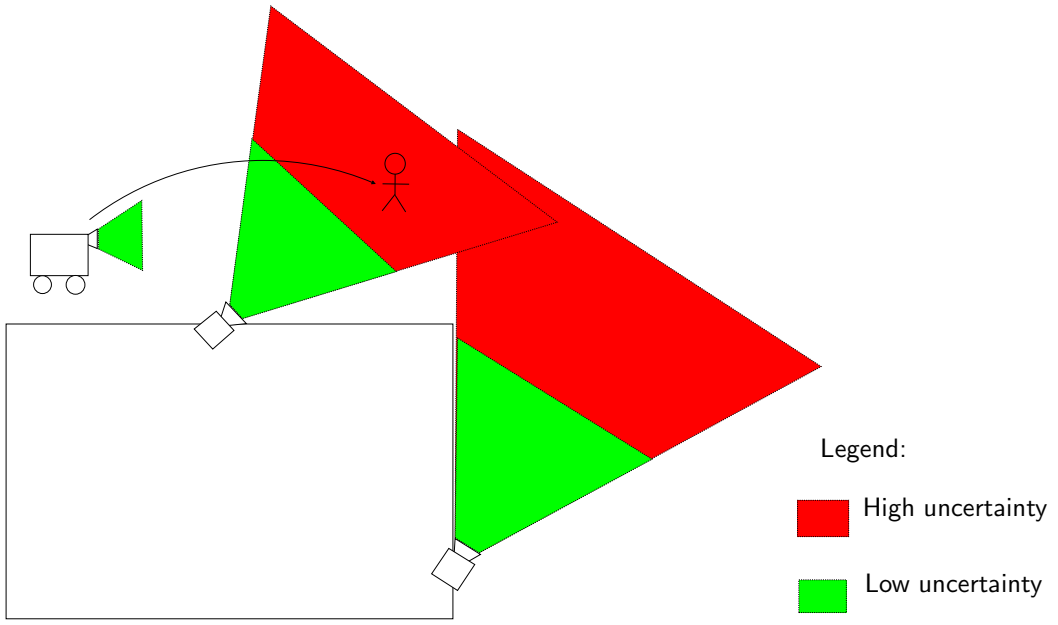


Figure 3.1: Surveillance task example, a person is detected in a high uncertainty area regarding event detection. Therefore, the robot moves to observe the person with its low uncertainty camera, and decrease general uncertainty in the system.

event is the detection of a person in the area, and from all possible events we want to detect when one is of interest, i.e., when the detected person is not allowed in this area.

An example of the situation described is shown in Figure 3.1. Here we have a possible scenario for a surveillance task: a central building which needs to be protected, a camera network mounted around it, and a robot with an additional camera. For explanation purposes, imagine this is a restricted area, where only a few people have authorization to access, and every camera runs continuously a face detector algorithm on every person detected walking in the area. The main issue in this setup is the fact that every camera has a particular field of view, and some areas have a high uncertainty for face detection, due to bad light condition, long distance to camera, etc. That is the problem we want to tackle by cooperation between robot and wall cameras. Although robot's camera has a small field of view, it has a low uncertainty when detecting person's faces, increasing the general awareness in the system. If someone is detected in the field of view of a wall camera, but in a high uncertainty area, the system may command the robot to move such in a way to increase observability. However, moving a robot has some associated costs, for instance, with energy, and the system needs to choose carefully what to do at every step.

3.2 Base Model

There is a clear similarity between our model for ACP in NRS and models used for robot navigation, localization and target tracking. That is not a coincidence, in the manner that those problems form the basis for our work. Note that some of the basic characteristics of our model are robot navigation and localization and person localization. At any timestep the robot should be able to localize itself and be able to navigate to any point in the environment. Also, the system should be able to detect and track a person in the environment. This is common to some already developed work [38] [22], with the difference that most of the work presented considers a robot acting alone and not in a network. Since sensors are not explicitly modeled in the POMDP model, we are still able to develop a cooperative system in a network, with part of it based on robot navigation and localization and with a series of different sensors in the network.

In Figure 3.2 we present a graphical representation of a model for what we call a 'meet person' task, in

which a robot should meet a person detected at a particular location by a camera network or any kind of sensor network used. We use two-stage dynamic Bayesian networks to represent POMDP models, which allows to formalize the model as a set of independent variables and use solvers which exploit such independence.

In this model only two types of information about the environment are in order, to know the position of the robot and the person, such that the system is able to accurately send the robot to meet the person. Therefore, the formal description of the model reflects those needs, including two state variables for localization: person location P and robot location X. Each localization variable can take a number of values, corresponding to different positions in the environment. Typically, such representation is based in a discretized description of the environment, since we are considering only discrete POMDP models. Note that these models are easily expandable, so we may include more robots or persons in the environment. It might be interesting for a particular task to study the effects of adding more robots and how the system deals with it. Typically, we expect that the closest robots are sent to move, minimizing any possible moving cost. Also, if there are multiple persons in the environment, the system needs to minimize the movements, and therefore try to send the robot through the shortest path which passes by all person locations.

As we may see in the DBN, the transition model for person location depends only on its actual position, as we usually don't have any clue about what the person will do. However, in problems in which we have more information, and we are able, for instance, to define a set of possible goals for the person, we may add one more variable to model such goals, and the transition model might include those variables as we expect the person to move towards one of the possible goals. In the other hand, robot location depends not only on its actual value, but also on the action it executes, which is typically to move in a defined direction.

The set of observations (person location O^P and robot location O^X) translates the observations needed for the system. Those variables denote the observed positions for person and robot, with some uncertainty. The observation system is usually considered to detect the correct position with a high probability, but sometimes it might incorrectly infer the position to be at some of the closest neighbours.

In this basis model, there is only one kind of action available to the agent. The agent in this case, is the robot, which is the only element the decision system may control in the environment. This kind of action includes actions which instructs the robot to move in a particular direction or to a particular location. There may be several ways of encoding actions, such as number the neighbour positions (or nodes, in a graph-like map) and define one action to move to each neighbour, or use cardinal points to represent directions, among others.

Finally, the system receives a positive reward whenever $P = X$, that is, when the robot is at the same position as the person. This forces the system to take decisions in order to take the robot to positions near the person. With the information from the transition model, and the possible movements of the person, the robot might execute predictory paths, which account not only with the actual position of the person, but also tries to minimize eventual future movements.

This is a simple model, but very important, in such a way that it allows to form the basis for other problems where robot and person movement and detection are important, as is the case of the ACP problem we want to model. Moreover, such model shows the advantages of using a POMDP framework to model such problems. Note that such models are easily adaptable to each particular problem, as we may only need to adapt the model to each particular problem and recompute the solution with a POMDP solver. In the following section we explain how to go from the base to a complete ACP model.

3.3 Model for Cooperative Active Perception

The similarities between the previous referred models, beside the description of the problem, may be seen

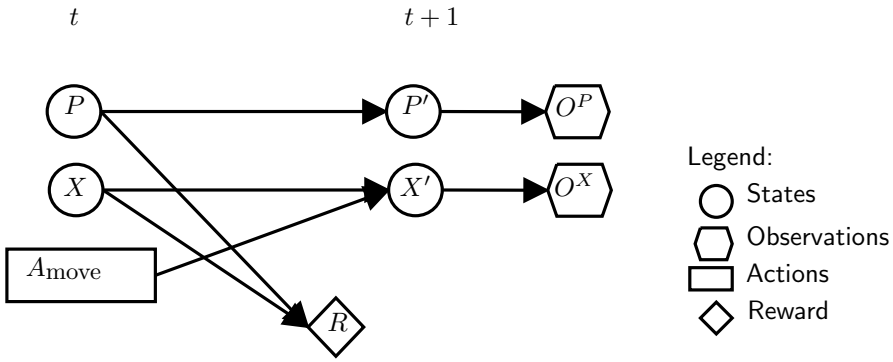


Figure 3.2: Two-stage dynamic Bayesian network representation of a 'meet person' model.

in the formal description of the DBNs. In Figure 3.3 we present a graphical representation of a POMDP model for ACP, namely for our particular problem, tracking and classification. Such model may be applied to any general tracking and classification problem, with proper changes in state and observation variables, by including additional robots, persons or features to be detected, at the expense of higher computation cost, or, in a wider sense, can form the basis to model any ACP problem, by adapting it to each particular task. For the sake of clarity, in the represented figure we assume a problem where we want to classify features of persons in an environment with a single robot.

We will go now to a more detailed description of our model, namely the set of states and observation, and the transition and observation models, used to model uncertainty, the set of actions available to the system and the reward model.

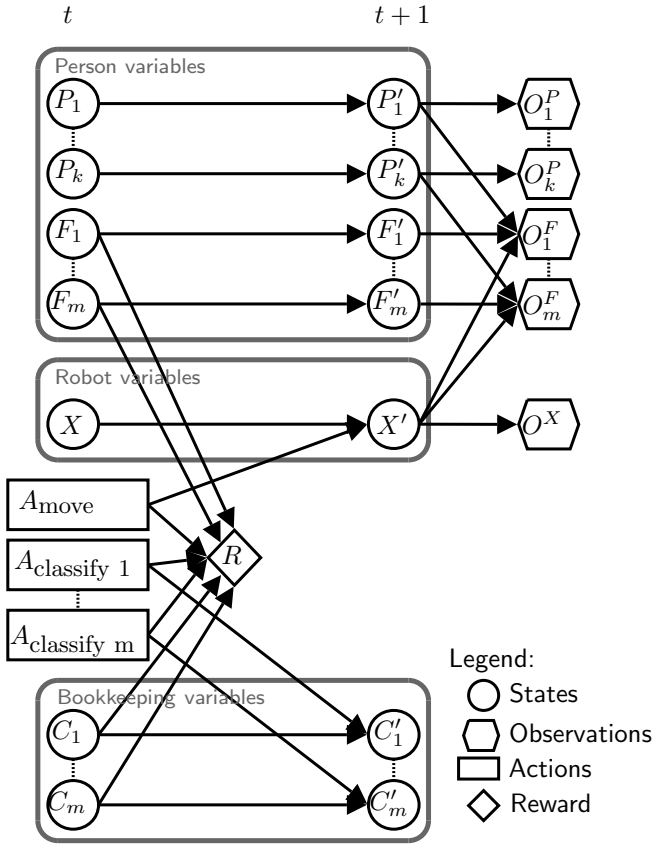


Figure 3.3: Two-stage dynamic Bayesian network representation of the proposed POMDP model. In this figure for simplicity we assume $m = k$, i.e., each person has only one feature.

3.3.1 States and transitions

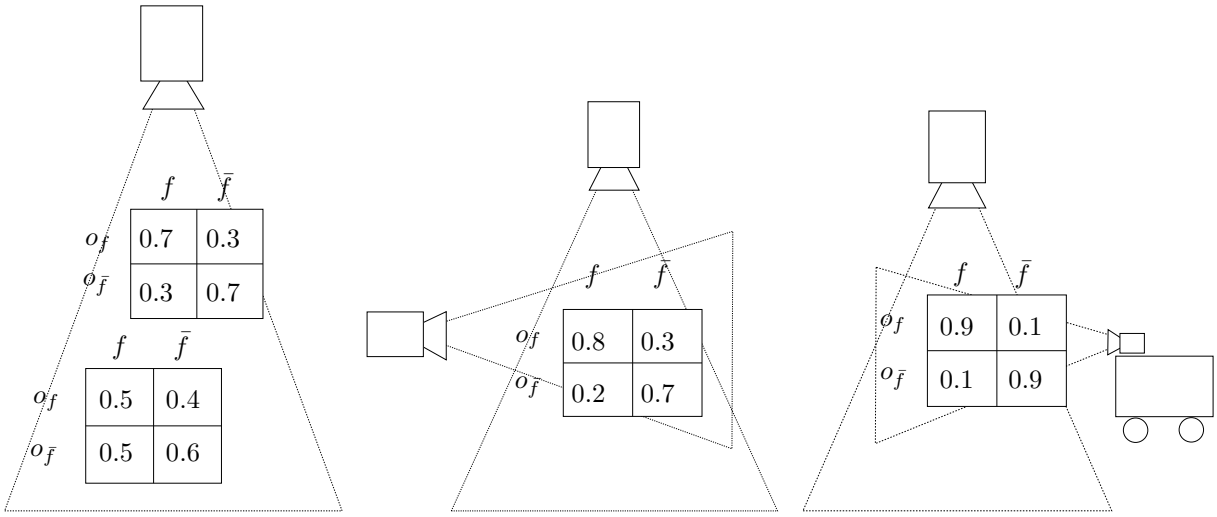
As mentioned, the problem is formalized in a set of variables, such that the transition model of each one is independent from the others, and we may exploit independence between them, namely exploring context-specific independence. For instance, in our problem we assume that the robot sensors are able to observe features only if person and robot are in the same location, thus feature detection is independent of robot location whenever they are far from each other. Therefore, the state space should encode the different problems we need to tackle in such task, namely: robot and person localization and feature detection. Assuming an environment with k persons where we want to track and classify m features, we define, then, four kind of state variables:

- Person location $P_i,\ 1 \leq i \leq k$;
- Robot location X;
- Feature $F_j,\ 1 \leq j \leq m$;
- Event classified $(C_j),\ 1 \leq j \leq m$.

In our task we may immediately define two problems: person tracking and robot navigation. Therefore, two localization variables are in order, P_i and X, which track positions of persons and robot in the environment. In localization and navigation the Markovian states are a representation of the environment, but as we are dealing with discrete POMDP models, locations must be represented by a discretization of the environment, for instance a topological map. Therefore the state space size of localization variables depends on how big is the environment and how detailed do we represent the map. We consider a division of the environment as a graph in a l-node map, where each node is the center of a cell, and every person or robot detected inside the cell is assigned to be at that particular node. After localize a person, the core of the task is to classify whether that person is associated with a number of features. Therefore, the model includes m variables for feature tracking. We assume that for each person we are looking for at least one feature, hence $m \geq k$. Each variable F_i intends to keep track on the belief of a particular feature for a particular person, and the expected behavior of the system is to use this belief to decide to classify people. Finally, m bookkeeping variables are also included. Those are boolean variables which keep track on which features have already been classified or not, intended to announce to the system whenever an event is positively classified, what in a surveillance task might be used as a signal that a possible threat has been detected. When such classification occurs these variables take the system to an absorbing state.

Transitions model the possible changes on each state at every timestep, taking into account the last known value of the state and the actual action. As transitions are independent for any state s the transition model becomes a product of transitions for each variable. Note on the Bayesian network that the value for each state variable at the next timestep do not depend on the value of other variables at the actual timestep. They are dependent only on its value and some also on the action. The transition probability at each state for this problem becomes:

$$\begin{aligned} p(s_{t+1}|s_t,a) \quad = \quad & p(P^1_{t+1}|P^1_t)\ldots\times\ldots p(P^k_{t+1}|P^k_t)\times \\ & \times p(X_{t+1}|X_t,a)\times \\ & \times p(F^1_{t+1}|F^1_t)\ldots\times\ldots p(F^m_{t+1}|F^m_t)\times \\ & \times p(C^1_{t+1}|C^1_t,a)\ldots\times\ldots p(C^m_{t+1}|C^m_t,a) \end{aligned} \tag{3.1}$$



(a) With 1 sensor. (b) With 2 sensors. (c) Fixed and mobile sensor.

Figure 3.4: Observation model examples. Each matrix show the observation probability for a single feature: $p(O^F|F)$.

Not all transitions depend on the action, as some changes in the environment are not controllable, namely those associated with person characteristics, person location and feature associated. We assume that a person follows a random motion pattern as we do not have knowledge of the person intended path, that is, at every timestep he might either want to move anywhere else or stay in the same place. Likewise, the feature or features associated with a particular person do not depend on actions, since they are a inner characteristic of people. Otherwise, robot and bookkeeping variables are depending on actions, beyond its previous values. As we will detail later, this model includes two types of actions, moving and classifying actions. Moving actions instruct the robot to move and classifying actions announce the classification of an event. The latter is a deterministic transition, since bookkeeping variables are considered binary. When the system classifies a particular feature the corresponding bookkeeping variable switches to keep track on classifications. Moving actions are the type of actions used in navigation tasks, which induct the robot to navigate between two points. Depending on the distance between nodes, robot velocity, and the interval between controller decision, there is a probability that the robot reaches the final destination for that move or is still on the way at the next timestep. Moreover, this actions may suffer from external and unexpected influences, such as unexpected obstacles in the way or misaligned wheels which lead the robot to end up in a different position than the destination.

3.3.2 Observations and observation model

To choose a suitable set of observations we should take into account the purpose of adding observations and how each one will be useful in state identification. As the set of states have been decoupled in a set of variable states, independent from each other, makes sense to keep a set of observations to provide identification of each state variable separately, except for bookkeeping variables, since those are deterministic. Therefore, we consider a set of three observations variables:

- Person location $O_i^P,\ 1 \leq i \leq k$;
- Robot location O^X;
- Feature observation $O_i^F,\ 1 \leq i \leq m$.

Each location observation follows the same environment description as state variables.Therefore, each observation $o_i^p \in O_i^P$ and $o^x \in O^X$ assigns an observation of a person or a robot to the closest position

$p_k \in P_K$ and $x \in X$ in a topological map. Feature observation indicates at each timestep if a particular feature has been detected associated with a particular person.

Observations are obtained from sensory information in the system, which collect the needed information to indicate positions and features. However, in real world there is no perfect sensor due to noise and changing in environment conditions. The observation model gives the controller a representation of the error probability for each observation. In our problem location observations depend only on actual position of person or robot. For instance, if a person is in a particular node, there is a given probability of be detected in the correct node, a lower probability of be detected in a neighbour node, and an even lower probability of be detected in a farther node.

Nevertheless, the key for cooperative perception in this model lies in the feature observation model, as probability for feature observation will differ with respect to person and robot location. The higher the probability of a correct observation, less uncertainty the system has regarding classifying features. Besides environment conditions, such probabilities differ with distance to sensors, the number of sensors which are pointed to a particular location, and if a person is on the field of view of a mobile sensor. We consider that a mobile sensor, typically, has a better accuracy than fixed sensors. Besides having a smaller field of view and thus when it is looking towards a person and analyzing features the distance between them is shorter, mobile and fixed sensors do not necessarily have to be of the same type. For instance, we may think of a scenario with a fixed camera network and robots with laser range finders to detect persons. Note that for economical reasons often when developing such systems we may need to choose where to include better sensors (more expensive) and worst sensors (cheaper), and clearly this model leads the developer to include better sensors in the robots. Therefore, the system should act in a way to decrease uncertainty, and move the robot when detection accuracy for actual person location is low. Figure 3.4 formalizes those concepts, already mentioned in the problem definition. Each matrix show the probability of correct observations, false positives and false negatives for a particular feature F_i. If a person is detected close to a sensor the probability of getting a correct positive $p(O^F = o^f | F = f)$ or negative observation $p(O^F = o^{\bar{f}} | F = \bar{f})$ is higher than when the person is farther. The opposite occurs with false positive and negative detections, $p(O^F = o^f | F = \bar{f})$ and $p(O^F = o^{\bar{f}} | F = f)$. Comparing with the probabilities of one sensor, with two or more sensors looking at the same node naturally that uncertainty decreases, Figure 3.4b. In particular, when one of those happens to be a mobile sensor, Figure 3.4c, then, due to its better accuracy, the error rates will decrease even more. This is a very important factor on decision making and planning, as observations, and, consequently, belief over a feature will depend much on person location and if the system has information from mobile sensors. This will lead the system to act in such a way to move robots towards areas where fixed sensors cannot provide high accuracy, in which its presence is more valuable.

3.3.3 Actions

In this model, we may distinguish between two different types of available actions:

- Moving actions, A_{move};
- Classifying actions, $A_{\text{classify i}},\ 1 \leq i \leq m$.

The first type of actions, are characterized by an effect on the physical world, that for instance move the robot. By the problem definition, the only physical part on the environment we can control is the robot. Therefore, makes sense for navigational tasks to have an action which instructs it to move whenever needed. Besides, this type of action may include additional physical effects, like adjusting robot's sensors to get better information at a particular location, for instance rotate a sensor, move it upward or downward, etc. Additionally

the robot might influence other elements in the system, for instance it might tell the camera network to run a different vision algorithm, or control some of the sensor's parameters, depending on how the robot observes the world.

Classifying actions, on the other hand, do not have physical effects in the environment, but instead announce that a particular event f has been classified. This ensures that, instead of classifying on the run, the system clearly separates actions which are intended to improve accuracy to others which encode the final goal.

Firstly, we had introduced two types of classifying actions for each event. One announces that the event had been detected and positively classified, the other announces the opposite, that a particular event had not been detected in the detected person and therefore that person should be discarded has a possible threat. This procedure has been discarded to maintain only the action which announces a positive classification, so the system assumes all the time that the event has not detected until it is classified. That helps to decrease the rate of false negative classifications, as in surveillance tasks it is more important not to discard possible threats.

We do not also directly include sensing actions in our model as our primary goal is not to reason over sensing or not in general, but rather in mobile sensing. Specially in the NRS we consider the focus is on cooperative environment perception, therefore, we should consider that the sensor network keeps continuously observing the environment, but the observation model changes with the position of the sensors. Since only mobile sensors are able to change its position, the focus of the model, and the focus of the actions we made available, are in moving mobile sensors in the environment. We do not consider moving actions to be directly sensing actions because on those actions we do not deal with the problem of choosing to observe a particular event with a particular sensor but rather the system tries to cooperate all the sensing systems together to obtain the best observation at the minimal cost.

3.3.4 Reward

The reward model encodes the general goal of the problem, which is to classify an event as being of interest or not. A POMDP approach provides a way to tackle this, since we include a variable to track the presence of a particular feature F_i, which indicates whether it is present or not. The system maintains a belief $b_t(F_f)$ on this variable indicating how sure it is about its presence. Therefore, in a bayesian network the reward depends directly on feature variables, but not on location variables, although indirectly they affect the belief due to its influence on feature observation model.

The model is built in a way that the system receives a reward signal at each action. To moving actions we simply assign a small negative reward, representing costs associated with moving robots, like energy spent, for instance. This influences the planner to minimize robot movement to its minimum, ensuring the robot will move only if strictly needed and that it chooses the smallest path. Classifying actions return a reward according to the following model: we assign a reward $r_f > 0$ when it indicates an event of interest and a penalty $r_{\bar{f}} < 0$ otherwise. That is, the immediate reward that the planner receives when classifying an event is $r_f \cdot b_t(F = f) + r_{\bar{f}} \cdot b_t(F = \bar{f})$. Furthermore, each feature can be classified only once, by using bookkeeping variables. Initially, each bookkeeping variable C_i has value *not yet classified*, and once the respective classify action is executed it changes to a *classified* value. From that moment on, the system receives a high negative reward if it classifies that feature again. This guides the system to get a high certainty with respect to detected features before classify them in order to get a higher reward.

Moreover, in the reward model we may encode priorities of different kind. Depending on each particular case, we may easily imagine some problems where it is important to establish priorities in the environment. From priorities between different areas, between features, between persons, etc. Imagine the case of priorities

between different areas. In such scenarios with different priorities, we may want the robot to check a high priority area, even if it has a lower event detection uncertainty, or when the robot needs to check on two different areas, it might want to go first to the higher priority area, even if its farther than others. Also, when a system needs to detect and classify more than one feature we may want to establish priorities between features. For instance, it might be more important for a surveillance task to positively detect faces which are not familiar with the environment, than to detect any other particular feature. Priorities are modeled in the reward model by giving a higher reward for positive classification associated with the respective priorities. For instance, in the case of areas oh higher importance, the reward model gives a higher reward for those areas, or in the case of more important features, the reward gives a higher reward for classification of those features than of other. When there are priorities in locations, the reward model will also depend explicitly on the person location.

A different approach has been followed initially, which considered the same kind of reward for classification as previously explained, but further explicitly required that person and robot were in the same location. This approach has been discarded and replaced by the previous presented model. Although this first approach also included a cooperative behavior, as we could merge both observations to improve accuracy on the system, it would not provide the cooperative behavior we are expecting. Than, we explicitly demanded the robot to always approach the position of the person, and the POMDP framework would be useful only for belief tracking, but would not bring many advantages in decision-making.

We did not consider using reward models based on the belief entropy, which would allow us to define reward models directly over beliefs instead of states, as in such cases we leave the framework of piecewise linear and convex value functions, in which many algorithms have proven to perform well, namely PERSEUS, the algorithm we apply in our models.

Chapter 4

Experiments

In order to validate the proposed POMDP model we performed a series of experiments. We propose to handle a surveillance task where the event of interest is to detect persons wearing a red shirt. The experiments took place at our lab, LRM (Laboratório de Robótica Móvel), at the Institute for Systems and Robotics at the 8th floor of the North Tower at Instituto Superior Técnico, where we have available a camera network mounted on the ceiling of the lab and mobile robots (Pioneer 3-AT) with on-board cameras and laser range finders [39].

In this chapter, we will first go through the model and its realization for this particular task and environment followed by a description of the hardware and software we needed to setup the experiments. Finally, we will present experimental results and discuss how our goals applied in this task.

4.1 Model

In Chapter 3 we described the POMDP framework for a general event detection task. Here we specialize that model for a specific task. We discretize part of the lab map into a 8-node map represented as a graph, which is used for location. Initially, we were expecting to use the entire lab space, but existing POMDP solvers are still limited in the state space they can cope with in an decent computational time, therefore we had to limit the space for the experiments and decrease the number of nodes, and consequently, the size of state and observation space. Each node defines a cell, which corresponds to the area of the lab where the distance to a particular node is less than to any other, as shown in Figure 4.1. In a discretized map, all positions in a cell are assigned to the corresponding node. Therefore, all state and observation variables which are used for location purposes assign a position in the map to the closest node ($p_i \in \{node01,\ node02,\ \ldots,\ node08\}$; $x \in \{node01,\ node02,\ \ldots,\ node08\}$). Since in this problem we are interested only in distinguish between persons with red shirt or not, we include one feature variable *color* for each person with two possible values: $f_1 \in \{\textit{red},\ \textit{other}\}$.

The observation model for color is obtained by off-line tests in which we collect detection data of a person with a known shirt color walking randomly in the environment. The color detection data is stored and summarized per node, giving the error rate and uncertainty in color detection per node. The resulting observation model is shown in Figure 4.2. Note that each matrix is located at the respective node location. Alongside, we plot all cameras used in the lab, with its real location and direction, and the observation model used for robot camera. Note that in the field of view of robot's camera its observation model overlaps with fixed cameras observation model.

Besides the classification action, there are four moving actions: *moveLeft*, *moveUp*, *moveRight* and *move-*

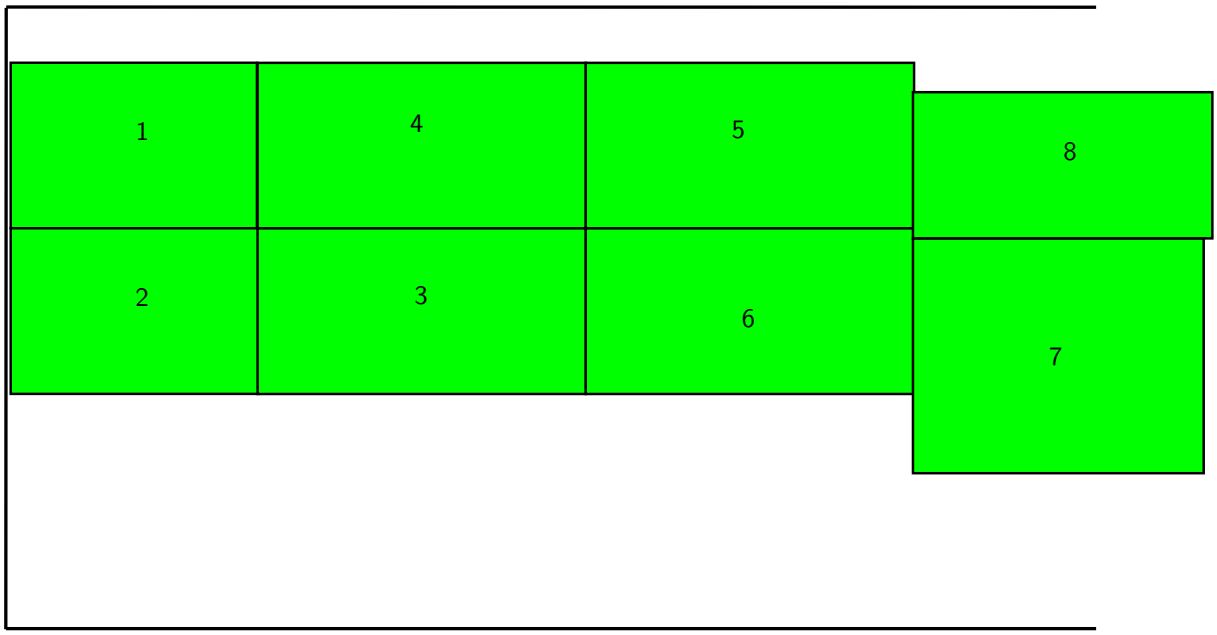


Figure 4.1: Map of the lab, each square represents one cell, each cell center is one node in the topological graph-like map.

Down. The intended outcome for each action is for the robot to move to the closest neighbour node in the respective direction, assuming those four directions are aligned with cardinal directions. In this manner, we allow the robot, at any moment, to move from its actual position to any of its neighbour nodes. We do not model the robot's heading, leaving it and navigation control to off-the-shelf software, therefore, we do not need to include turning actions in the model, but rather we care only about what is the final destination of the robot at each action. A positive reward $r_f = 10$ is given when the person is correctly classified and $r_{\bar{f}} = -10$ otherwise, with a penalty cost of -0.1 for each robot movement. The discount rate is set to a high value, $\gamma = 0.99$.

4.2 Experimental Setup

Taking into account the typical interaction of a POMDP agent with the environment there are 3 main blocks to be built in such a way that they may be able to communicate between them: the observation, decision and acting systems.

The observation system uses the available sensors and through manipulation of the sensors measures inform the decision system how is the environment being observed at the current moment. The decision system should keep track of the information from the observation system and the actions executed, knowing the observation and transition models it keeps a belief for the probability of being in each state. From this information the decision system should decide which action to execute as part of a policy with the goal of getting the biggest reward possible. The acting system consist in receive from the decision system which action to execute and control the hardware in order to follow that order. Typically, in robot navigation, the action will be some kind of local navigation, specifically in our case the actions will be moving the robot from the actual node to a neighbour node. The focus of this system is on control the robot's hardware. Finally, all the systems must be implemented and be able to communicate in such a way that the information flows over all the system. The software used on the implementation of the system was:

- OpenCV (Open Source Computer Vision) library

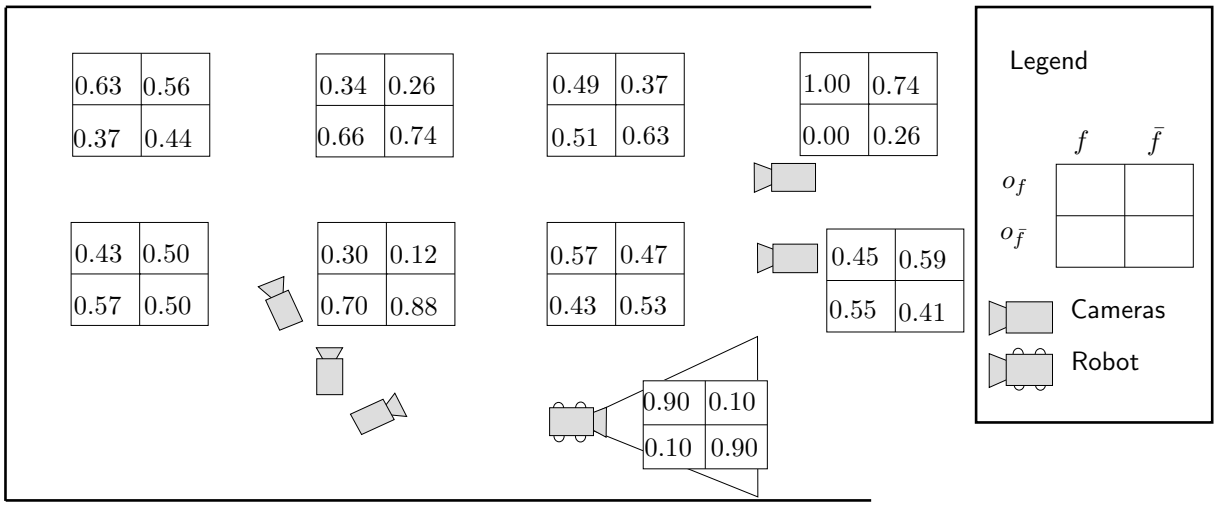


Figure 4.2: Learned observation model for fixed and mobile cameras. Each matrix is positioned at each node in the map, except the one for the mobile camera, which show the observation at any location if $X = P_i, 1 \leq i \leq k$.

- CARMEN (Carnegie Mellon Robot Navigation Toolkit)
- `MeRMaID` (Multiple Robot Middleware for Intelligent Decision-making)

The vision systems have been developed making use of the OpenCV library [40], an open source computer vision library developed by Intel. To handle with robot localization and navigation we used CARMEN [41], an open-source source collection of software, which was originally designed to provide navigation and localization support for a wide variety of robotic platforms. CARMEN has a modular architecture and was designed to provide basic navigation primitives including: base and sensor control, logging, obstacle avoidance, localization, path planning, and mapping. An advantage of CARMEN is that it is easy to receive and send information to its modules. This facilitates the integration with other modules, namely for the observation system to receive information from robot localization and the decision system to send the target destination to robot navigation. For the localization and navigation tasks, CARMEN requires a map which can either be built manually or by using sensorial data. `MeRMaID` [42] is a middleware original developed for the ISR's RoboCup Middle Size League robot team and later used for integration purposes on URUS project [39]. In the same sense, we use it to integrate all the subsystems and to send information along the system.

4.2.1 Observation system

We have available two types of sensors, cameras (placed in the ceiling and on the robot) or laser range finder (onboard the robot). Besides we can also obtain information from robot's odometry. In our work, cameras are used for person and color detection while laser combined with odometry is used for robot localization.

Detect persons

Detection of persons is performed by real-time algorithms, applied to the images provided by the fixed camera network. We use the algorithms developed in [43] integrated in the URUS project. Those algorithms detect moving objects using the LOTS adaptive background differencing algorithm. It returns a list of image blobs from where can be extracted features, namely in our case, whether there's a person being tracked and what is his position in the environment. LOTS algorithm starts by initializing two background models $B1$ and $B2$ in order to have a lower and higher value of the pixel, contemplating this way the variations of a non-target pixel (due to noise, light conditions, etc). Two thresholds are initialized, a low threshold T_L, bigger than the

Figure 4.3: Bounding box of a person. The box represents the area where the person is detected. From the center bottom of the box, it is possible to extract the position of the person in the world plane, by homography

difference between the two backgrounds, and a high T_H, obtained by adding a constant to T_L. Those values are used in the detection algorithm, summarized as follows:

1. Create D, containing the least differences between the new image I, and the backgrounds.
2. Create binary images D_1 and D_2, where the active pixels are the pixels of D which are higher than the thresholds T_L and T_H respectively.
3. An algorithm is performed over D_1 and D_2, which labels every pixel as *detected*, *not detected* or *false detection*. The *detected* pixels with connectivity 4 are labeled as targets.
4. Backgrounds are adapted:

$$B_t^{t+1}(\phi) = \begin{cases} (1-\alpha^{'})B_i^t(\phi) + \alpha^{'} I^t(\phi) & \phi \in T^t \\ (1-\alpha)B_i^t(\phi) + \alpha I^t(\phi) & \phi \in N^t \end{cases} \tag{4.1}$$

   where $\phi \in T^t$ is a pixel that is from a target and $\phi \in N^t$ is a false or not detected pixel.

5. The threshold is updated for each pixel:

$$T_L^{t+1}(\phi) = \begin{cases} T_L^t(\phi) + 10 & \phi \in false\ detection \\ T_L^t(\phi) - 1 & \phi \in not\ detected \\ T_L^t(\phi) & \phi \in detected \end{cases} \tag{4.2}$$

The mapping between an image point p_i and a point in the ground plane p_w is done by a homography matrix H:

$$p_w = Hp_i, \tag{4.3}$$

where

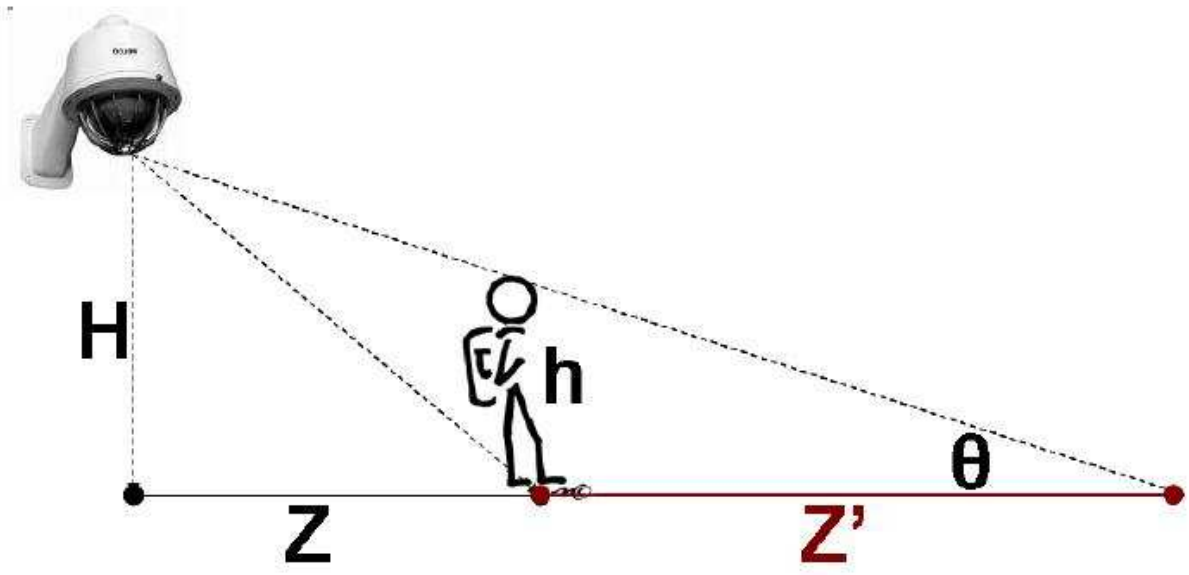


Figure 4.4: Target height computation schematic. Knowing the position of the camera in the world plane, the position of the target, and the projection of the top center of the bounding box in the world plane, it is possible to compute the actual height of the target, excluding targets which height do not correspond to a person..

$$p_w = \begin{bmatrix} kx_w \\ ky_w \\ 1 \end{bmatrix}, p_i = \begin{bmatrix} kx_i \\ ky_i \\ 1 \end{bmatrix}, H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}. \tag{4.4}$$

Finally, the position in the world plane is extracted:

$$\begin{aligned} x_w &= \frac{h_{11}x_i + h_{12}y_i + h_{13}}{h_{31}x_i + h_{32}y_i + h_{33}}, \\ y_w &= \frac{h_{21}x_i + h_{22}y_i + h_{23}}{h_{31}x_i + h_{32}y_i + h_{33}}. \end{aligned} \tag{4.5}$$

The detection algorithm returns bounding boxes corresponding to the events of interest detected (Figure 4.3).

The next step consists of extracting some features of these regions in order to identify if the object detected is a person. The classifier works as a decision tree which takes into account four features:

- Ground plane position: the middle bottom point of the bounding box is mapped to the ground plane by the homography. If this position is outside the limits of the room (pre-specified) the target is discarded.

- Number of detected pixels: every target smaller than a pre-defined minimum size of the detection is rejected.

- Target height: by homography, it is possible to compute the projection of the top and the bottom of the bounding box in the ground plane. Knowing this and the camera position, the target height can be computed by (4.5), which comes from the sketch of Figure 4.4.

$$\begin{aligned} \theta &= \arctan\left(\frac{H}{Z + Z'}\right) \\ h &= Z' \tan(\theta) \end{aligned} \tag{4.6}$$

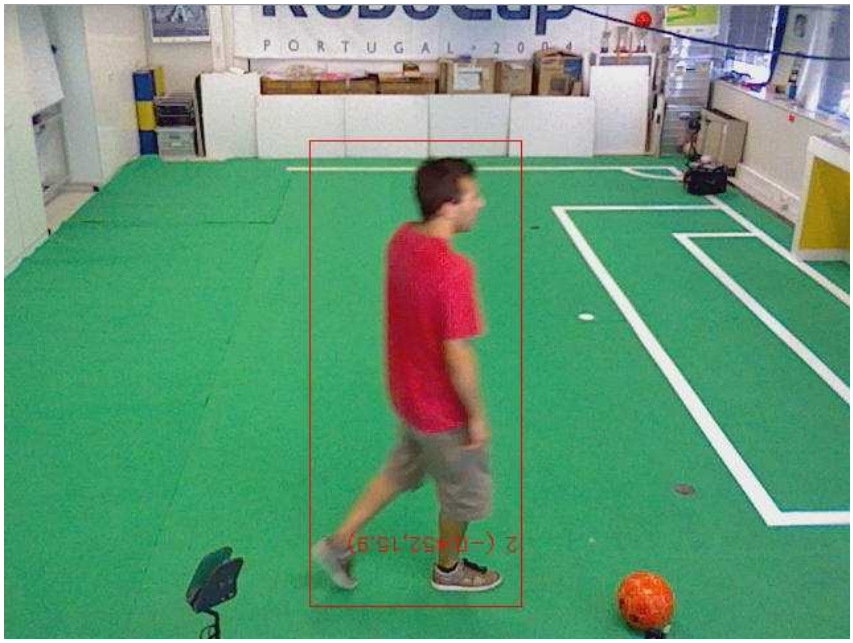


Figure 4.5: Original Image with bounding box

- Occupancy ratio: this ratio is computed by comparing the total number of pixels inside the bounding box and the actual number of detected pixels in the region. There is an absolute maximum for this parameter when the detected object is a person, therefore the detection is accepted if the occupancy ratio is below 35%.

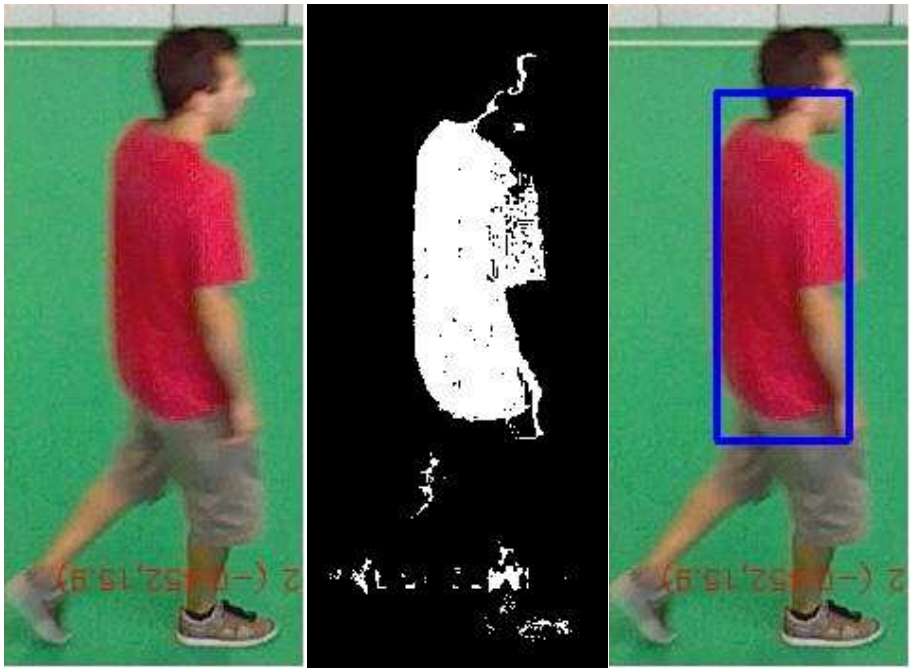
In summary, a detection is accepted as being a person if its world position is inside an expected region, has a minimum number of pixels, its height is within the expected for a person and has an occupancy ratio below the maximum limit.

Detect color

When it comes to detect color we take into account that the person detection algorithm provides the bounding box of the person (Figure 4.5). Since we want to detect the person's shirt color, makes sense to use this information in some way as the person's image is inside the box. In this manner an algorithm as been implemented which looks for color blobs only inside the bounding box. The idea is to search for a big enough red color blob inside the box (Figure 4.6a), if we find it, we assume that the person is wearing a red shirt. The algorithm can be summary explained as doing some filtering to find the pixels which are within some pre-defined thresholds and find the biggest blob area. It is considered a positive detection if this area is bigger than some pre-defined minimum area.

The algorithm to search for color blob is divided in five steps:

- Separate the channels (red, green, blue).
- Equalize the histogram of each channel, in order to increase the global contrast of the image. This operator allows for low contrast areas to gain higher contrast without affecting the global contrast, by distributing the intensity values more uniformly on the histogram. This permits better distinguish between lower and higher intensity areas, which allows for better distinguish between colors.
- Filter each channel based on pre-defined thresholds. This step returns one binary image BI for each channel, where the minimum value is assigned to the pixels ϕ which are outside the threshold limit and



(a) original (b) filtered (c) result

Figure 4.6: Illustration of the steps before computing color blob. (a) Cut of the original image, representing the region of interest where the person is, defined by the bounding box. (b) Final binary filtered image obtained by filtering the 3 channels of the original image with the pre-defined thresholds. (c) Final blob detection, found from the biggest contour on the filtered image.

the maximum value to the pixels inside the limits.

$$BI(\phi) = \begin{cases} 255 & I(\phi) \in [T_L, T_H] \\ 0 & otherwise \end{cases} \tag{4.7}$$

- Compute the conjunction of the three binary images, in order to obtain a final binary image where the pixels within the threshold limits are highlighted (Figure 4.6b).

$$I_F(\phi) = I_R(\phi) \cap I_B(\phi) \cap I_G(\phi) \tag{4.8}$$

- From this final binary image, find the biggest blob contour, if it is higher than some pre-defined minimum value it is considered a positive detection (Figure 4.6c).

A note on robot's camera, in which we use the same color detection algorithm, but due to its special characteristics, we are not able to run a person detection algorithm (at least not a background detection algorithm as we use for the fixed camera network) and as such we assume some simplifications. We increase the minimum value for a detection to be considered positive, as we assume that the robot will only observe person's within approximately between 1 and 2 meters away, and we point the camera in such an angle (approximately 45 degrees) so we consider that when a person of average height is in front of the robot, the camera points to the center of the shirt, and is able to successfully detect a color blob. Moreover, we assume that the robot will not face any red object in the environment on its field of view which may be falsely detected as a person with a red shirt.

Robot localization

As mentioned, for robot localization we use CARMEN toolkit, specifically for the localization task CARMEN has a *localize* module which implements a variation of Monte Carlo Localization (MCL) algorithm, Mixture MCL [44]. MCL represents the robot's belief by a set of particles, drawn according to the posterior distribution over robot poses. Mixture-MCL merges plain MCL with dual MCL, which inverts MCL's sampling process. While regular MCL first estimates a new pose using odometry, then uses sensor measurements to adjust the importance of the guess, dual MCL first estimates a new pose using the sensor measurements and then uses odometry to adjust the importance of the guess, according to the previous belief and odometry data. Mixture MCL combines the advantages of both methods (while regular MCL works best with sensors with some amount of noise, dual MCL is best for use with accurate sensors) and provides a robust robot localization algorithm.

4.2.2 Decision system

The decision system implements the solution of the POMDP model described in Chapter 3. Also as mentioned in Section 2.5 we used the symbolicPerseus POMDP solver. It computes off-line an approximate solution for the POMDP problem, decreasing the computational requirements on the on-line execution of the policy. The off-line computation is based on a straightforward implementation of the source code (MATLAB/Java) available online [1]. It outputs a set of alpha-vectors and the optimal policy for each vector.

The online computation reduces to update the belief state from the transitions and observations, and therefore compute the value for each alpha-vector, the action associated with the highest value vector is the action to execute next.

4.2.3 Acting system

The acting system is basically based on a navigation system. It receives a target destination from the decision system and drives the robot to the desired location. As mentioned, this system is implemented with CARMEN, namely by its *navigation* module. It implements Konolige's local gradient descent planner [45], which uses a navigation function to generate a gradient field representing the lowest-cost path to the waypoint goal. This method is efficient enough to be used for real-time robot control, even whilst using large maps.

An important feature of CARMEN's *navigation module* is also the integration of obstacle avoidance, which allows for more reliable navigation paths, especially when unmapped obstacles appear in the environment which causes the module to compute alternative paths.

4.2.4 Subsystem Integration

In the end, all the subsystems must be assembled together, must be run in real time, be capable to receive and send information, and to process and interpret the information. Figure 4.7 presents the communication schematic on our system. The architecture is based on the service-oriented middleware used in `ISRobotNet` testbed [39], based on `MeRMaID`.

The middleware provides a better integration approach and standardization in communications within the system. A service-oriented architecture means that each system component runs a service, which has some kind of internal dynamics. Moreover, services need to communicate with each other in order to transmit information over the system. In this manner, two types of interaction are defined: service request and data feed. A Service Request is the preferred mechanism for single-shot interactions, where a *Request Service* sends a request to a *Target Service*, which processes the request and send a reply back. A Data Feed is a data push

[1] http://www.cs.uwaterloo.ca/~ppoupart/software.html#symbolic-perseus

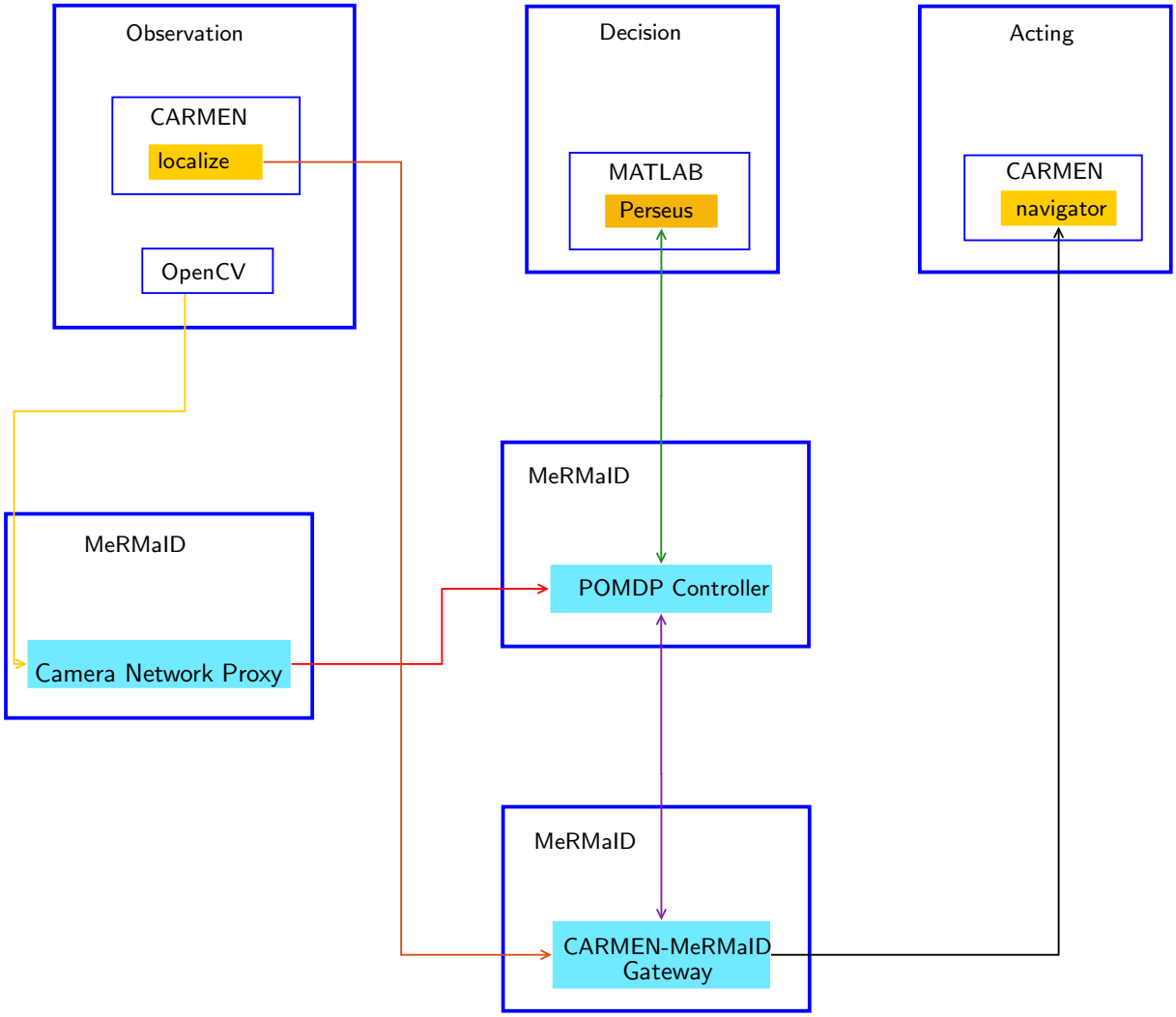


Figure 4.7: Integration schematic of all the subsystems with software used in each module. Arrows represent information flow.

mechanism through which data generated by a *Writer Service* is made available to the rest of the system, so any other service interested may read it. Both interactions are used in our system depending on specific context of each service.

The observation system is divided into camera detection and robot localization. Each camera runs a person and color detection algorithm in real time. Whenever a person is detected, it generates a pack of information about the detection (coordinates of the person position and whether its color is red or other), and pushes it to the system through a proxy. Robot localization runs in CARMEN and also sends a bottle with the actual position coordinates of the robot. For this, a gateway has been implemented, in order to establish communication between CARMEN modules and MeRMaID. CARMEN provides helper functions to subscribe for messages which allows for an exterior entity to get data from CARMEN modules. Namely, for observations, we are interested in getting the actual position of the robot and the respective message from the `localize` module.

A similar process is used for the navigation task, but the other way around, that is, the target of the next action is sent by the decision system to the gateway by a Data Feed. The gateway then follows the message in CARMEN's format to the `navigator` module and toggles the navigator in autonomous motion mode.

The decision system runs a `POMDPController` service which is connected to Data Feeds from the observations system and creates a Data Feed to send actions to the acting system.

4.3 Experimental results

We have conducted a series of experiments in order to show that our model applies to the surveillance task we defined and is feasible in a real robot scenario. We present, then, some of those final experiments, showing different behaviors of the system and with different world configurations. For each experiment, we present 3 figures with, respectively, the path of the robot during the experiment, alongside person position, evolution of cumulative reward over time, and evolution of belief over time. The cumulative reward $\sum_t \gamma^t r_t$ shows the total sum of rewards received at each time step weighted with the discount factor over time. The belief $b_t(F = \textit{red})$ shows the belief for feature variables, in this case it indicates how sure the system is over time that the shirt color of detected persons is red. Although the system decides based on a joint belief over all variables, since we are in the domain of factored POMDPs, it is possible to separate beliefs over each variable, and thus, we are more interested in observing the evolution of the belief over feature variables, since it is crucial for decision making in this model. In every experiment the belief is initially set to a uniform distribution, as initially we may expect any event to occur.

All experiments take place like represented in Figure 4.8. Nothing happens in the system until a person is detected. When that happens, the decision system will decide, depending on which observations it receives (person detected position and shirt color), and the robot will follow whatever instructions it receives. In the example case, a person without a red shirt is detected, however the system decides that the place is uncertain enough to send the robot there to improve accuracy, and the robot will move to the same position as the person is being detected.

A number of alternatives heuristics have been proposed [46], mostly acting in the underlying MDP of the general POMDP model. We did not compare our POMDP methods with those solutions, such as Q_{MDP} [47], as policies resulting from those methods are myopic regarding uncertainty, and do not reason over observations when planning. That is crucial in ACP problems, as we expect the system to follow policies which take actions in order to reduce general uncertainty in the system, and the reward models are also built according to such property. For instance, if we follow MDP-based heuristics in our event detections system, the final policy will always follow a single action, which instructs the robot not to move, as in the point of view of such methods, the system will gain nothing in moving the robot towards a detected person, but will not either classify a feature since it will not take into account the observation model.

In our event detection system we start with a uniform distribution over features belief and we need to plan in order to break such uniformity and make a decision on whether to classify an event or not. If we follow MDP-based heuristics we will be constructing policies without taking into account the observation model and the final policy will always

4.3.1 Experiment A

In the first experiments we considered a scenario with only 1 person. In experiment A a person is detected in a high red detection uncertainty area, while the robot is in the opposite side of the environment, considering only the 8-node map. In this case, the policy followed by the system is to move the robot across the room so with its camera the system confirms the information received from the rest of camera network. The evolution of belief (Figure 4.9c) and cumulative reward (Figure 4.9b) is consistent with the general system behavior. Since the uncertainty in red detection is high in the area where the person is detected, the belief increases slowly over time. Note that in this experiment, although the area has a high uncertainty, the camera network keeps observing *red*. However, that uncertainty means that when receiving an observation *red* there is a small difference in the probability that it actually is a correct observation or that it is a false positive, which explains the need of the system to get a better observation with the robot. Thus, as mentioned, while the robot is far

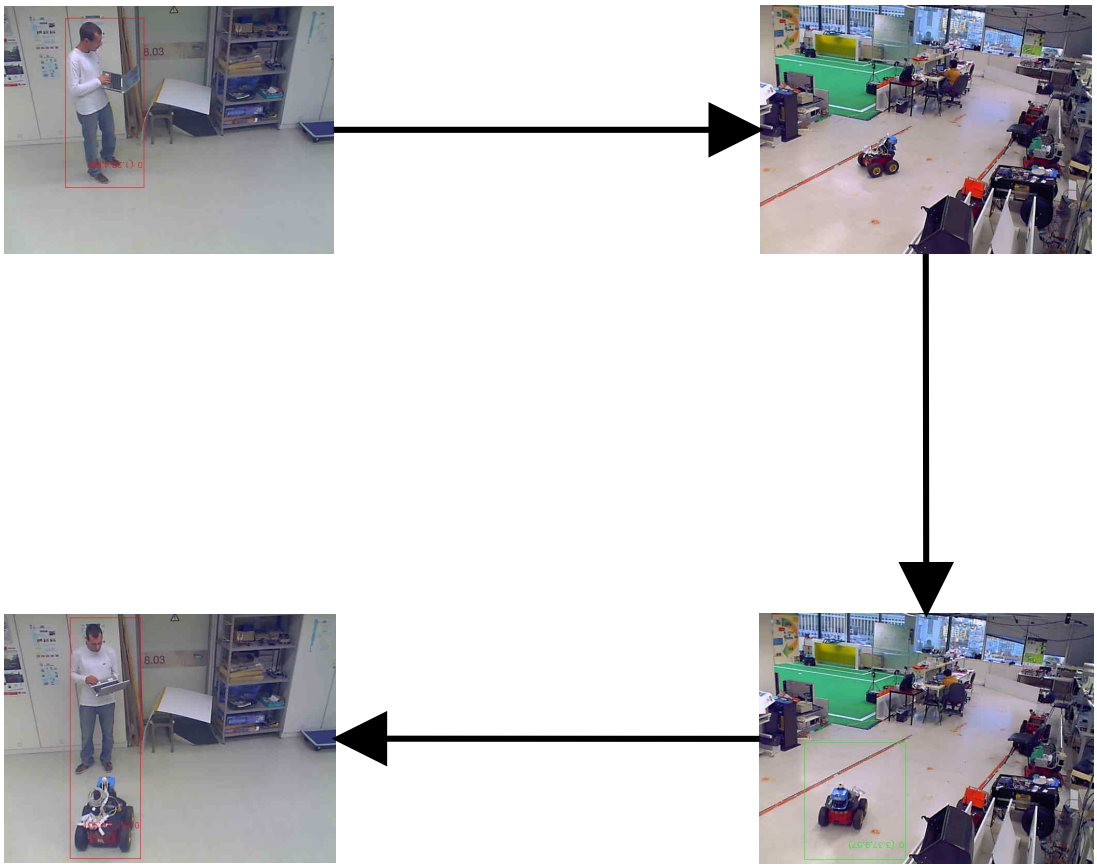


Figure 4.8: System behavior example, in which a person is detected and the system decides to send the robot near the person to check for features. This example shows the steps on the system behavior since person detection to eventual classification or not.

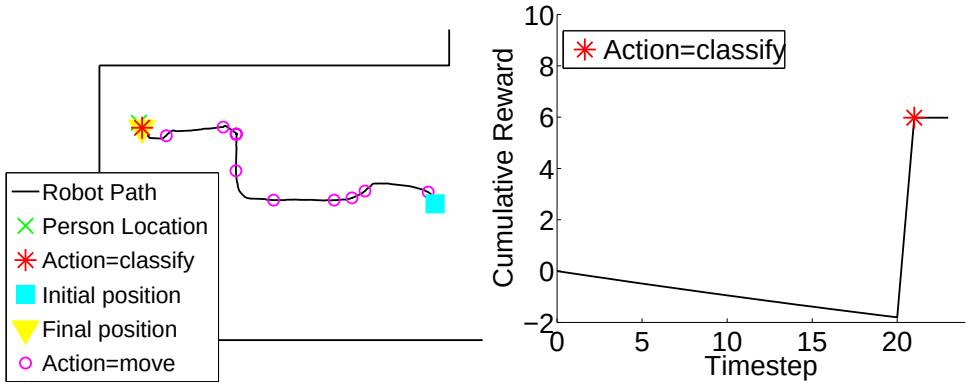
from the person the system only receives the information from the camera network and the belief increases slowly until finally the robot meets the person. At this point, at timestep 20, there is an increase in the slope of belief, showing that robot's camera is also detecting that the person is wearing a red shirt, and the belief gets to a point in which the system already has a low degree of uncertainty regarding to *red* detection, and decides to classify this event, when it calls a classifying action, at timestep 21. In the same time, when the person is classified, the system receives a high positive reward. In the mean time, while the robot was moving towards the person's position, the system kept collecting negative rewards due to robot movement cost, and therefore, at the moment when classification occurs, the cumulative reward is negative, and increases with the classification.

Mathematically, if we call the cumulative reward at timestep t, $CR(t)$, the cumulative reward at timestep 20 is the sum of rewards weighted with the discount factor over all timesteps. As the robot moves since the first timestep we have:

$$CR(t=20) = \sum_{t} \gamma^t r_t = \sum_{t=1}^{20} 0.99^t \cdot (-0.1) = -1.803 \tag{4.9}$$

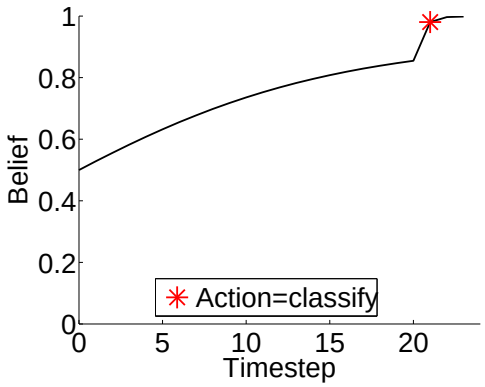
Then, when the robot is looking to the person the belief increases to $b_{21}(F_1 = \textit{red}) = 0.9804$ and the system gets an immediate reward for classifying of $r(t=21) = 10 \cdot 0.9804 + (-10) \cdot (1 - 0.9804) = 9.608$. Therefore, the final cumulative reward after classifying is:

$$CR(t=21) = -1.803 + 9.608 \cdot 0.99^{21} = 5.9769 \tag{4.10}$$



(a) Exp. A: Robot's path.

(b) Exp. A: $\sum_t \gamma^t r_t$.



(c) Exp. A: Belief $b_t(F_1 = \textit{red})$.

Figure 4.9: Exp. A: Experiment with 1 person, successful classification and robot moves.

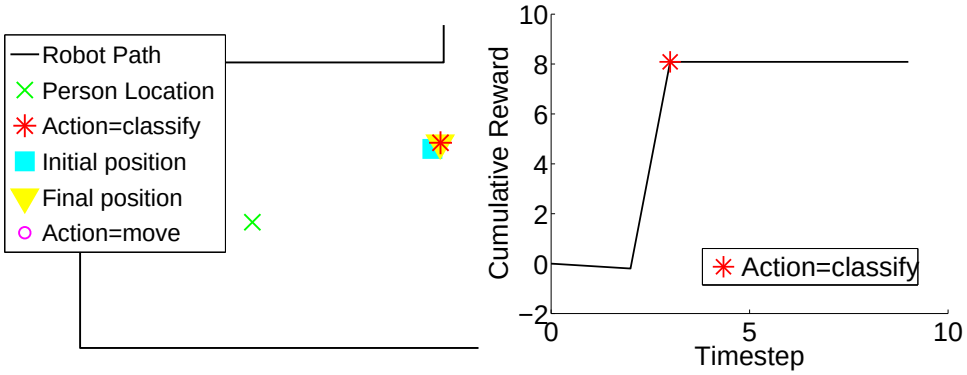
After classification bookkeeping variable C_1 switches to *classified*. From then on the system enters in an absorbing state, the robot will not move since there's nothing left to classify, and the systems keeps receiving a zero reward, stabilizing the cumulative reward value.

4.3.2 Experiment B

In this other experiment, unlike the first one, the person is detected in an area with less uncertainty regarding feature detection. This leads to a different behavior, as in this case the robot does not move from its initial location. The belief increases more rapidly than Experiment A, leading to a classification only within a few timesteps after the person being detected. Moreover, due to a quick classification, the system receives less negative rewards, because although the robot stays at the same location, in the initial timesteps the policy still is to move the robot closer to the person. However, the robot does not move as a classification is reached quickly. This smaller collection of negative rewards leads to a higher final cumulative reward. The cumulative reward at timestep 2, which is the timestep immediately before classification, is:

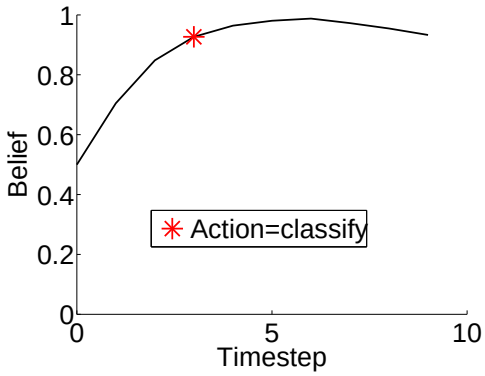
$$CR(t=2) = \sum_{t=1}^{2} 0.99^t \cdot (-0.1) = -0.197 \tag{4.11}$$

When the system classifies, at timestep 3, the belief $b_3(F_1 = \textit{red})$ is 0.9268, and therefore the system gets an immediate reward of $r(t=3) = 10 \cdot 0.9268 + (-10) \cdot (1 - 0.9268) = 8.536$. The final cumulative reward after classification is:



(a) Exp. B: Robot's path.

(b) Exp. B: $\sum_t \gamma^t r_t$.



(c) Exp. B: Belief $b_t(F_1 = \textit{red})$.

Figure 4.10: Exp. B: Experiments with 1 person, successfully classification and robot does not move.

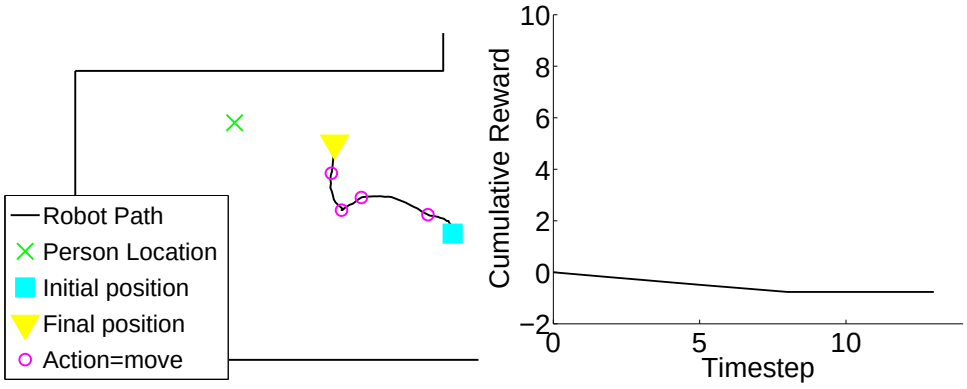
$$CR(t=3) = -0.197 + 8.536 \cdot 0.99^3 = 8.0855 \tag{4.12}$$

Note that although the immediate reward awarded by the system when classification occurs is lower than in Experiment A, as it classifies much quicker in this case, the final cumulative reward is higher, due to the discount factor, which penalizes rewards received in the long term.

The differences between these first two experiments show the purposes of our classifier system. A mobile sensor (the robot in the case) will only spend its resources if the system really needs its information to make an informed decision whether to classify the event or not.

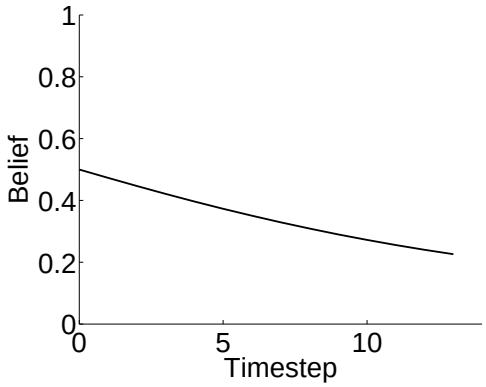
4.3.3 Experiment C

It is also important to guarantee the system is also accurate when no event took place. Experiment C shows what happens when a person not wearing a red shirt is detected. The person is detected in an area with some uncertainty and, in summary, the behavior of the system is to move the robot towards the person until the belief reaches a level of less uncertainty, whether by observation with robot's camera, or if observations from camera network are consistent enough to make a decision. We end up in the latter in this experiment, as we show in Figure 4.11. Like in every experiment, the belief $b_t(F = \textit{red})$ initially is 0.5 and the camera network observes *other* for variable *color*. As that area does not have a low degree of uncertainty regarding detection of *other* color, the belief drops slowly, while the system keeps receiving observations. Therefore, in the mean time, the robot moves towards the person in order to get a better observation with its camera. However, since the system keeps receiving the same information(*other*) from the camera network, the belief



(a) Exp. C: Robot's path.

(b) Exp. C: $\sum_t \gamma^t r_t$.



(c) Exp. C: Belief $b_t(F_1 = \textit{red})$.

Figure 4.11: Exp. C: Experiments with 1 person, no classification.

drops low enough for the system to decide that it is not worth it to move the robot anymore, and so it stops in the middle of the way.

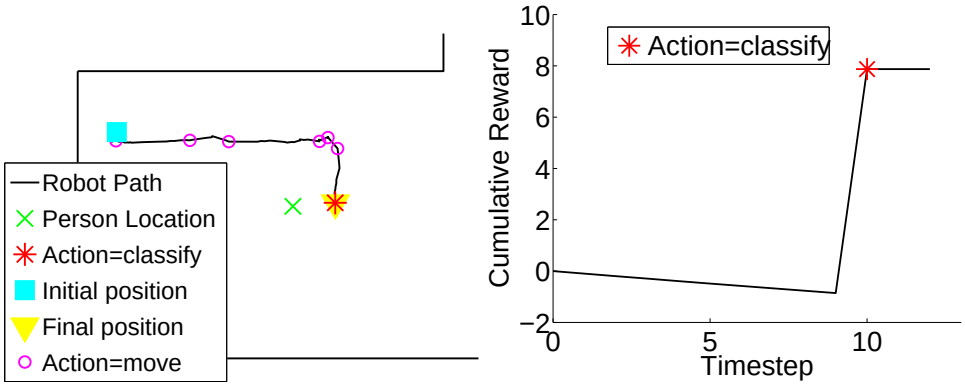
Since the robot is moving in the first timesteps, the system collects negative rewards, which are not compensated by a classification reward, and therefore, the final cumulative reward on this experiment is negative. The system keeps receiving negative rewards until timestep 8, when the robot stops. At this timestep the belief $b_8(F_1 = \textit{red})$ is 0.3089 and the final cumulative reward is:

$$CR(t = 8) = \sum_{t=1}^{8} 0.99^t \cdot (-0.1) = -0.7648 \tag{4.13}$$

4.3.4 Experiment D

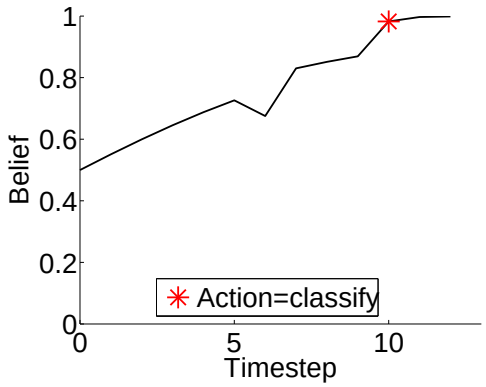
Experiment D shows a situation similar to Experiment A, in which a person is detected in a more uncertain area, such that the robot needs to check it. However, here we show a particular characteristic of this system, which is the capability to handle with incorrect observations. Note in this experiment the belief does not increase monotonously, since some false negative detections are received along with correct observations. Although this reduces the belief at that particular timestep, it does not influence the general behavior of the system, as the robot still goes to check on the detected person and eventually classifies.

The classification occurs when the belief reaches a value similar to Experiment A, $b_{10}(F_1 = \textit{red}) = 0.9825$ at timestep 10. Until then, the robot executes moving actions, which leads to a cumulative reward before classifying of:



(a) Exp. D: Robot's path.

(b) Exp. D: $\sum_t \gamma^t r_t$.



(c) Exp. D: Belief $b_t(F_1 = red)$.

Figure 4.12: Exp. D: Experiment with 1 person, successful classification and robot moves.

$$CR(t=9) = \sum_{t=1}^{9} 0.99^t \cdot (-0.1) = -0.8562 \tag{4.14}$$

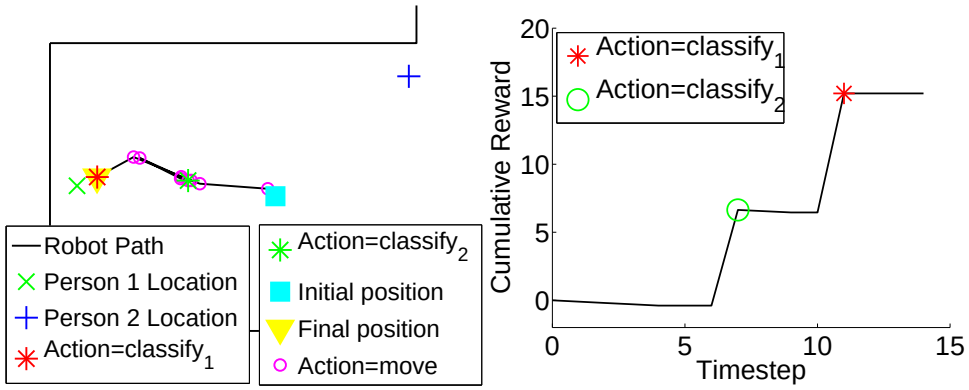
When classifies, the system receives an immediate reward of $r(t=10) = 10 \cdot 0.9825 + (-10) \cdot (1 - 0.9825) = 9.65$ which leads to a final cumulative reward of:

$$CR(t=10) = -0.8562 + 9.65 \cdot 0.99^{10} = 7.8711 \tag{4.15}$$

4.3.5 Experiment E

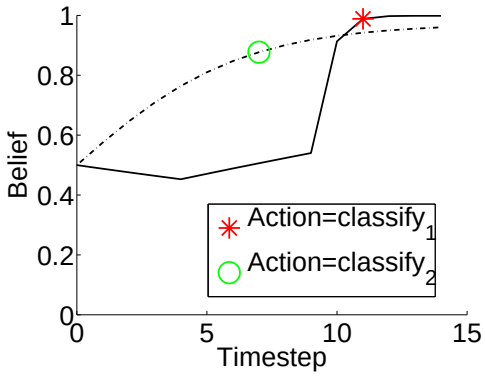
We did also considered more complex scenarios, namely when instead of a single person we consider to have 2 persons on the environment. Now, the system must reason whether to classify each one, and if the uncertainty is high and there is the need for the robot to check, in which order to do so.

In this experiment, each person is detected in areas with different uncertainties: person 2 in a low uncertainty area and person 1 in a region with higher uncertainty. The system behaves as expected, taking into account the difference between uncertainties in feature detection for both persons, the robot goes to check on person 2, and the system classifies person 1 without the need for the robot to check it. Note the evolution of both beliefs, $b_t(F_2 = red)$ increases faster than $b_t(F_1 = red)$, due to less uncertainty in the observation model for this location and because some false negatives detections are received in the first timesteps for person 1. Therefore, person 2 is classified first, at timestep 7, with a belief $b_7(F_2 = red) = 0.8772$, giving an immediate reward of $r(t=7) = 10 \cdot 0.8772 + (-10) \cdot (1 - 0.8772) = 7.544$. Before classification the system executes



(a) Exp. E: Robot's path.

(b) Exp. E: $\sum_t \gamma^t r_t$.



(c) Exp. E: Belief $b_t(F_1 = red)$.

Figure 4.13: Exp. E: Experiments with 2 persons, both classified.

some moving actions, ending with a cumulative reward at timestep 6 of $CR(6) = -0.3901$ and the cumulative reward after classifying person 2 is:

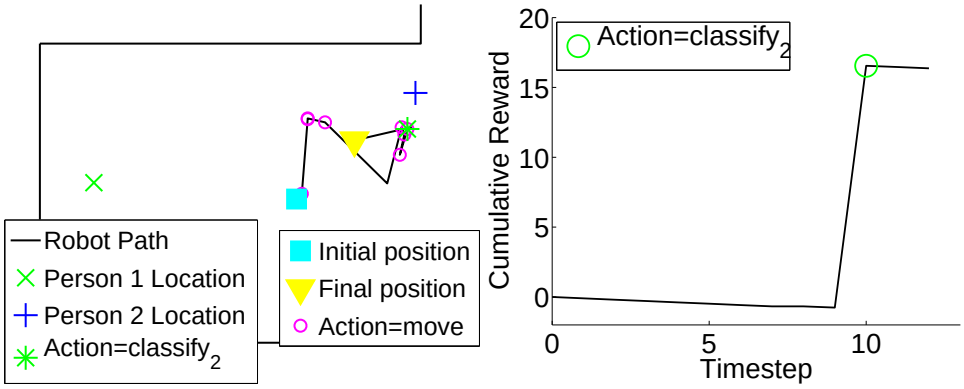
$$CR(t = 7) = -0.3901 + 7.544 \cdot 0.99^7 = 6.6414 \tag{4.16}$$

Afterwards, the robot continues moving because there is still one unclassified person, for which the system needs aid from robot's camera. Therefore, the cumulative reward drops until $CR(t = 10) = 6.457$. At timestep 11 the belief for shirt color of person 1 is $b_{11}(F_1 = red) = 0.9885$ and person 2 is classified as wearing a red shirt. The immediate reward for this classification is $r(t = 11) = 10 \cdot 0.9885 + (-10) \cdot (1 - 0.9885) = 9.77$. The final cumulative reward is:

$$CR(t = 11) = 6.457 + 9.77 \cdot 0.99^{11} = 15.2045 \tag{4.17}$$

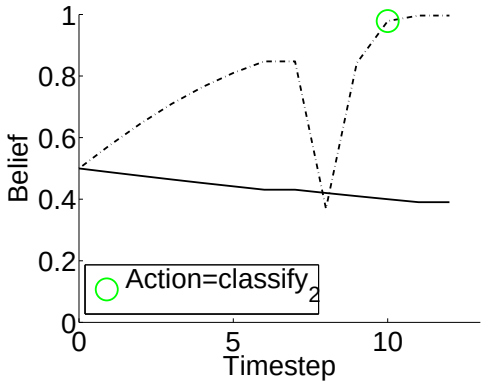
4.3.6 Experiment F

So far, all experiments have assumed equal priorities for all locations and people. Now, we will consider another interesting scenario, where we consider different priorities, namely with priority areas, that is, when there is one or more areas in the environment which are most important, and therefore require special attention. These priorities are encoded in the reward function. In this experiment, we considered the same reward model as described for previous experiments, but with one modification: we double the reward received when a person is detected and positively classified at node 8:



(a) Exp. F: Robot's path.

(b) Exp. F: $\sum_t \gamma^t r_t$.



(c) Exp. F: Belief $b_t(F_1 = red)$.

Figure 4.14: Exp. F: Experiments with 2 persons with different priorities.

$$r(F = red, P_i) = \begin{cases} 20 & \text{if } P_i = node\ 8 \\ 10 & \text{otherwise} \end{cases} \tag{4.18}$$

In this experiment, persons are detected in the same locations as Experiment E, with the particularity of person 2 be on node 8, exactly the priority area. Remember that person 2 is in a low uncertainty area and person 1 in a higher uncertainty. We verify a change in the system behavior, as the robot now goes first to check on person 2, even though is in a lower uncertainty area, it is more important for the system to verify with the best camera available at the environment (robot's camera) in order to get the highest possible belief, and so receiving a higher reward for classification. In the meantime, the system receives some false negative observations, but as the robot approaches the person, we get observations from its camera, and the belief is high enough for the system to classify. The classification occurs at timestep 10, when the belief is $b_{10}(F_2 = red) = 0.9785$. Note the high cumulative reward, although person 1 is not classified, as fixed cameras detect no red shirt, and although it has a high uncertainty, while the robot goes to check on person 2, the observations are consistent enough for the belief to be low, and the system decides not to spend resources to move the robot there, and also because is more important to stay protecting the priority area. The final cumulative reward, though the system classifies only one person, is higher than any previous experiment. At timestep 9 the cumulative reward is $CR(t = 9) = -0.7639$, and the system classifies at timestep 10, receiving an immediate reward of $r(t = 10) = 20 \cdot 0.9785 + (-20) \cdot (1 - 0.9785) = 19.14$. The final cumulative reward after classification is:

$$CR(t=10) = -0.7639 + 19.14 \cdot 0.99^{10} = 16.546 \tag{4.19}$$

4.3.7 Discussion

The results presented in this section show that, in general, our models behave well in the proposed scenario. The system reasons over belief of features, and the robot only moves whenever the system is not certain enough about the detections. We tried to test the model in different environment configurations, namely with the person in different positions, and with different shirt color, to make sure that the model is robust enough to handle with such cases. The POMDP framework adopted is mathematically interesting as it allows for direct measuring on the system behavior through analysis of the cumulative reward awarded by the system over time. Such information along empirical observations provides a good tool to derive conclusions over a model's behavior.

We may easily notice the influence that the observation model has on the system, and the robot in particular, behavior. Those areas of the lab where the uncertainty in color detection is higher are the ones to where the system most often sends a robot to confirm the camera network observations. As such, it is important in ACP problems where this model is applied to work on an accurate observation model for the sensor network which models as good as possible the uncertainty in all areas. With a good observation model the system has more information on where it should be more important to send the robot, avoiding spending unnecessary resources, and also avoids the system to trust in noisy sensors.

Note also on Experiment F, which successfully proves that our model is easily adaptable. With only a small change in the reward model we were able to model a different task, in which feature detection is prioritized, that is, detection of some feature is considered to be more important than others, influencing the general system behavior. This proves experimentally one of the great advantages of using POMDP to model such problems.

Chapter 5

Conclusions and Future Work

In this thesis we successfully developed and applied a model to tackle active cooperative perception (ACP) problems in a Network Robotic System (NRS) framework. NRS is a newly concept over which some work has been developed and it is important that research continues on this area, namely on the development of new applications which explore the potential on cooperation offered by NRS. Although we present all the testbed used for experiments, our focus is in the problem of planning in NRS, in particular in ACP problems, in which it is extremely important to have good cooperation between all sensors in the environment in order to successfully complete tasks with the minimum of uncertainty.

We proposed and developed a model based on POMDP, in order to prove that they offer a strong framework for decision-making under uncertainty, and in particular, to model our problem. A POMDP approach offers several advantages:

- POMDP's solvers reason directly over uncertainty in the system and on a belief over all states of the problem, allowing to easily model mathematically the knowledge we have about the world;
- The development of algorithms which allows a factored representation of the state and observation spaces, makes it easy to develop human-readable models, by representing the state space as a set of variables, giving names to each variables and representing different transitions for different variables;
- With the same base model, it is easily adaptable to other ACP problems, by adapting the encoding of the environment (variables and transitions) and the encoding of the goal (rewards).

In the context of ACP MDP-based solutions are not very useful, as they do not compute policies to take informative actions, and our work is based exactly on the capability of the system to take informative actions to decrease uncertainty in the system regarding event detection.

The experimental results in a particular problem demonstrate that our approach to the problem performs well. We particularize our experiments in an ACP problem in which we want to detect persons in an environment and classify them according to a number of features, whether the system detected the presence of those features or not. The system successfully trade off the costs of moving the robot vs getting the desired level of confidence regarding an event, moving the robot only when uncertainty is low enough for the system to have a reasonable doubt and to need the aid of mobile sensors, which in general have a smaller field of view, but a much lower uncertainty.

5.1 Contributions

Contributions of this thesis include some work which brought some new ideas to the research community

in the area of planning under uncertainty and in the development of NRS. The biggest achievement and considered to be the main contribution of this work was to show that we can successfully develop an ACP problem within a NRS and model it as a POMDP problem. We defined the problem and presented a model which deals with decision-making in ACP contexts, with different kind of sensors cooperating to complete tasks.

This presents an important development such as for the NRS research community as for the POMDP community, as we bring together both worlds. While we present a new approach for planning in NRS, where decision making based on incomplete and noisy perception is often crucial for successful task completion, we also present a new area of application for POMDPs. POMDP theory has been the focus of research and there is the need to prove that POMDPs actually apply to real world scenarios, and we give our contribution in this area.

Additionally, in our experiments we particularized the model to a single problem, extending the set of applications where NRS have been tested and used with our system for a surveillance task. Moreover, although we base our system on the already developed work for URUS [39] we develop some new capabilities on the system, such as shirt color observation and we implemented the use of CARMEN software for robot control.

5.2 Future Work

Despite all the positives we achieved on this work, a lot needs to be done before we can start applying such techniques in totally common real world scenarios. POMDP solvers couple well with our model until we model two persons in the environment, not more. With more persons the time and memory consumption are impracticable, as the state and observations space increases exponentially. Therefore, we arise with the question of scalability, and while POMDP theory does not develop further more to reduce time and memory needs for policy computation, there is the need to find alternative ways to scale up to larger problems. For instance, it could be interesting to try different map descriptions, rather than the usual graph-like description, to use a higher level description, like assigning one node per room in a larger scenario, as in [21]. Also, mixed-observability models [48] may be used, in which even if the system's state is not fully observable some of its components may still be fully observable, and online optimization, in order to reduce offline computation, and the amount of time needed for policy computation.

Another important feature we did not considered in our work, but very important in even more realistic scenarios is the existence of blind spots in the environment for the network system. Typically, the network of fixed sensors does not cover all the environment and some places are out of the field of view of all sensors. In these cases, mobile sensors are crucial as they are the only possible source of information for such areas, and the planning system must take that into account. To include such information more variable values for localization should be included in the model to prevent the case when the person is out of sight of all sensors, as even no person is detected by the fixed sensor network, there is a chance that one might be detected in a blind spot.

It would be also interesting to apply and experiment the model in different scenarios and with different ACP tasks, despite our experiments proved to be successful further validations would be obviously useful. That will allow to work on and improve to even better models and extend the scope of our work.

Also, in the NRS setup, it is important to ensure that communications do not fail, and if the sensor network is too big the information flow over the system might require more bandwith than available. In those cases, it could be interesting to combine two types of decision-making, our model with the one presented in [49], although it have effects on the general uncertainty in the system.

Bibliography

[1] Pascal Poupart. *Exploiting structure to efficiently solve large scale partially observable markov decision processes*. PhD thesis, Toronto, Ont., Canada, Canada, 2005.

[2] C. Marques, J. Cristovao, P. Lima, J. Frazao, I. Ribeiro, and R. Ventura. Raposa: Semi-autonomous robot for rescue operations. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 3988–3993, Oct. 2006.

[3] Iwan R. Ulrich, Francesco Mondada, and Jean-Daniel Nicoud. Autonomous vacuum cleaner. *Robotics and Autonomous Systems*, 19:4–233, 1997.

[4] Reid Simmons, Richard Goodwin, Karen Zita Haigh, Sven Koenig, and Joseph O'Sullivan. A layered architecture for office delivery robots. In *In Proceedings of the First International Conference on Autonomous Agents, Marina del Rey*, pages 245–252. ACM Press, 1997.

[5] W. Burgard, A.B. Cremers, Dieter Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and Sebastian Thrun. The interactive museum tour-guide robot. In *Proc. of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 1998.

[6] A. Saffioti, P. Lima, H. Levent Akin, A. Birk, A. Bonarini, G. Kraetzschmar, D. Nardi, E. Pagello, M. Reggiani, A. Sanfeliu, and M. Spaan. Two "hot issues" in cooperative robotics: Network robot systems, and formal models and methods for cooperation. EURON Special Interest Group on Cooperative Robotics, 2008.

[7] A. Sanfeliu, N. Hagita, and A. Saffiotti. Network robot systems. *ROAS*, 56:793–797, 2008.

[8] J. H. Kim, K. H. Lee, Y. D. Kim, N. S. Kuppuswamy, and J. Jo. Ubiquitous robot: A new paradigm for integrated services. In *Proc. IEEE Int. Conf. Robot. Autom.*, pages 2853–2858, 2007.

[9] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1995.

[10] A. Sanfeliu and J. Andrade-Cetto. Ubiquitous networking robotics in urban settings. In *Workshop on Network Robot Systems. Toward Intelligent Robotic Systems Integrated with Environments. Procs. of 2006 IEEE/RSJ International Conference on Intelligence Robots and Systems (IROS2006)*, October 2006.

[11] Masahiro Shiomi, Takayuki Kanda, Hiroshi Ishiguro, and Norihiro Hagita. Interactive humanoid robots for a science museum. *IEEE Intelligent Systems*, 22(2):25–32, 2007.

[12] Alessandro Saffiotti and Mathias Broxvall. Peis ecologies: ambient intelligence meets autonomous robotics. In *sOc-EUSAI '05: Proceedings of the 2005 joint conference on Smart objects and ambient intelligence*, pages 277–281, New York, NY, USA, 2005. ACM.

[13] B. Mazzolai, V. Mattoli, C. Laschia, P. Salvini, G. Ferri, G. Ciaravella, and P. Dario. Networked and cooperating robots for urban hygiene: the eu funded dustbot project. In *URAI 2008: The 5th International Conference on Ubiquitous Robots and Ambient Intelligence*, 2008.

[14] L. Hoesel A. Erman and P. Havinga. Aware: Platform for autonomous self-deploying and operation of wireless sensor-actuator networks cooperating with aerial objects. In *CTIT Symposium*, 2007.

[15] Abdolkarim Pahliani, Matthijs T. J. Spaan, and Pedro U. Lima. Decision-theoretic robot guidance for active cooperative perception. In *IROS'09: Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems*, pages 4837–4842, Piscataway, NJ, USA, 2009. IEEE Press.

[16] AnYuan Guo. Decision-theoretic active sensing for autonomous agents. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 1002–1003, New York, NY, USA, 2003. ACM.

[17] Pradeep Varakantham, Janusz Marecki, Yuichi Yabu, Milind Tambe, and Makoto Yokoo. Letting loose a spider on a network of pomdps: generating quality guaranteed policies. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8, New York, NY, USA, 2007. ACM.

[18] D. Hsu, W.S. Lee, and N. Rong. A point-based POMDP planner for target tracking. In *ICRA*, pages 2644–2650, 2008.

[19] Diane J. Cook, Piotr Gmytrasiewicz, and Lawrence B. Holder. Decision-theoretic cooperative sensor planning. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(10):1013–1023, 1996.

[20] Reid Simmons and Sven Koenig. Probabilistic robot navigation in partially observable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1080–1087, 1995.

[21] Elena López, Rafael Barea, Luis Miguel Bergasa, and Marisol Escudero. Visually augmented pomdp for indoor robot navigation. In *In Applied Informatics*, pages 183–187, 2003.

[22] Nicholas Roy, Geoffrey Gordon, and Sebastian Thrun. Planning under uncertainty for reliable health care robotics. In *in: The Fourth International Conference on Field and Service Robots (FSR*, 2003.

[23] Jennifer Boger, Pascal Poupart, Jesse Hoey, Craig Boutilier, Geoff Fernie, and Alex Mihailidis. A decision-theoretic approach to task assistance for persons with dementia. In *IJCAI'05: Proceedings of the 19th international joint conference on Artificial intelligence*, pages 1293–1299, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

[24] Anthony Cassandra. A survey of pomdp applications. In *Proc. of the AAAI Fall Symposium*, 1998.

[25] Jim Blythe. An overview of planning under uncertainty, 1999.

[26] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1994.

[27] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[28] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[29] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1-2):99–134, 1998.

[30] E. J. Sondik. *The Optimal Control of Partially Observable Markov Decision Processes*. PhD thesis, Stanford, California, 1971.

[31] Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Comput. Intell.*, 5(3):142–150, 1990.

[32] Kevin Patrick Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, University of California, 2002.

[33] J.A. Tatman and R.D. Shachter. Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, 20(2):365–379, 1990.

[34] Craig Boutilier, Richard Dearden, and Moises Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121:2000, 1999.

[35] Jesse Hoey, Robert Staubin, Alan Hu, and Craig Boutilier. Spudd: Stochastic planning using decision diagrams. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 279–288. Morgan Kaufmann, 1999.

[36] Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Kolle. Context-specific independence in bayesian networks. In *Proc. of 12th Conference on Uncertainty in AI*, pages 115–123, 1996.

[37] Mathhijs Spaan and Nikos Vlassis. Perseus:randomized point-based value iteration for pomdps. *Journal of Artificial Intelligence Research*, (24):195–220, 2005.

[38] T. Bandyopadhyay, N. Rong, M.H. Ang Jr., D. Hsu, and W.S. Lee. Motion planning for people tracking in uncertain and dynamic environments. In *IEEE Int. Conf. on Robotics & Automation, Workshop on People Detection & Tracking*, 2009.

[39] Marco Barbosa, Alexandre Bernardino, Dario Figueira, José Gaspar, Nelson Gonçalves, Pedro Lima, Abdolkarim Pahliani, José Santos-Victor, Matthijs Spaan, and Jo ao Sequeira. Isrobotnet: A testbed for sensor and robot network systems. In *In Proc. of IROS 2009 - The IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009.

[40] Intel. *Intel Open Source Computer Vision Library Reference Manual*, 2001.

[41] Michael Montemerlo, Nicholas Roy, and Sebastian Thrun. Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (carmen) toolkit. In *In Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 2436–2441, 2003.

[42] M. Barbosa, N. Ramos, and P. Lima. Mermaid - multiple-robot middleware for intelligent decision-making. In *In Procs. of the IAV2007 - 6th IFAC Symposium on Intelligent Autonomous Vehicles*, 2007.

[43] Dario Figueira, Plinio Moreno, Alexandre Bernardino, and José Gaspar. Detection and localization of persons and robots in a networked camera system. Submitted. Also as ISR/IST Technical Report, 2009.

[44] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2):99–141, 2000.

[45] Kurt Konolige. A gradient method for realtime robot control. In *In Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2000.

[46] Anthony R. Cassandra, Leslie Pack Kaelbling, and James A. Kurien. Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. In *In Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 963–972, 1996.

[47] Michael Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 362–370. Morgan Kaufmann, 1995.

[48] Sylvie C. W. Ong, Shao W. Png, David Hsu, and Wee S. Lee. Pomdps for robotic tasks with mixed observability. In *Robotics: Science and Systems*, volume 5, 2009.

[49] Matthijs T. J. Spaan and Pedro U. Lima. A decision-theoretic approach to dynamic sensor selection in camera networks. In *Int. Conf. on Automated Planning and Scheduling*, pages 279–304, 2009.