Human-Aware Task Planning Under Uncertainty in Networked Robot Systems

Ricardo Dantas da Silva Simas

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor: Doctor Pedro Manuel Urbano de Almeida Lima

February 2017

Agradecimentos

Começo por agradecer ao Professor Pedro Lima, pela sua orientação e por me ter dado a oportunidade de trabalhar na área de robótica da qual constitui um grande exemplo. Só as suas ideias e preciosa ajuda permitiram que levasse o meu trabalho a bom termo. Um agradecimento especial também à equipa do SocRob pelo auxílio que me foi prestado ao longo do desenvolvimento da tese.

Nestes cinco anos, para além dos conhecimentos que adquiri, fiz também amizades que espero me fiquem para a vida. Graças a vós, este tempo foi muito mais divertido. Obrigado também aos meus amigos de longa data pelo apoio que me têm dado.

Agradeço à minha família. Uma palavra de dedicatória aos meus avós, em especial à memória do meu avô Fernando e da minha avó Rosa. Ao meu avô Constantino e em particular à minha avó Alice por tudo o que tem feito por mim.

Aos meus pais. Sempre apoiaram as minhas decisões, encorajaram-me e estiveram presentes em todos os momentos. Sempre me protegeram sem me privarem de viver e brincar. Proporcionaram-me uma infância feliz, para além do que consigo pôr por palavras. O meu pai que é o pai mais orgulhoso do mundo. A minha mãe com quem sempre tive uma ligação especial e a quem dedico a minha tese. Obrigado.

Resumo

Esta tese apresenta uma abordagem ao planeamento de tarefas sob incerteza. O objectivo principal da tese é alcançar planos optimizados, tendo em conta a presença humana e optimizando uma recompensa acumulada num horizonte de tempo finito, calculados através de métodos teóricos de decisão e executados por um Sistema Robótico em Rede (SRR) num espaço fechado. Um SRR é uma rede de robots autónomos, assim como de outros dispositivos, capazes de seleccionar e executar as suas próprias acções, em presença de vários acontecimentos que ocorrem no seu ambiente circundante. Os acontecimentos são detectados com diferentes níveis de confiança, devido à incerteza em percepção. As acções dos robots têm igualmente diferentes níveis de incerteza relativamente ao seu impacto no mundo. Pacotes de ROS são utilizados na implementação de acções e percepções do SRR.

Para realizar planeamento sob incerteza, um Processo de Decisão de Markov Parcialmente Observável (POMDP) é aplicado a uma tarefa. POMDPs são ferramentas matemáticas para tomada de decisões sequencial em ambientes parcialmente observáveis, com modelos de transição Markovianos e recompensas aditivas. A tarefa desenvolvida corresponde a uma acção de prevenção e segurança na qual o SRR deverá ser capaz de alertar humanos para o facto de se encontrarem ou se deslocarem para um local reservado ou perigoso, no qual não deverão estar. Experiências demonstraram que o modelo pode ser utilizado para atingir os objectivos e que é possível realizar interacção entre humanos e robots em ambientes no mundo real.

Palavras-Chave: Planeamento sob Incerteza, Sistemas Robóticos em Rede, *Robot Operating System*, Processo de Decisão de Markov Parcialmente Observável, Interacção Robot-Humano.

Abstract

This thesis presents an approach to task planning under uncertainty. The main goal of the thesis is to attain optimal human-aware plans, by optimizing an accumulated discounted reward over a finite time horizon, computed using decision-theoretic methods, and executed by a Networked Robot System (NRS) in an indoor space. A NRS is a network of autonomous robots and other devices that must be capable of selecting and executing their own actions, in the presence of several events occurring in their surrounding environment. Events are detected with different levels of confidence, due to uncertainty in sensing. Robot's actions have different levels of uncertainty concerning their impact in the world. ROS packages are used to implement actions and perceptions of the NRS.

To achieve planning under uncertainty, a Partially Observable Markov Decision Process (POMDP) is applied to a task. POMDPs are a mathematical framework for sequential decision-making in partially observable environments with Markovian transition models and additive rewards. The task developed corresponds to a prevention and safety action where the NRS should be capable of warning humans that they may be moving or staying at a restricted or dangerous location and should not be at that place. Experiments evidence that the model can be used to accomplish the objectives and that it is possible to perform human robot interaction in real world environments.

Keywords: Planning Under Uncertainty, Networked Robot Systems, Robot Operating System, Partially Observable Markov Decision Process, Human Robot Interaction.

Contents

List of Tables

List of Figures

Acronyms

ADD Algebraic Decision Diagrams. 5, 20, 21, 33, 35, 59

AMCL Adaptive Monte Carlo Localization. 37

HOG Histogram of Oriented Gradients. 37–41, 43

HRI Human-Robot Interaction. 1–4, 6, 25, 30, 32, 35, 53

INSIDE Intelligent Networked Robot Systems for Symbiotic Interaction with Children with Impaired Development. 4

ISR Institute for Systems and Robotics. 2, 30

MCTS Monte-Carlo Tree Search. 6

MDP Markov Decision Process. 2, 5, 6, 9–14, 16, 17

MiGS Milestone Guided Sampling. 5, 16

NRS Networked Robot System. 1–6, 21, 25, 29, 31, 35–37, 53, 54

PBVI Point-Based Value Iteration. 17, 18

POMCP Partially Observable Monte-Carlo Planning. 6

POMDP Partially Observable Markov Decision Process. 2–6, 9, 12–19, 21–23, 26, 29, 31, 32, 34–37, 43, 45, 46, 51–54, 59

RL Reinforcement Learning. 2, 4, 6

RNN Recurrent Neural Networks. 6

ROS Robot Operating System. 3, 30, 36, 42, 54

RPG Recursive Policy Gradient. 6, 16

SVM Support Vector Machine. 37, 39–41

Chapter 1

Introduction

1.1 Motivation and problem definition

Technological progress in the last few decades allowed the evolution of scientific fields such as computation, networked vehicles, communications and artificial intelligence. This progress brought improvements in the area of robotics. Robots capabilities have been expanding and, while there are still limitations in the usage of robots (e.g., batteries, mobility, equipment), their presence and importance in society is increasing. Robots can now interact with human beings and other robots in cooperative ways. These recent technology systems are denominated Networked Robot Systems (NRSs). A NRS is a network of devices, with sensors and actuators, as well as mobile robots, capable of making autonomous decisions to complete tasks [1]. To make autonomous decisions, a NRS acquires information about the environment, using sensors, and executes actions according to the information received and its goals, using actuators. These systems are connected to a communications network that are wired or wireless.

The usage and popularity of NRSs in society have been increasing over time due to the possibilities they generate. One of the main objectives and purposes of creating NRSs is enabling Human-Robot Interaction (HRI). The number of applications for robots in the real world has increased in the past and continues to expand in the present. Applications include manufacturing, interaction with human beings in tasks of guidance, personal assistance, transportation of goods, elderly care, providing information, rescue operations, bomb diffusion, surveillance, monitoring and entertainment [2].

The successful completion of a task depends on the robots decisions and actions. It is important to guarantee that a robot makes good decisions that end in reaching the proposed goal. Actions are chosen according to the robot's environment, which may be deterministic or stochastic. In a deterministic environment, there is no randomness involved, the same input always produces the same output. That is not the case in a stochastic environment, where events are unpredictable. NRSs environments are typically stochastic. Uncertainty in robotics derives from the noisy and/or incomplete observation of the state and the uncertain outcomes of the robot system actions perception. This lack of information is usually due to limitations in the equipment or the environment. One main issue responsible for uncertainty is the quality of the robot's sensors. Sensors are necessary to collect information about the environment, which causes their limitations to have a big impact in the observation models of a robot. Robot's sensors are often limited to observing its direct surroundings, and might not be appropriate to monitor features of the environment's state beyond its vicinity. Furthermore, robot's sensors may contain technical limitations or imperfections (e.g. range, resolution, noise) which are another cause of uncertainty. Sensors are not the only cause of uncertainty. Robot's actuators involve motors that are affected by noise, legs or wheels that may not be able to roam in certain locations and possibly arms which are limited to certain movements because of its joints. Since NRSs are real-time systems, the amount of computation that can be carried out is limited which may decrease the accuracy of algorithms used.

Planning is made in a model of the world. Since models are abstractions of the real world, not all effects of the real world are modelled. Therefore, state, observation and action models are not completely accurate. Human presence in environments increases the chance of inaccuracy in modelling, since their intentions are difficult to predict [2]. Constant movement and change of behaviour in humans is hard to measure and analyse. Physical similarities between humans is also a reason for uncertainty thus making identifying a specific human a quite difficult task. Frequent movement of objects in the real world may also decrease accuracy in modelling.

Human-aware task planning under uncertainty is an important research topic in robotics. It consists of designing and implementing optimal plans, subject to uncertainties regarding action effects and sensor observations, to perform HRI. Optimal planning is what makes robots capable of selecting and executing actions on their own, in the presence of several events occurring in their surrounding environment and of their modelled uncertainty. To achieve planning that can be seen as optimal, it is necessary to choose the actions, at each moment, that will help reach a more positive outcome in the future. Every time the agent acts, it receives a reward associated with the action taken and the state of the environment. The goal in optimal planning is for the agent to act in a way to maximize a cumulative function of expected long-term rewards. Therefore, there is an optimization of the accumulated discounted reward over a finite time horizon.

To determine the actions of a NRS, Markov Decision Process (MDP) models of the world and Reinforcement Learning (RL) components are used. Assuming an agent, in this case a NRS, knows with full certainty the state of the environment and it is able to fully observe other states, then it is possible to compute optimal behaviour using the MDP framework. However, in the case under study, the presence of uncertainty in sensing prevents states of the environment to be fully observable. To overcome this problem, an extended version of the MDP framework, Partially Observable Markov Decision Process (POMDP) is used. POMDPs enable optimal decision making in environments which are only partially observable to the agent by relating possible observations to states.

A brief summary of the work developed in this thesis is presented in this paragraph. To validate human-aware planning, a prevention and safety task capable of warning people to avoid moving or staying in a restricted/dangerous place was created. A POMDP model was developed for this purpose. The model assumes navigation actions that were implemented in a robot. An observation system, composed by a robot and a surveillance camera, enabling to observe the position and movement of a person was developed. The POMDP observations models were conceived based on the accuracy of this observation system. Experiments were conducted to test the implemented model at the Laboratório de Robótica Móvel (LRM) at Institute for Systems and Robotics (ISR). Results showed that the robot is capable of moving to restricted/dangerous places when a person is observed going (or staying) to the exact same place, in order to avoid that person from going (staying) there. These results validated the usage of a POMDP in uncertainty conditions.

The thesis is inserted in the SocRob@home[1] project at ISR[2] at Instituto Superior Técnico. The work carried out can be integrated in the project and help improve it. Finally, while there is some investigation on POMDPs, including at ISR, it is still a subject with great possibility to evolve. The practical work in this thesis as well as the work developed by researchers play an important role for that evolving.

HRI provides therapeutic benefits to some people while it helps others complete simple daily tasks. There is still a lot of work to be done in making robots fully autonomous but as research and technology advances, so does HRI, which will prove to be more useful and helpful to people in the future.

[1]`http://socrob.isr.ist.utl.pt/dokuwiki/doku.php?id=socrobhome:socrobhome`
[2]`http://welcome.isr.tecnico.ulisboa.pt`

1.2 Goals

The main goal of the thesis is to attain optimal human-aware plans, computed using decision-theoretic methods, and executed by a NRS in an indoor space. Planning is conditioned by uncertainties in acting and sensing. Once optimal planning is achieved, a NRS should be capable of selecting and executing actions in order to interact with people in an environment similar to a home or a hospital. The interaction with people has a specific designed task where previous planning is essential. The developed task corresponds to a prevention and safety task where the NRS should be capable of warning people that they may be moving or staying at a restricted or dangerous location and should not be at that place.

To successfully complete the proposed main goal, other related objectives have to be accomplished. Since the POMDP framework is a decision method that provides information on how to perform planning under uncertainty, it is necessary to develop a POMDP model for the given human-aware task. Once the model has been fully designed, optimal actions are determined using an algorithm to solve the formulated POMDP problem. This constitutes the decision system in the overall task. Due to the existence of multiple solver algorithms, it is important to study several of them to use one that is efficient and effective. Actions selected by the decision system need to be implemented in the NRS. Integration between the NRS and the decision system, and implementation of actions and perception are conducted with the Robot Operating System (ROS)[3]. ROS is an open source set of software libraries and tools that can be used for robot applications. Finally, a series of experiments with interactions between a NRS and a human are carried out.

1.3 Related work

Interactions between robot systems and people have been increasing over time. While in the past, robots were mainly confined to laboratories and research facilities, nowadays they can be used and tested in private homes or public spaces like hospitals or schools. Robots' capabilities have also improved, which enlarges the spectrum of usage and allows them to be applied in a large range of tasks including helping people.

In [3], a survey on long-term interaction between social robots and people shows that robots can be used in health care and therapy as participants showed reduced levels of stress and better social interaction. Autistic children expressed interest and affection towards the robot. The survey also evidences that robots can be used in education as robots behaving as English tutors were able to improve children's English skills. In at home interactions, robots help people in chores. However the survey shows that some people lose interest over time unless robots are capable of engaging them on extended periods of time and display high levels of functionality. In [4], a survey on assistive robotic technology demonstrates that robots can be used in areas such as autism; elder-care; intelligent wheelchairs; assistive robotic arms; prosthetic limbs; and post-stroke rehabilitation.

One example of how robots can be used in therapy is presented in [5]. Therapeutic seal robots were introduced in an elderly care house in order to evaluate the psychological and social effects of interactions between robots and residents. The seal robots of the experiment resembled stuffed seals. The robots also had multiple devices such as microphones, a speaker, tactile and light sensor, and multiple actuators for movement. These made the robot more attractive and friendly to users. Results were positive as residents displayed a better mood and became more active and communicative with each other and with caregivers. Residents also showed lower stress levels. On patients with dementia, the interaction with the robot seal improved their cortical neurons activity.

[3]`http://www.ros.org`

The study in [6] focuses on HRI in domestic environments. The study was developed by inserting vacuuming robots in thirty households and observing their usage over a time span of six months. Most participants expected the robots to be able to clean their entire house thus improving the overall cleanliness and their quality of life. The high price of robotic devices is one of the reasons why people expect robots to fully complete their tasks with minimal or no manual assistance. The fact the robot had limitations while cleaning the house was not appreciated by some of the participants. However, experiments were positive with householders approving the robot's cleaning ability and even cleaning more often. Participants also interacted with the robots on different levels by helping them clean, giving them a name, talking to them and putting them in specific indoor places to see how they would behave. The study shows that people are willing to use and interact with robots, especially if their functions are useful and well executed. It also suggests that communication between robots and cameras or other robots can improve and diversify the usage of robots.

While NRSs are a relatively new field, there is quite some work done in this area. Several organizations around the world have projects and research in the field and have been able to test HRI.

One example was the URUS (Ubiquitous Robotics Network System for Urban Settings) project. The URUS project involved eleven European partners and focused on urban pedestrian areas. This is an important topic in Europe as reducing the number of cars in the streets is becoming a necessity and it helps improving the overall quality of life. The URUS project focused on designing and developing a network of robots that, in a cooperative way, would be able to interact with human beings and the environment for tasks of guidance and assistance, transportation of goods, and surveillance in urban areas [1].

One other major project in the field is the Japan NRS project, which has the purpose of providing information and support to people. Robots have visible forms, such as humanoids and pet robots, which helps to enable a user-friendly interaction and are capable of recognizing each person. Robots engage with users in communication tasks taking into account their behaviour and needs [1].

The Intelligent Networked Robot Systems for Symbiotic Interaction with Children with Impaired Development (INSIDE) project "explores the benefits that the interaction with a heterogeneous network of intelligent devices can bring to the therapy of children with Autism Spectrum Disorder"[4].

The mentioned projects are only a small part of the work developed in the field. To accomplish HRI it is necessary to overcome problems related to planning under uncertainty, due to action effects and sensor observations. Extensive work in human-aware task planning under uncertainty has been developed over time. This work allowed the appearance of techniques capable of solving and using the POMDP framework to design and implement optimal solutions for planning. Applications of POMDPs in real world situations have been studied over time and have been of great importance in different areas.

In [7], a POMDP model is used on tasks relevant to support elderly people with cognitive or physical disabilities. The tasks are: assisted hand washing, health and safety monitoring, and wheelchair mobility. Experts or caregiving professionals choose reward and transition functions adequate to the target population. A prior model is obtained from a set of training data gathered using an existing automated system or a human caregiver. The model is learned using the expectation-maximization (EM) algorithm and is adjusted for each task and user, through Bayesian RL.

In [8], an approach is made to find the best course of action to assist people with dementia in a task of handwashing. During the handwashing task, an environment called COACH is used to monitor progress and provide help to subjects. Since COACH uses a computer vision system to identify hands (by colour) there is uncertainty in the visual monitoring of patients caused by noise. Decisions of when and how assistance is needed in a scenario require some form of planning due to the present and future impact that assistance actions have. Therefore, a POMDP model is used to deal with partial observability

[4]`http://gaips.inesc-id.pt/inside`

and plan the appropriate actions for each situation. The developed POMDP model has a large state and action space. Transition probabilities over the state variables of the state space are specified by dynamic Bayesian networks with the conditional probability tables represented by Algebraic Decision Diagrams (ADD). The reward function was obtained through interaction with caregivers, previous clinical trails with patients and evaluation of MDP policies. The model was solved with Perseus-ADD, a redesigned version of the Perseus solver to take into account the ADD structure. Tests showed that the POMDP model designed outperformed in simulation a MDP model created for the same task but considering full observability. Results evidenced that while human caregivers had a higher rating in the evaluated aspects, the MDP model can be used as a supplement in aiding subjects.

In [9], it is shown how to model a cooperative perception task in a NRS. The NRS is constituted by a camera network and a robot with an on-board camera and a laser range finder. The task consists of tracking and classifying people. As this task requires the use of cameras and sensors, there is uncertainty related with the task, especially when the person being tracked or identified is far away from the sensors. The POMDP model is represented as a two-stage dynamic Bayesian network, which allows independence between variables related to the robot and person's location and identification. Results illustrated that the higher the certainty in the detection of a person is, the more the robot needs to move to check that detected event, which means it is possible to trade the cost of moving a robot with desired level of confidence regarding an event.

The Nursebot project was created to help nurses accomplish their duties in a more effective way and improve the daily quality of life of elderly people [10]. The project uses autonomous mobile robots equipped with navigation and interaction sensors. Decisions taken by the robots are selected according to information gathered by sensors and with a control architecture. The control architecture is modelled as a POMDP. In the Nursebot project, since the working domain is highly structured, the decision making problem is divided as subset of problems (Dynamic Programming). For this reason, it is used a hierarchical POMDP where the action space is partitioned.

A common task where POMDPs are applied is navigation. In [11], robot navigation under uncertainty in scenarios of long planning horizons (the amount of time an agent looks into the future when preparing a plan) is performed. Due to the presence of uncertainty, POMDPs are used in planning and it is studied how a point-based algorithm can help computing POMDPs to successfully perform navigation in 2D and 3D environments. Tests of a mobile robot finding a moving target also took place. The fact that planning occurs in scenarios of long planning horizons increase the complexity of the problem. The solver, developed by the authors, to compute the formulated POMDPs, takes the high complexity and dimension into consideration. The method is the Milestone Guided Sampling (MiGS) which is a point-based POMDP solver, that reduces the planning horizon. MiGS samples a set of points, called milestones, acquired from the agent's state space, and uses them to form a compact, sampled representation of the state space. Results in both 2D and 3D navigation and in target finding were positive with the robot completing its tasks in few minutes.

One of the most demanding challenges in planning stands on the scalability of models and their applicability in the real world. In [12], it is presented a memory-bounded dynamic programming algorithm. This algorithm was designed considering finite horizon decentralized POMDPs. The algorithm selects relevant points of the belief space, which are identified using a set of heuristics. The belief points then allow to choose the best sequence of policies in each finite horizon and these policies are stored in memory. While the algorithm is able to tackle problems with large planning horizons and of high complexity, it runs out memory rapidly. A possible strategy to overcome this problem consists of diminishing the total number of policies kept in memory.

The Monte-Carlo Tree Search (MCTS) [13] is a search method for online planning that has a high performance in large and fully observable domains. This method is computationally efficient and highly parallelisable. In short, the method evaluates the state space, by taking random samples of each state, and by using the average outcome of computed simulations for each state, is able to build a search tree corresponding to the desired actions. The method was extended in [14], in order to also enable planning under uncertainty. The authors created an algorithm named Partially Observable Monte-Carlo Planning (POMCP). POMCP uses the same set of Monte-Carlo simulations to perform tree search and to update the agent's belief state, recurring to a particle filter. The search is conducted through the PO-UCT algorithm, which allows to choose actions at each time step, by applying a Upper Confidence Bounds for Trees (UCT) search to a history-based MDP. Experiments of POMCP were carried out in two problems: *battleship* and *pocman*. Both problems have a high dimension, with the *battleship* problem having a high number of states and actions and the *pocman* problem having a high number of states and observations. The POMCP was able to obtain good results. In the *battleship* problem, POMCP won twenty-five moves faster, on average, than a human being. In the *pocman* problem, POMCP obtained an average undiscounted return larger than other tested methods.

In [15], it is discussed how policy gradient methods can be successful in solving RL tasks. A policy gradient method is presented, capable of dealing with memory by representing it in a Recurrent Neural Networks (RNN) architecture that captures long term time dependencies. This method is called Recursive Policy Gradient (RPG) and is an algorithm that backpropagates in time the return-weighted eligibilities, in the RNN. The algorithm works as a supervised RNN but considers the outputs as a probability distribution. RPG is therefore able to learn memory-based policies for deep memory POMDPs which enables to know policy updates of any event in history. Experiments were conducted in three tasks: a double pole balancing POMDP with incomplete state information; a T-maze; and a car racing simulator where the agent needs to learn how to drive. In the pole balancing task, RPG achieved a better performance compared with other policy gradient methods. In the T-maze task, RPG outperformed two value iteration methods, since the T-maze task only required a simple policy which is less complex that the usage of a value function. Finally, in the car racing simulation task, the agent was able to learn how to drive but obtained worse results than a human player.

1.4 Outline

The following five chapters include theory concepts, development of the thesis work, experimental results and conclusions. The thesis is structured as described below:

- Chapter 2 introduces a background in planning under uncertainty, namely the concepts of the MDP and POMDP mathematical frameworks. Ways of solving POMDP problems and of representing them are also presented;
- Chapter 3 presents an HRI task where uncertainty is involved and decision-making is required. This task is modelled into a POMDP with the associated states, actions, observations, transition models, reward function and observation models being explained;
- Chapter 4 covers the implementation and resolution of a POMDP model for the task, presented in Chapter 3, the integration of the HRI task with the NRS, how actions are executed and the way observations are perceived;
- Chapter 5 exhibits the experiments conducted and the results and their analysis in the context of the thesis goals;
- Chapter 6 concludes the thesis with remarks on the work developed and possibilities/suggestions of research and experiments that may be explored in the future.

Chapter 2

Background

This thesis focuses on human-aware task planning under uncertainty. There are a few tools that allow to plan under uncertainty. In the case in study, the MDP framework handles uncertainty in action effects but it does not deal with uncertainty in sensor observations. To overcome this problem, one tool that can be used is the Partially Observable Markov Decision Process (POMDP) which is a generalization of the MDP and allows decision making under uncertainty in environment observation and action outcomes. To understand the theory related with this thesis, it is first necessary to understand MDPs and POMDPs, concepts that are introduced in this chapter.

2.1 Markov Decision Process

A Markov Decision Process is a sequential decision problem for a fully observable, stochastic environment with a Markovian transition model and additive rewards [16]. This process assumes the existence of an agent that interacts synchronously with the environment. As shown in Figure 2.1, the agent takes as input the state of the environment, generates as output actions, which affects the state of the environment, and receives a reward. In a MDP, there may be uncertainty related to the effects of the agent's actions, but there is never uncertainty about the agent's state [17].

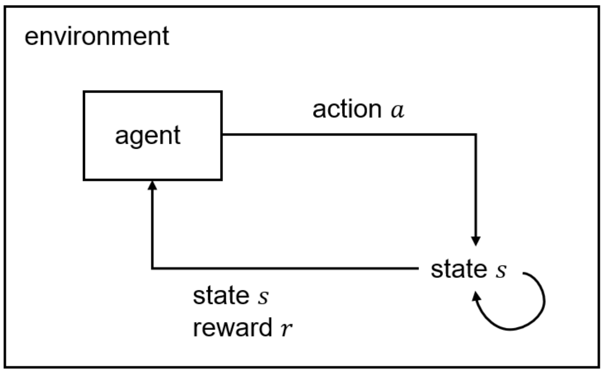


Figure 2.1: MDP agent interacting with the environment.

The MDP framework can be described as:

- a finite set of states S;
- a set of actions $A(s)$ for each state s;

- a transition model $T(s,a,s') = P(s'|s,a)$ that corresponds to the probability that action a in state s will lead to state s';
- and a reward function $R(s,a)$ which is the immediate reward received for taking action a when in the current state s.

The notation $R(s,a,s')$ can also be used to describe the reward function. However, considering the next state s' increases the difficulty of computing the reward function. The model presented in Chapter 3 was developed using only the current state s and an action a. Therefore, the notation $R(s,a)$ is the one used in this dissertation.

The next state s' depends on the current state s and the chosen action a. After moving to the next state, the agent receives a reward r. Given that, the current state and the chosen action are independent from previous states and actions, the state transitions satisfy the Markov property. The Markov property corresponds to stochastic processes in which future states depend only on the current state and are therefore independent of previous events. A process that respects this property is called a Markov process.

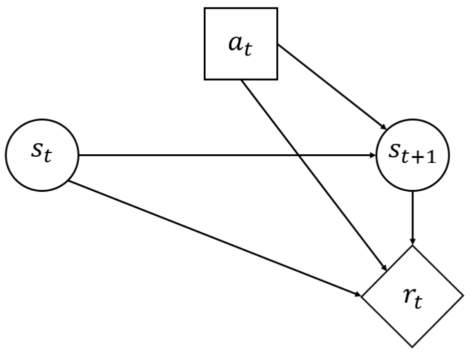


Figure 2.2: Decision network representing a MDP model.

2.1.1 Policies and Value Functions

The main problem of MDPs consists of determining the optimal action for the agent in a given state. A solution that specifies the agent's behaviour for any state it might reach is called a policy. If the agent has a complete policy, then it always knows how to behave, no matter the outcome of an action or the current state [16]. A policy is denoted by $\pi(s)$, which is the action mapped by policy π for the state s. When choosing a policy, there is the intention of finding the optimal action for each situation. The goal is to choose a policy that allows the agent to act in a way as to maximize a cumulative function of expected long-term rewards. Typically,

$$E\left[\sum_{t=0}^{h} \gamma^t R(s_t, a_t)\right], \tag{2.1}$$

where $E[\cdot]$ denotes the expectation operator, h is the planning horizon and γ is a discount factor that satisfies $0 \leq \gamma < 1$.

The discount factor γ ensures that rewards received in the near future are more valuable than rewards received later. The discount factor also ensures that the series converges in the case of an infinite horizon problem ($h = \infty$).

When a given policy is executed, the stochastic nature of the environment may lead to a different environment history. The quality of a policy is therefore measured by the expected utility of the possible environment histories generated by that policy. The expected utility value function $V^{\pi}(s)$ is obtained from the expression (2.1) where the action executed corresponds to the policy for each given state s.

$$V^{\pi}(s) = E\Big[\sum_{t=0}^{h} \gamma^t R(s_t, \pi(s_t))\Big] \tag{2.2}$$

The utility of being in a state corresponds to the expected sum of discounted rewards from that state forward. Therefore, assuming the agent chooses the optimal action based on a policy for the given state, it can be said that the utility of a state is the reward of that state plus the expected discounted utility of the next state. This means that the expression (2.2) becomes,

$$V^{\pi}(s) = R(s, \pi(s)) + E\Big[\sum_{t=1}^{h} \gamma^t R(s_t, \pi(s_t))\Big] \tag{2.3}$$

The expectation operator corresponds to the utility value function of the next state and its average is equivalent to the stochastic transition model between states. This leads to a recursion called the Bellman recursion. Finally the utility of a state with a policy is given by,

$$V_h^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) V_{h-1}^{\pi}(s') \tag{2.4}$$

The expression (2.4) is the Bellman equation, also known as a dynamic programming equation, and describes the expected reward for taking the actions prescribed by some policy π.

The policy that provides the highest expected utility is called an optimal policy. Its expression is,

$$\pi^*(s) = \operatorname*{argmax}_{\pi} V^{\pi}(s) \tag{2.5}$$

Knowing the optimal policy, it is possible to obtain the optimal utility value function which describes the reward for taking the action with the highest expected return. This function is the Bellman optimality equation and is given by,

$$V_h^*(s) = \max_{a \in A(s)} \Big[R(s, a) + \gamma \sum_{s'} P(s'|s, a) V_{h-1}^*(s')\Big], \tag{2.6}$$

where the action is given by the optimal policy for the current state.

2.1.2 Value Iteration

The Bellman equation (2.4) can be used for solving MDPs. There are as many Bellman equations as states since each state has its own equation. Because the equations are nonlinear, solving the system of Bellman equations may prove to be difficult and computational expensive, especially if the number of states is high [16]. To solve this problem, an approximation technique using iterations can be performed. The technique consists of granting initial values to utilities and performing successive iterations which allow to update the value of each utility. This process is repeated until the utility of each state reaches an equilibrium point. The method is denominated value iteration algorithm and it allows to solve MDPs. The iteration step is called a Bellman update and its expression is presented in (2.7).

$$V^*_{n+1}(s) = \max_{a \in A(s)} \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*_n(s') \right] \tag{2.7}$$

The presented solution allows to determine optimal actions for an agent by solving MDPs. However it assumes that, when choosing an action for the agent, the state s is known. If that is not the case, it is not possible to calculate a policy with the solution. This problem is called a Partially Observable Markov Decision Process.

2.2 Partially Observable Markov Decision Process

Partially Observable Markov Decision Process is a mathematical model for sequential decision-making in partially observable environments. This process is an extension of a MDP where the agent cannot directly observe the entire state space. POMDPs and MDPs share many elements as both frameworks enable the agent to influence the system state of an environment by executing actions. In each state, the agent takes an action that enables transitions between states according to a probabilistic transition function and a reward given to the agent. What distinguishes a POMDP from a MDP is the way observations are made when the agent interacts with the environment. In a MDP the agent is able to observe the state space directly while in a POMDP the agent is unable to observe states, due to imperfections in sensing, observing instead possible sensor readings. So essentially, the difference is that a MDP observes a state s and a POMDP perceives an observation o. Figure 2.3, an adaptation from [18], represents the interaction between a POMDP agent and its environment.

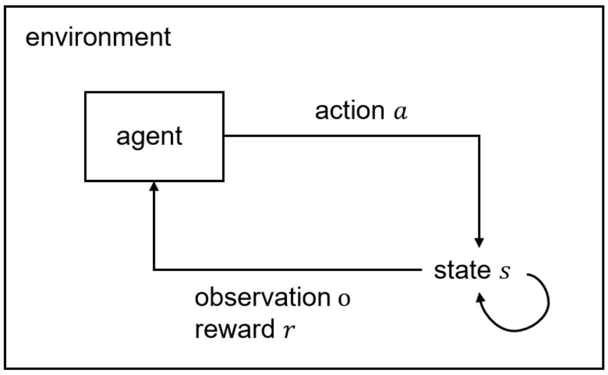


Figure 2.3: POMDP agent interacting with the environment.

A POMDP model can be described as:

- a finite set of states S;
- a set of actions $A(s)$ for each state s;
- a transition model $T(s,a,s') = P(s'|s,a)$ that corresponds to the probability that action a in state s will lead to state s';
- a reward function $R(s,a)$ which is the immediate reward received for taking action a when in the current state s;
- a discrete set of observation Ω;
- an observation model $O(s',a,o) = P(o|s',a)$ representing the probability of observing o in state s' after executing action a.

Like in a MDP, the next state s' depends on the current state s and the chosen action a. After the transition between states, the agent receives a reward r and perceives an observation o providing information from the state s'.

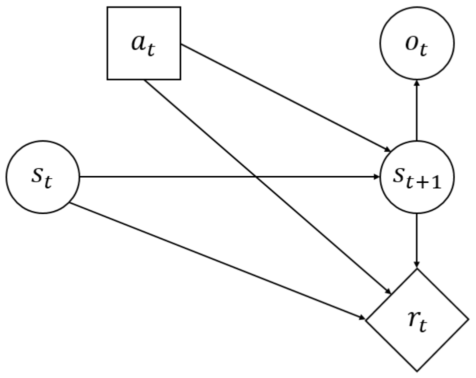


Figure 2.4: Decision network representing a POMDP model.

In order for $O(s', a, o)$ to be a valid probability distribution over possible observations the conditions (2.8) and (2.9) must be respected.

$$\forall s' \in S, a \in A, o \in \Omega, O(s', a, o) \geq 0 \tag{2.8}$$

$$\sum_{o \in \Omega} O(s', a, o) = 1 \tag{2.9}$$

Just like in a MDP, the goal in planning is to find a way of choosing actions that maximize the expected sum of future rewards [19]. For instance,

$$E\Bigg[\sum_{t=0}^{h} \gamma^t R(s_t, a_t)\Bigg], \tag{2.10}$$

where $E[\cdot]$ denotes the expectation operator, h is the planning horizon and γ is a discount factor that satisfies $0 \leq \gamma < 1$.

2.2.1 Belief State

In a MDP, it is possible to maximize the expected long-term reward by choosing a policy for the given state. However, in a situation where the current state is not fully observable, choosing a policy may prove to be quite difficult as the state of the environment is unknown. The agent's observations, from sensors, do not uniquely identify the state of the environment. So, as the agent progresses in the state space, the chances of wrong observations from sensors increase. One way to avoid this problem would be to store in memory the agent's behaviour, that is, to store the sequence of actions executed and the observations received. However, this would make the process non-Markovian and could require a huge amount of memory as the number of states and observations is high in long planning horizons [18].

Instead of using memory and considering that the agent is in one single state, multiple states are considered. The agent is in what is called a belief state (b) which is the set of possible states where the agent might be. In POMDPs all possible states in the environment are considered which means that the belief state is a probability distribution over all states. This probability distribution allows to keep the same amount of information as saving the agent's behaviour over time would and keeps the process Markovian [17].

The usage of belief states is in fact one of the main advantages of POMDPs against heuristics. While an heuristic only considers current observations to choose actions, or past ones when using memory, a POMDP considers belief states whose values are updated over time according to past values and the transition and observation models. Therefore, a POMDP model is capable of selecting actions according to observations perceived over time, without the need to store them in memory.

The probability distribution over states is presented in Figure 2.5.

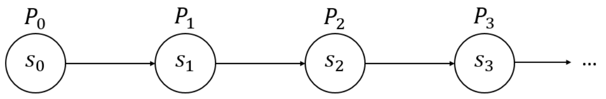


Figure 2.5: Probability density distribution over states.

A POMDP problem assumes an initial belief state b_0. Every time the agent executes an action and makes an observation, the belief state is updated. The main idea is that the optimal action for the agent only depends on the current belief state. Hence, decision making of a POMDP agent consists of: executing an action according to the current belief state; receiving an observation of the agent's state; updating the belief state in compliance with the action taken and the observation obtained.

The belief state is updated by the Bayes rule: [20]

$$b'(s') = P(s'|o,a,b) \tag{2.11}$$

$$b'(s') = \frac{P(o|s',a,b)P(s'|a,b)}{P(o|a,b)} \tag{2.12}$$

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

$$b'(s') = \frac{O(s',a,o)}{P(o|a,b)} \sum_{s \in S} T(s,a,s')b(s) \tag{2.14}$$

It is important to point out that $b(s)$ refers to a belief state and $b'(s')$ is the corresponding updated belief state. It is the transition from one belief state to the next one. Initially $b(s) = b_0$. $P(o|a,b)$ is a normalizing constant with $P(o|a,b) = \sum_{s' \in S} O(s',a,o) \sum_{s \in S} T(s,a,s')b(s)$.

2.2.2 Policies and Value Functions

In POMDPs, policies map beliefs to actions and not states to actions like in MDPs. A chosen policy generates an action which makes the agent move from one state to another. As this happens, the belief state is updated and a new action is generated. Just as in a MDP, the goal of the agent is to choose actions that fulfil its task in the best possible way. To choose which actions are most fulfilling, it is necessary to select them according to the long-term effects on the agent's total reward [21]. Accomplishing this means finding the optimal policy $\pi^*(b)$ for the agent. The quality of a policy is measured by an expected utility value function $V^\pi(b)$. The value function is defined as the expected future discounted reward the agent can collect by following a policy π and starting in a belief b_0.

$$V^{\pi}(b_0) = E\Big[\sum_{t=0}^{h} \gamma^t R(b_t, \pi(b_t))|b_0, \pi\Big] \tag{2.15}$$

The optimal policy is obtain by optimizing the long-term reward.

$$\pi^*(b) = \underset{\pi}{\operatorname{argmax}}\, V^{\pi}(b_0) \tag{2.16}$$

Using the same reasoning that allowed to reach the Bellman equation in (2.4) and knowing the optimal policy, the highest expected reward value for each belief state, that is, the Bellman optimality equation is given by,

$$V_h^*(b) = \max_{a\in A}\Big[r(b,a) + \gamma \sum_{o\in O} O(b,a,o) V_{h-1}^*(b')\Big], \tag{2.17}$$

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

Computing utility value functions over a continuous belief space may seem intractable. But for finite-horizon POMDP, the optimal value function is piecewise linear and convex. It can be parameterized by a finite set of vectors/hyperplanes. The technique of value iteration uses a Bellman update to achieve a better value approximating it to an optimal value function, maintaining its piecewise linearity and convexity [22]. Since the value of the optimal policy depends on the value function, the value iterarion algorithm also improves the policy.

Figure 2.6 shows an example of a value function for a two-state POMDP taken from [18]. The y-axis shows the value of each belief and the x-axis shows the belief space $\Delta(S)$.

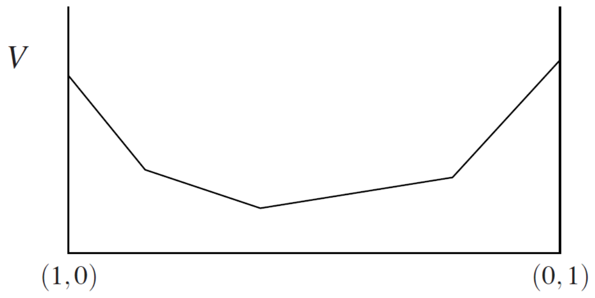


Figure 2.6: Example of a value function in a two-state POMDP.

Convexity implies that the values of beliefs closer from the corners of $\Delta(S)$ are higher. Since beliefs located at corners have higher values, then they also have better policies and a higher certainty of the agent's state. In the example of Figure 2.6, as the belief space is a simplex, it is possible to represent any belief in a two-state POMDP on a line, as $b(s_2) = 1 - b(s_1)$. This means that the highest value of $b(s_2)$ (1) is reached with lowest value of $b(s_1)$ (0). The same reasoning goes for the highest value of $b(s_1)$. Therefore, the highest value of a belief is in a corner. In the case of $b(s_2)$ it's in the corner (0,1), and in $b(s_1)$ it's in (1,0). This concludes that a belief located in a corner has a higher (or equal) value than a belief in another position of the belief space [18].

2.2.3 Techniques for solving POMDPs

When a model of the environment is available, a policy for the agent can be computed. However, even when the environment is known to the agent, computing POMDPs can be computationally demanding, particularly for major problems. This problem can be surpassed by using methods that allow to compute approximate solutions of problems in efficient manners. Examples of methods such as MiGS and RPG have been mentioned at the Related Work section of Chapter 1. In this section, other relevant and common methods are presented.

2.2.3.1 Q_{MDP} algorithm

One way of solving problems is through heuristic control strategies. Solving MDPs is of much lower complexity than solving POMDPs, so using MDP-based heuristics is a possible strategy. These strategies are greedy approaches that assume the POMDP will become observable after the next action [23].

The Q_{MDP} algorithm is a combination between MDPs and POMDPs. The algorithm estimates the expected return for each action and selects the action that contains the highest value. The value function is MDP-optimal, dismissing the state uncertainty in POMDPs. Q_{MDP} is computationally as efficient as a MDP but returns a policy that is established over the belief state. The control policy can be defined as,

$$\pi_{Q_{MDP}}(b) = \operatorname*{argmax}_a \sum_s b(s) Q^*_{MDP}(s,a) \tag{2.18}$$

The policies of the Q_{MDP} algorithm do not carry out informative actions since Q_{MDP} evidences that any uncertainty regarding the state will disappear after one action is carried out [24]. This means that, since uncertainty disappears after one action, Q_{MDP} policies fail in domains where it is necessary to gather repeated information [18]. Therefore, the Q_{MDP} algorithm is only useful in specific situations and cannot be applied to every problem.

2.2.3.2 Value Iteration

Just like in MDPs, it is possible to determine optimal actions for an agent by solving problems with the value iteration algorithm. In a POMDP, the optimal value function is continuous, piecewise linear and convex, which is a property that can help find a solution. The value iteration method builds a sequence of value function estimates which tends to the optimal value function for the current task [22]. A value function in a finite-horizon POMDP can be parameterized by a finite number of vectors over the belief space. Each vector increases the value function in a certain region. Each vector has an associated action, which is the optimal one for beliefs in that region [18]. Figure 2.7, obtained from the example in Figure 2.6, shows an example of how a belief space may be divided into regions, each with an associated vector.

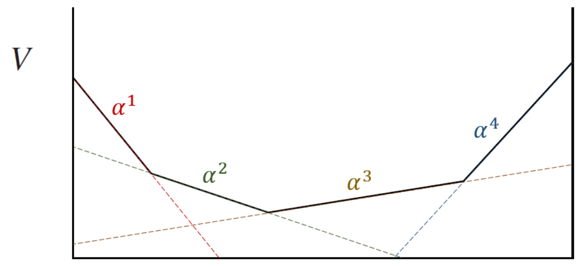


Figure 2.7: Vector distribution in a belief space.

An optimal policy can be calculated by the optimal value function which can be computed by iterating a number of stages. Each stage constitutes a step into the future. Assuming a planning horizon of $h > 0$, a value function V_n at stage n may be parameterized by a finite set of vectors $\{\alpha_n^k\}, k = 1, ..., |V_n|$. The utility value of a belief b is given by [24],

$$V_n(b) = \max_{\{\alpha_n^k\}_k} b \cdot \alpha_n^k, \tag{2.19}$$

where $(\cdot)$ denotes the inner product, and each vector corresponds to a immediate reward function for each action, that is, $\alpha_n^a(s) = R(s, a)$.

For a given initial value function and vectors, it is possible to compute vectors of steps further away and updating the value of each utility value function. The process is repeated until an equilibrium point is achieved. The iteration step (Bellman update) is calculated by the expression (2.20).

$$V_{n+1}(b) = \max_{a \in A(s)} \left[b \cdot \alpha_0^a + \gamma b \cdot \sum_o \operatorname*{argmax}_k \left[b \cdot \sum_{s'} O(s', a, o) T(s, a, s') \alpha_n^k(s') \right] \right] \tag{2.20}$$

Essentially, both in the case of MDPs and POMDPs, the Bellman update can be written using a backup operator H. A more general equation to denote a value backup is given by,

$$V_{n+1} = HV_n \tag{2.21}$$

2.2.3.3 Point-based value iteration algorithms

Planning over the complete belief space of an agent can be very difficult as computation of POMDPs in large state spaces is computationally demanding. A more efficient approach consists of planning only on a limited set of beliefs that have been sampled by letting the agent randomly interact with the environment. Instead of computing all the belief points in the state space, only a finite small set of reachable points are computed.

One point-based method for POMDP planning is the Point-Based Value Iteration (PBVI) algorithm [25]. PBVI approximates the value function by selecting a finite small set of representative belief points and tracking the value merely for those points. PBVI is similar to the value iteration algorithm as it also uses vectors to divide the belief space into regions, but it does not compute each vector in the same way. Instead, the algorithm uses a finite set of belief points $B = \{b_0, b_1, ..., b_q\}$ of the state space, and initiates a separate α vector for each selected belief point. Then, it repeatedly updates the value of that vector until it converges. Knowing the value of the vectors, the value function of a belief can be computed in the same way as in the value iteration algorithm, using the expression (2.22).

$$V_{n+1} = \widetilde{H}_{PBVI} V_n \tag{2.22}$$

Figure 2.8 shows how belief points can be chosen from the belief space. Only the represented belief points are used to calculate the corresponding vectors.

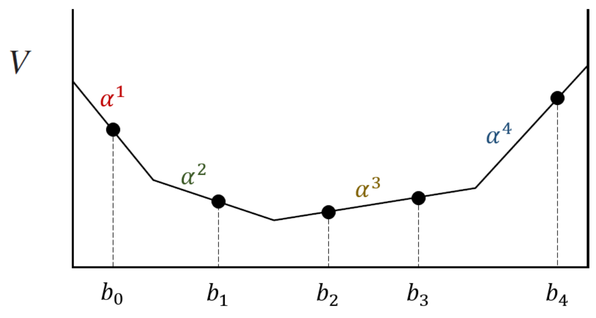


Figure 2.8: Value function using a PBVI algorithm.

Since PBVI algorithms only select reachable beliefs, instead of random belief as some algorithms do, belief selection outperforms multiple other methods [25]. The fact that the belief space is divided into regions but only belief points are updated, also improves the speed of the solver.

Other algorithms however propose different ways of belief sampling the environment. Perseus is one of them. Perseus is similar to PBVI as it also uses a sampled set of belief points to compute the value function and determine the optimal policy for an agent. The difference between the two methods is that Perseus implements a different and randomized backup operator $\widetilde{H}_{PERSEUS}$ [18]. This backup operator increases or maintains the value of all belief points in B. The idea is that by randomly sampling belief points from subsets of B, it becomes possible to obtain a function V_{n+1} that is an upper bound to V_n over B. In each backup update, the value of all points of the belief set is improved by merely updating the value and the gradient of the randomly selected sampling belief points. Thus, the number of vectors generated in each backup step will be small compared to the size of B. And with a smaller number of vectors to compute, the faster the algorithm reaches an approximation of the value function and therefore the solution [24]. The value function is obtained through (2.21) but with its own backup operator, that is,

$$V_{n+1} = \widetilde{H}_{PERSEUS} V_n, \tag{2.23}$$

making certain that $V_{n+1}(b) \geq V_n(b), \forall b \in B$.

2.2.4 POMDP tree representation

When depicting a problem, it is important to correctly define its model. In the case of a POMDP that means not only choosing the required sets of states, actions and of observations, but also understanding how they interact with each other. The combination between states and actions is important when formulating the transition models and reward functions, while the association of observations and actions is fundamental when composing the observation model. When choosing a reward or a probability, it is important to take into account the current state of a space state, the taken action of an action space and the received observation of an observation space. For a better explanation of modelling a problem, the following example is provided.

One example could be a problem with one set of states $S = s_1, s_2$; one set of actions $A = a_1, a_2, a_3$; and one set of observations $\Omega = o_1, o_2$. To construct the observation model, for instance, it is necessary to define the probability of observing o in state s' after executing action a. This corresponds to the conditional probability distribution $P(o|s', a)$. One probability value for this problem could be $P(o = o_1|s' = s_1, a = a_3) = 0.6$. In this problem, the total number of probabilities to be defined is twelve. This problem considers small spaces and few variable, so the number of probabilities is low which does not involve an increase of the computation costs when planning and solving. However, the problem becomes more difficult to solve as the number of variables and states increases. Thus, the way the problem is represented is of extreme importance in avoiding major problems. Ideally, the problem should be portrayed in a way that minimizes the number of probabilities to be represented, that is, the number of cases taken into consideration. Now the following four probabilities are known:

$P(o = o_1|s' = s_1, a = a_3) = 0.6$, $P(o = o_2|s' = s_1, a = a_3) = 0.4$, $P(o = o_1|s' = s_2, a = a_3) = 0.6$ and $P(o = o_2|s' = s_2, a = a_3) = 0.4$.

When analysing the four probabilities, it is possible to realize that the state s' has no influence in the probability of observing o_1 or o_2 when the taken action is a_3. Therefore, these four probabilities can be shortened into two, those being:

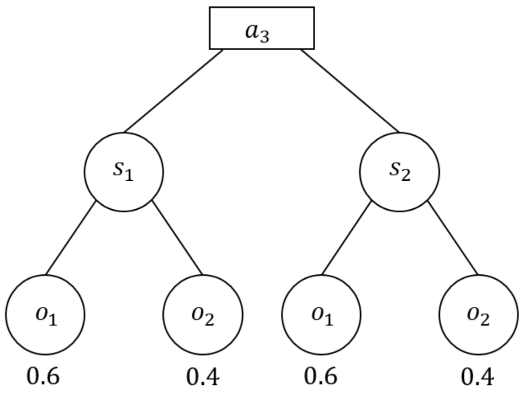
$\forall x$, $P(o = o_1|a = a_3, x) = 0.6$ and $P(o = o_2|a = a_3, x) = 0.4$.

One way to represent this excerpt of the observation model is through a probability table.

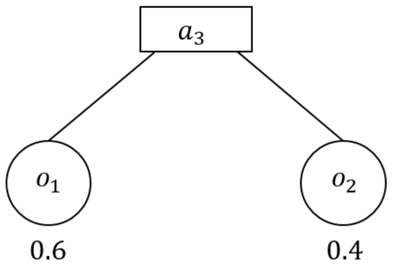
Ω	S	A	$P(o\|s', a)$
o_1	s_1	a_3	0.6
o_2	s_1	a_3	0.4
o_1	s_2	a_3	0.6
o_2	s_2	a_3	0.4

Table 2.1: Probability table of an example of an observation model.

The disadvantage in using a probability table is that every possible case needs to be represented in the table. In the example's observation model, the four probabilities have to be inserted in the table since the information of the states, actions and observations are required. A better way of representing the model is a decision tree. In a decision tree, it is possible to perform pruning, that is, remove branches of the tree. Another possibility is to switch leafs in the tree to accomplish pruning in the most efficient way possible. Figure 2.9 displays a way to represent the same decision model presented in Table 2.1 and also shows how the decision tree can be pruned.



(a) Before pruning.

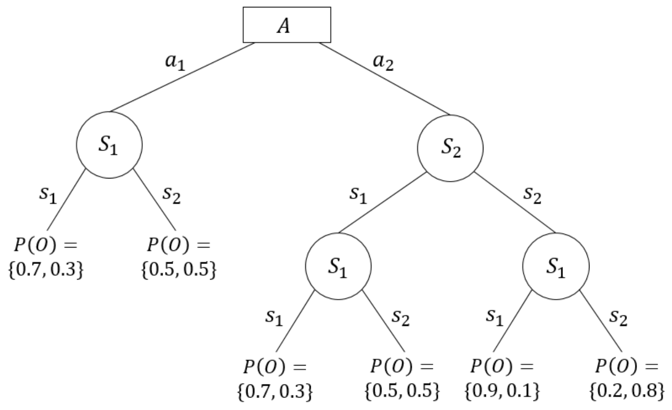


(b) After pruning.

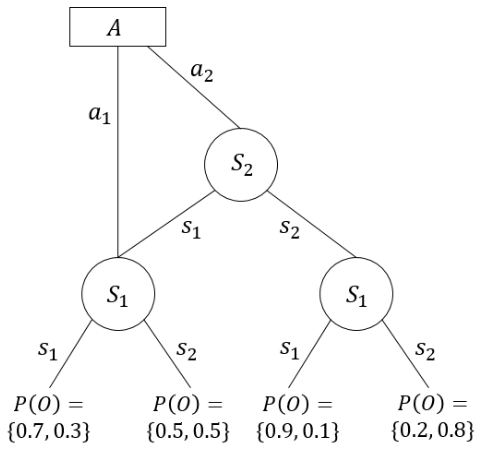
Figure 2.9: Decision tree of an example of an observation model.

Since the states of S have no influence in this excerpt of the observation model, using pruning the states can be "removed". Consequently only two nodes are considered and the number of probabilities is reduced from four to two. This means that a decision tree enables to represent models of a POMDP in a way that reduces the overall number of probabilities and rewards. Using a decision tree, with an increase of the amount of variables, the problem does not grow as rapidly as in the case of a table. Therefore, POMDP representation with a tree makes the problem less computationally expensive and easier to solve.

An even more efficient manner to display a problem is using an Algebraic Decision Diagram (ADD). ADDs are symbolic representation of weighted directed graphs, whose leafs may take any value of a set of constants instead of only considering boolean values [26]. One advantage of ADDs is that this type of diagrams uses independence between variables to provide the possibility of removing entire branches and reallocates certain leafs of the tree in order to aggregate them. A different example of an observation model is used to further explain the advantage in representing a problem as a ADD. The example is displayed in Figure 2.10 and considers a problem with two sets of states $S_1 = s_1, s_2$ and $S_2 = s_1, s_2$; one set of actions $A = a_1, a_2$; and one set of observations $\Omega = o_1, o_2$.



(a) Decision tree.



(b) Algebraic Decision Diagram.

Figure 2.10: Representation of a decision tree as an Algebraic Decision Diagram.

In this example, the usage of a decision tree is already an advantage over a probability table. The left section of the tree does not contain the state variable S_2 (pruning occurred) since it has no influence on the probability of perceiving each observation. The total number of probabilities of the model is thus reduced from sixteen to twelve. The problem may however be represented in a more condensed way using an ADD. This representation allows to reconnect the branch of $A = a_1$. The resulting graph contains the same meaning but has fewer nodes. The number of probabilities is once again reduced, this time to eight.

Note that the tree representation in Figure 2.10 is different from the one in Figure 2.9. This is due to the fact that the second example contains a higher number of probabilities and using the same tree representation, as in the first example, would make the graph in Figure 2.10 wider and therefore more difficult to read and comprehend.

Chapter 3

POMDP Design

To accomplish the goal of human-aware task planning under uncertainty, a model of a POMDP task was developed and is presented in Chapter 3. The solving of that POMDP was made with the Symbolic Perseus solver. Symbolic Perseus[1] is a version of Perseus that uses ADDs as the underlying data structure to deal with large factored POMDPs. Symbolic Perseus is an improvement of Perseus where α vectors are represented using ADDs. The main idea is that ADDs join entries of vectors that are identical [27]. In consequence, the number of α vectors computed is lower, which means the algorithm performs fewer operations when solving a problem. The initial policy is determined using the Q_{MDP} method. The way Symbolic Perseus solves POMDPs, evaluates and executes policies is explained in Chapter 4.

3.1 Task definition

In order to achieve the proposed goal, a task was developed and tested. The main purpose of this task is for an agent to interact with a human and make optimal plans to execute an action that guarantees the best possible outcome. The developed task takes advantage of the network of devices that constitute an NRS. This network of devices allows a robot to perceive information and act on it. Information is perceived as a set of observations with an associated uncertainty, and executed actions depend on the information received and the agent's goals. The chosen actions and observations were decided according to the capabilities of the sensors and actuators of the NRS. Among the multiple functions of the used NRS are: sensors for localization and visualization of the environment as well as actuators for navigation and speech.

The objective of the developed task is for an agent, in this case an NRS constituted by a robot and a surveillance camera, to predict the behaviour of people and act accordingly to keep them from moving or staying in a specified location. A person moves freely in an environment which contains restricted/dangerous areas where the person should not move into. When a person moves to one of those areas or evidences the intention to, the robot should move to the corresponding place and verbally advise the person that the place is restricted/dangerous and should not be visited. A person shows intention of moving to a place, by continuously moving towards it. If a person is moving at a higher speed, it is also more likely to be moving somewhere. This means that the robot needs to predict the behaviour of each person to a degree of certainty that allows the robot to know how to act. The person's behaviour is analysed using the robot's sensors and a camera attached to the ceiling of the indoor environment. When choosing how to act, the robot takes into consideration two main goals: either informing a person that they are in, or moving to, a restricted/dangerous place; or letting

[1]`https://cs.uwaterloo.ca/~ppoupart/software.html`

the person roam with no restrictions since it has no intention to going to a restricted/dangerous area. It is highly relevant to note that when no person is near a restricted/dangerous area but the robot is far from every restricted/dangerous spot, the robot should navigate to an equidistant position from all restricted/dangerous areas. This would enable the robot to reach a restricted/dangerous place faster and inform people that they should not be or go there.

This task has multiple real world applications in indoor environments. The main possibilities for the task consist on protecting people from dangerous situations, in particular children, elderly and people with some sort of mental impairment. Children do not have the same concept of danger as adults and require supervision. It is of extreme importance to prevent children from moving to places such as fireplaces and steps in homes or buildings. Accidents are more likely to occur particularly in situations where the number of children is considerably higher than the number of adults. Robots may be used to prevent children from moving or staying in dangerous places and provide assistance to adults in charge of supervising children. The task may also be applied to prevent adults, in particular elderly people who may not be in possession of all faculties, from entering restricted or hazardous places in hospitals such as radiation or operating rooms. This kind of prevention is essential but often the hospital staff is occupied which makes it hard to guarantee. Robots can be helpful in this type of prevention.

3.2 General model

The task is formulated with the POMDP framework. As explained in Chapter 2, a POMDP model is constituted by a set of states, a set of actions, transition models, reward functions, a set of observations and observation models. When modelling the task's POMDP it is necessary to define all these parameters. While the purpose of the task is to prevent a person from moving to a place in an indoor environment, there may be multiple people in that environment as well as various restricted/dangerous areas. The model of the task considers an n people and r restricted/dangerous areas.

The model contemplates two state variables for every person, these two variables being the position and movement direction, extracted from a person's velocity, of each person. The model also takes into account the position of the robot and its navigation actions. Finally, both the robot's and the people's position as well as the people's velocity are observed and interpreted by the model. Velocity is a physical vector quantity that is defined by two factors: magnitude and direction. Velocity's magnitude corresponds to speed, while direction corresponds to the direction of motion. This means that, if a person is moving to north at 3 meter per second, 3 meters per second is speed and north is direction.

The supposed behaviour to correctly execute the task is guaranteed with the model's reward functions. Restricted/dangerous areas and the equidistant place are defined using rewards, as further explained in the reward functions of the model in 3.2.3.

Considering n people in the environment, the general framework has the following terms:

- a state variable for each person's movement direction, $D1, ..., Dn$;
- a state variable for each person's position, $P1, ..., Pn$;
- a state variable for the robot's position, R;
- an action variable containing the possible navigation actions for the robot to perform, A_{Move};
- a reward function, with costs for every action taken and where rewards are assigned according to the robot's position and people's behaviour;
- an observation variable for each person's velocity, $O_{V1}, ..., O_{Vn}$;

- an observation variable for each person's position, $O_{P1}, ..., O_{Pn}$;
- an observation variable for the robot's position, O_R.

The connection between all variables constitutes the POMDP model for the given task. This model is presented in Figure 3.1.

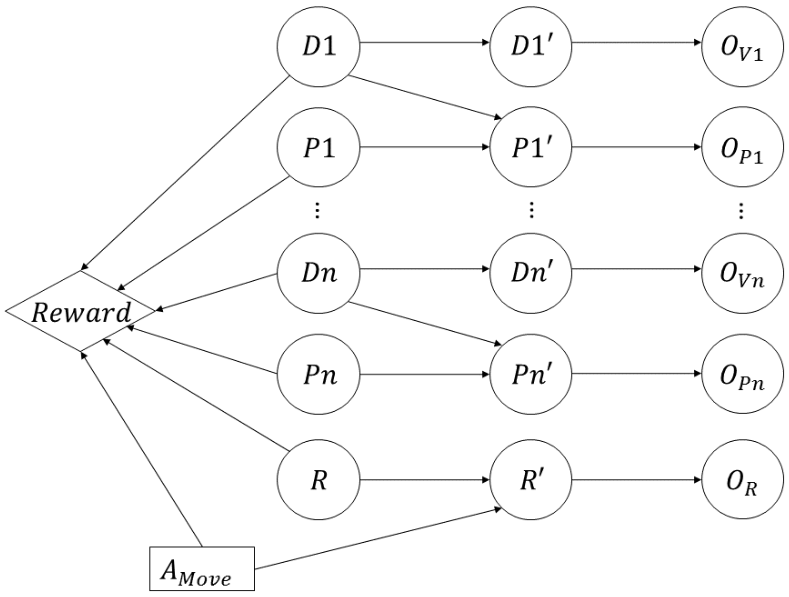


Figure 3.1: Graphical representation of the general POMDP framework to perform optimal planning on the described task.

The simplest case in the task is to consider only one person and one restricted/dangerous area, that is $n = 1$ and $r = 1$. Since, in the general framework of the problem, the number of people and the size of the environment state space are not fixed, the modelled problem is scalable. However, increasing the amount of people and areas expands the complexity of the model. Each person has two associated state variables which means that for each person considered, the complexity of the problem increases. The number of restricted/dangerous has no direct influence on the model's complexity. However, by considering a high number of restricted/dangerous areas, it is also preferable to have a considerable amount of possible positions, that is, a large environment state space. By increasing the people's and robot's number of possible positions, the complexity of the problem also increases. Despite the fact that the problem may be of high computational complexity when a large number of state and observation variables exists, it is possible to use a high enough amount of variables and possible states for the problem to be adapted into plenty of real life situations.

Each component of the POMDP model is important to guarantee the designated behaviour and the accomplishment of a goal in an optimal way. A description of the model is fully explained in sections 3.2.1 through 3.2.4.

3.2.1 States and transition models

The model needs information about the robot's position in the environment. Assuming the environment's map contains p positions, the robot may be in any of those p positions. It is also necessary to consider one extra position which is the possibility of the robot not being in any of the p positions, essentially, out of the map. Therefore the model considers one state variable for the robot with $p + 1$ possible states. The state of the robot not being in the map exists in order to prevent the robot from leaving the map. This behaviour is further explained in the reward function of the model.

Since it is necessary to predict if a person is moving towards a specific area, it is necessary to extract information from every person in the environment. The required information consists of checking every person's place and moving direction. Just as it happens with the robot's state variable, a person may be in any of the p positions of the map's environment. Positions are seen as cells of the map. Cell boundaries are defined using two dimensional Cartesian coordinates (x, y). There is also an extra state in case no person is detected. This means that for each person position state variable, there are $p + 1$ possible states. The moving direction of a person has five possible states, four of them correspond to the cardinal directions and one considers that the person is not moving.

Since there is a state variable for the robot, a state variable for the position of each person and a state variable for the moving direction of each person, the total number of state variable in the model is $2n + 1$, where n is the number of people considered in the model. Table 3.1 presents the model's state variables and corresponding possible states.

Variables	Robot (R)	Person Position (P)	Person Direction (D)
States	$1, ..., p$, Out of the map	$1, ..., p$, No person	North, West, East, South, Not moving

Table 3.1: State space in the POMDP model.

The transition model for the robot's position depends on two factors, the robot's previous belief state and the action taken. Since the actions are navigation actions, each time an action is carried out, the robot changes its position in the environment, unless the robot is not required to move. Assuming the policy is to move, the next state will likely (higher probability) be the one to where the robot should navigate according to the action taken. If the robot moves to east, for example, its position will likely change to the adjacent east position. There is also the possibility that the robot is unable to move due to an obstruction in the environment or due to a problem with its actuators. In that case, the robot will stay in the same position. The probability of the robot staying in the same position, in the next time step, when the policy is to move is low. If the action required is not to move, the robot stays in the previous state.

Since the action taken has only an impact on the robot's position, the action variable has no influence on the transition model of each person's state variables. Even if a person should stop moving to a restricted/dangerous area following the robot's advice, the person may still choose to disobey the robot which is why the person's state variables cannot depend on the action taken or the robot's state. State variables of different people also do not influence each other, meaning that the behaviour of a person has no impact on another one.

The transition model of a person's moving direction only takes into account the previous belief state. This means that the transition state of this variable considers it more likely for the person to keep moving in the same direction. There is still a relatively high probability of changing the movement direction. The lowest transition probability consists of the person moving in the opposite direction in the next time step. Due to the possibility of a person's movement changing, between time steps, since the person may modify its behaviour while the robot's action occurs, all states in the transition model have a non-zero probability.

The transition model of a person's position considers the previous belief state of the person's position and moving direction. If a person is not moving then it is likely that they will stay on the same position. If people are moving towards a certain direction, there is a higher probability that they will either stay on the same position or on a position located in that direction with adjacent positions being of higher probability.

The reason a person may be moving without leaving the current position is because a position is defined as a surface area of the environment, as further explained in Chapter 4, and therefore the person may not leave that area despite moving. If a person moves to a direction where there is a wall, it is more likely that they will either stay on the same position or on an adjacent position.

3.2.2 Set of actions

Actions are important since they enable an agent to reach its goal and complete tasks. The possible actions of the model are confined in one variable called A_{Move} which is presented in Table 3.2.

Variable	Action (A_{Move})
Actions	North, West, East, South, Do nothing

Table 3.2: Action space in the POMDP model.

The actions are of navigation and may possibly change the robot's position and thus influence the environment. Only navigation actions are considered since the task is a navigation task where the robot needs to prevent a person from entering an area. Four actions help the robot to move to a desired place while one action allows the robot to avoid moving. The navigation actions are named as the four cardinal directions representing the direction to where the robot should move. So, if the generated policy is "West", the robot should move to the west adjacent position. There is no need of moving unless a person shows intention of entering a restricted/dangerous area or in case no person is approaching a restricted/dangerous place and the robot moves to the equidistant spot from all restricted/dangerous places. The navigation actions are designed to make the robot move continually through out the environment instead of moving directly to the restricted/dangerous area. This happens because the person may decide not going into that area and in this case the robot can stop moving towards it.

Additional actions may occur. However, these do not have to be considered in the model. The NRS may act when sensing by, for example, rotating while staying in the same place to improve sensor readings. These actions are not considered in the model as no planning is required. Sensing actions are only required for the agent to observe the environment. Furthermore, once the robot reaches a restricted/dangerous area, it speaks with a person telling him/her to not stay there. This speech action is carried out immediately after the navigation action that allows the robot to reach the restricted/dangerous area. This means that the robot speaks when it reaches a specific waypoint with no planning required to perform the speech action and therefore this speech action is not included in the model. The speech action is used to improve the task's HRI as the interaction should be as realistic as possible. While the position of the robot is influenced by the actions of the model, the orientation is not considered in the model and does not require planning.

3.2.3 Reward function

Rewards are useful in achieving the goal of a task. When developing the reward function for a problem, it is important to think ahead and take into account that an action that may seem inadequate in a time step, may help the agent achieve its goal more efficiently in the long term. In other words, rewards are assigned according to what would be the best action in a current state that will help the agent accomplish its task in a more satisfactory way in the long term. Once the reward function has been determined and the planning carried out, the chosen policy is the one to maximize the cumulative function (2.1). So, when selecting rewards in the POMDP model, positive rewards can be associated to cases of benefit while the opposite happens to negative rewards.

Since the task requires navigation, the robot needs to move in order to reach the restricted/dangerous place to protect the person. However, the robot should only move when the person evidences intention of moving there or when they are already there. The robot should move as minimum as possible and only when necessary. It is important to avoid unnecessary navigations to protect battery life, reach a restricted/dangerous place as soon as possible and to prevent possible obstruction to people in the room. To control excessive robot navigation, the reward function attributes a cost to actions. Each moving action has a small negative reward, a cost, and not moving has no reward.

Rewards are also supplied according to both the robot's position and a person's behaviour. The agent receives a high positive reward when it reaches a restricted/dangerous place, if a person is near the place or is somewhat near and moving towards it. The model could only consider the agent receiving a reward when a person is at the exact restricted/dangerous spot. However, in the real world, a robot is not able to move as quickly as a human which means the human will reach the spot before the robot. This reward attribution allows to make sure that once a person is at an unsafe distance from the restricted/dangerous spot and moving towards it, with a certain degree of confidence, the robot will immediately start moving to that same spot to prevent the person from staying in that location for a long time thus decreasing the chance of jeopardy.

It is also important to ensure that the robot is at an equidistant from all restricted/dangerous areas, when no person is displaying intention of moving to a restricted/dangerous location. In fact, the robot should only travel to reach restricted/dangerous spots or the equidistant one. This behaviour is assured by the agent receiving a negative reward, that is a penalty, if it is in any position other than a restricted/dangerous place or the equidistant one. In the equidistant position, the agent receives a reward of 0. If the agent is at a restricted/dangerous place but no person is displaying intention of visiting that place, it also receives a penalty. This essentially means that the robot travels along the environment receiving negative rewards for its positions and moving actions in order to reach the equidistant point where it will not receive further penalties, or to reach a restricted/dangerous area to receive a high positive reward. This positive reward is much higher than the absolute cost of each moving action and the penalties of being in an undesired location. In case multiple people are considered in the model and multiple people move to different restricted/dangerous places, the robot will proceed to the area that is either closest or more likely to be visited by someone, according to those people's positions and movement direction. Finally, it is important to avoid the robot leaving the environment. This is done by giving a penalty with a high absolute value, if the robot is ever outside of the map (by being on an edge of the map and exceeding that edge in a navigation action). The negative reward of leaving the map avoids this situation. The overall behaviour ensures a higher reward in the long term.

3.2.4 Observation models

States affect perceived observations. If a variable is in a specific state, it is more probable to receive an observation that corresponds to that state. Therefore, when an observation is made, the model considers that the state variable is more likely to be in the same state as the perceived observation. Since an observation has, usually, an associated uncertainty, the probability of being correct is below 1. At each time step, the belief state for each variable changes according to the transition model and the observation received.

There is an observation variable made for each state variable. There are two observation models for each person, one for position and one for velocity, and one observation variable for the robot's position. Therefore, the total number of observation variable is the same as state variables in 3.2.1, which is $2n+1$ assuming n people in the model. Table 3.3 displays the model's observation variables and corresponding possible observations.

Variables	Obs Robot (O_R)	Obs Person Position (O_P)	Obs Person Velocity (O_V)
Observations	$1, ..., p$, Out of the map	$1, ..., p$, No person	$\overrightarrow{v_1}, ..., \overrightarrow{v_i}$, Not moving

Table 3.3: Observation space in the POMDP model.

Each variable has multiple possible observations, each with a certain probability value depending on the model's state. When choosing the probability of an observation being made, values between 0 and 1 can be set. However, when the model is not deterministic, exact values of 0 and 1 must be avoided since this restricts the problem. A probability of 1 assumes that a certain observation can be completely accepted. A probability of 0 for any observation, in a given scenario, means that it is impossible for that observation to occur. However, if for some reason that observation were to be received in the same specific scenario, the agent would not be able to act due to not having planned that observation for that scenario. Therefore, not planning every situation could result in the model getting stuck and not knowing how to act. The presence of uncertainty means that any observation can be made in any scenario, no matter how unlikely it may seem.

When the robot navigates through the environment, its position changes. At each time step, the robot either moves to an adjacent position or stays in the same one. In the case of the robot's position, the observation is deterministic. That is because the robot always moves in the direction where it should move, or stays in the same position. This behaviour is in fact inserted in the robot's transition model. It is impossible for the robot to be observed in a place which was not its target when it started navigating. The navigation system implemented in the robot ensures that the robot moves to the supposed goal unless the robot cannot do it due to a possible obstruction or a failure in the actuators.

There is the need to check the position of a person at each time step. Since a person moves in real time and may act faster than the robot, the possibility exists that the transition model, which considers a person's previous moving direction and position, is not accurate. This means that the observation of a person's position in the environment is essential. The observation model for the position of a person considers that if a person is in a given state (position), there is a high probability that the observation received is being in that state. However, it exists a low probability that the person is seen in a different position, resulting from bad readings in sensing or false positives in people detection. If they are seen in a different position, it is more likely to be near the correct one than far away. If there is no one in the environment, there is a high probability than no one is detected. There is still the possibility of a person being detected due to the existence of false positives.

While a person's moving direction is a state variable used to determine if a person has intentions of moving to a restricted/dangerous place, the observation made of a person's movement also takes into account the person's speed, that is, an observation of a person's velocity is made. The observation variable of a person's velocity has multiple observations having for each of the four cardinal directions a number of possible speeds. Speed is discretized in classes. The observation model for a person's velocity considers that, if a person is moving towards a direction, it is more likely that they are moving at a higher speed. When someone is moving to a place, they usually walk at a high and constant speed. An observation of a person moving towards a direction, decreases the belief that they are moving to a different one, particularly the opposite direction (east is opposite to west while north is opposite to south). So, if a person's movement direction state displays a certain direction, the probability of moving in that direction is higher than others, and the probability of moving at a higher speed is also higher. If no person is detected or if a person is not moving, it is more likely observing no movement which is indicated as "Not moving". When a person is not moving, the probability of moving at a certain speed is low. As speed increases, the lowest is the probability.

Chapter 4

Implementation

In Chapter 3, a task and the corresponding POMDP general model were presented. With the general framework designed, it is important to implement and solve the model in order to create the decision system of the task. Integrating the decision system with the agent and the environment is crucial. Finally, acting and perceiving need to be enabled in the NRS. This chapter describes the implementation and integration between systems.

4.1 NRS and indoor environment

The NRS used in experiments consists of a robot called MBot and a camera network system. The robot is used in SocRob@home[1] projects and competitions as well as in the MOnarCH[2] project.

MBot, Figure 4.1, is constituted by a network of devices that give it versatility in ways of sensing and acting. The robot has two main parts, a body and a head. The head can turn sideways; has LED's in the eyes, cheeks and mouth areas in order to express emotions; a microphone on top of the head for voice recognition; and a camera on top for image detection. The body contains two motherboards with i7 processors; a touchscreen monitor displaying the operating system interface; a Cyton Gamma 1500 Robai arm, with 7 degrees of freedom and a $68cm$ reach, attached to the left side of the body; a laser scanner for detecting and avoiding obstacles while the robot is navigating; and a four-wheel omnbidirectional mecanum drive.

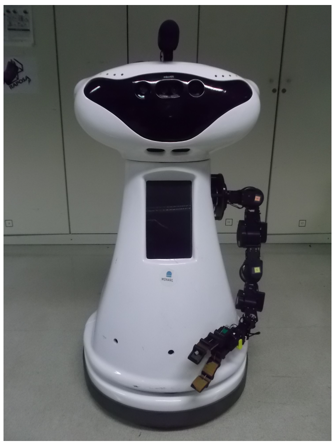


Figure 4.1: The robot MBot.

[1]http://socrob.isr.tecnico.ulisboa.pt/dokuwiki/doku.php?id=socrobhome:socrobhome
[2]http://monarch-fp7.eu

The camera network system is constituted by surveillance cameras on the ceiling of the Laboratório de Robótica Móvel (LRM). In the performed experiments, only one surveillance camera was used. The camera's model is the AXIS P1344 which has, among other specs, RGB color, varifocal lens of $3-8mm$, resolution up to 1280×800 pixels and live video streaming. The camera's live feed can be accessed via IP address. A picture of the AXIS camera is presented in Figure 4.2. The camera allows the visualization of the environment and is essential to observe a person's behaviour in it.

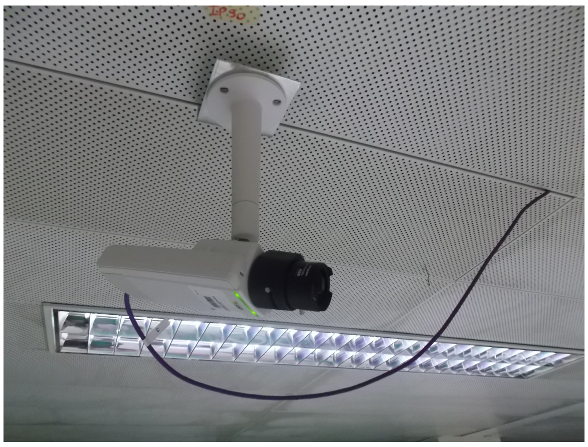


Figure 4.2: Surveillance camera placed on the ceiling of the Laboratório de Robótica Móvel.

The HRI occurs in an indoor environment. The environment where HRI experiments were conducted is the LRM, at ISR, which is located in the 8th floor of the north tower at Instituto Superior Técnico. Experimental work includes navigation actions in the LRM indoor environment. This means the robot needs to have previous knowledge of the environment in order to navigate to waypoints, avoid walls and prevent from leaving the environment. This can be achieved using a 2-D occupancy grid map of the environment. The map was created from laser and pose data collected by the mobile robot. The map used in the conducted experiments can be visualized in Figure 4.3. The map was created by the SocRob@home team by performing manual teleoperated robot navigation and collecting data from the robot's laser and pose. To achieve this a ROS package called gmapping, which provides laser-based Simultaneous Localization and Mapping (SLAM), was used.

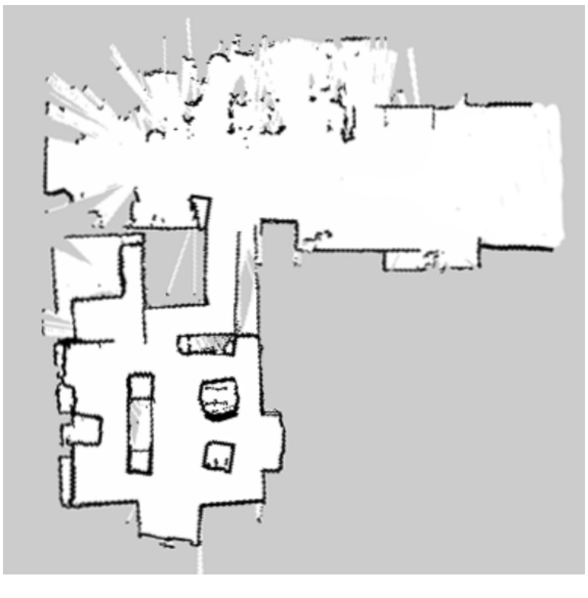


Figure 4.3: 2-D occupancy grid map of Laboratório de Robótica Móvel.

The POMDP model explained in Chapter 3 considers the positions of the robot and the person in the indoor environment. The positions of the map, presented in Figure 4.3, are defined by x, y Cartesian coordinates. However, there is a large number of x and y values that indicate all possible positions in the map. One way of reducing the number of possible positions on the map consists of a discretization of the environment. The idea is that it is possible to divide the environment into relatively small regions, or cells, and then check the cell where a person is. The robot itself, also moves between these cells every time it executes a moving action.

Even though the environment is discretized as cells, using the entire lab space of LRM would force to use a high number of cells to represent the environment. This is not favourable since the higher the number of cells, that is, the number of possible positions for the robot and a person, the larger the state and observation spaces for the people's and the robot's position variables. Big state and observation spaces are not feasible since they increase the computation time when solving the POMDP. Another possibility could be using the entire lab space and keeping a low number of possible states for a person with cells of large dimensions, thus not increasing the complexity of the model. However, some zones of the lab are not covered by surveillance cameras and in those zones, people detection would not be possible.

Another reason in avoiding a large number of cells is that the robot does not move as quickly as a person and it stops in every cell to access the best action to execute next. If a person is seen moving to a restricted/dangerous place it is important that the robot reaches that place quickly enough to alert the person for the danger of that place.

Finally, observations of a person's speed is detected using the robot's laser scanner which has a limited range. Using the entire lab would imply a person's speed not always being detected and therefore a higher uncertainty would exist in observing a person's behaviour.

The region of the LRM used in the experimental work is selected with a rectangle in Figure 4.4.

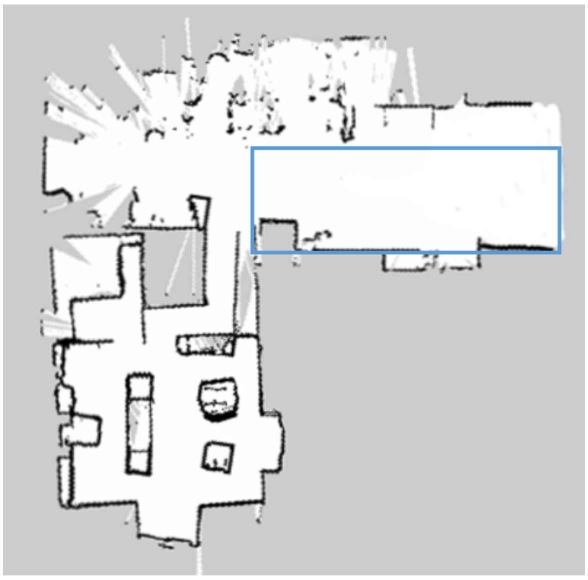


Figure 4.4: Section of the map in Figure 4.3, where experiments were conducted.

For the interaction between the NRS and people to occur in an indoor environment, it is necessary to implement the general model framework explained in Chapter 3, into a particular case study to conduct experiments.

4.2 Case study model

The established model in Figure 3.1 is designed to be scalable to any number of n people and r restricted/dangerous areas. POMDP planning may take quite some time, especially in cases of problems with high complexity. The case study model, created from the general model in Chapter 3, was designed having this in consideration. The case study model considers one person in the environment, $n = 1$, and twelve positions, $p = 12$. The twelve positions consist of twelve cells that represent the environment in Figure 4.4. When a person is an area of the environment that belongs to a cell, that person is seen as being in that cell position. The cells are numbered from 1 to 12 and therefore the person's and robot's position, if in the map, is also indicated as being between 1 and 12. The robot navigates between waypoints, positioned in the map, each inside a cell. Figure 4.5 contains the way the environment was divided into twelve regions.

12	10	8	6	4	2
11	9	7	5	3	1

Figure 4.5: Cell division of the environment's map. Each cell represents a region of the map where a person may be present.

Two versions of the same model were developed. One with one restricted/dangerous area in the environment, $r = 1$, and the other with two areas, $r = 2$. The difference between the two versions stands only on the reward functions since it is the reward function the one responsible in assigning a purpose to certain areas. When there is only one restricted/dangerous area, that area corresponds to the region of cell number 1. In case of two restricted/dangerous areas, these areas correspond to cells number 1 and 12. In the second case, the region of cell 7 is the equidistant spot. When two restricted/dangerous places are considered, it is useful to have an equidistant spot as it guarantees that the robot always tries to move in a way to reach a restricted/dangerous area as quick as possible and alerting the person promptly. 7 is three robot movements away from both restricted/dangerous areas, as the robot only moves horizontally and vertically. In the case of only one restricted/dangerous area, an equidistant spot is not necessary. However 7 is still seen as a special location. The robot should be positioned in a place that does not bother people, that is, it should not be an obstacle. Therefore, when no one is moving to the dangerous/restricted place, the robot should be at 7 instead of potentially blocking passage in a different position. This version could also see the robot stay permanently at 1, owing to the fact that it is the only restricted/dangerous area of the environment. However, that place does not necessarily have to be restricted to every person. If the robot is always at 1 and blocks passage to the restricted area, it could end up hindering the access to 1 of authorized personnel.

Using the mentioned parameters of n, r, p and applying them to the general POMDP model in Chapter 3, the case study model becomes the one in Figure 4.6. The model is designed to consider only one person since the task can be achieved this way and the robot's behaviour would be identical with more people. Using one person in the model is enough to attain HRI. Using twelve map positions also guarantees a good experimental setup when using one or two restricted/dangerous areas since the number of positions is enough to evaluate the behaviour of a person over time and consider that a person may be near or far from a restricted/dangerous area.

The model has a state space of 31 states, divided in three state variables, an observation space of

39 observations, also divided in three variables, and 5 possible actions, contained in one action variable.

Transition, observation models and reward functions are defined using an ADD. This is due, not only to the benefits of representing a problem in this form, presented in Chapter 2, but also to the fact that the solver Symbolic Perseus, takes advantage of ADDs to perform value iteration in fewer steps.

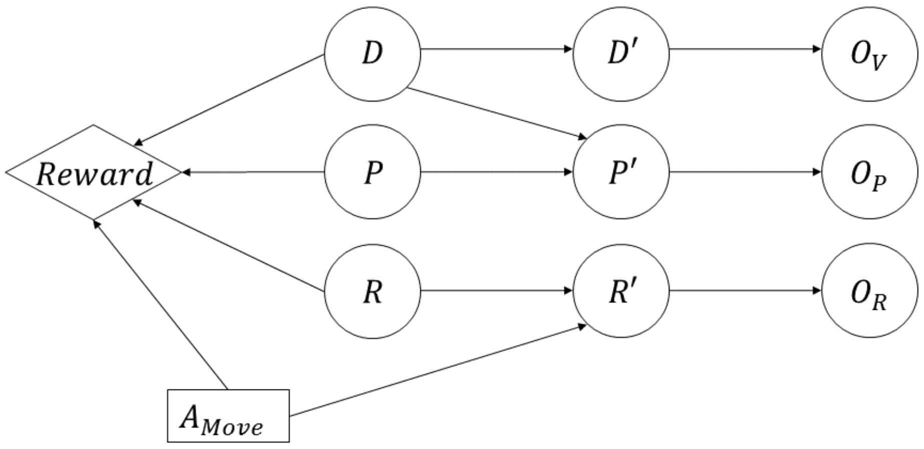


Figure 4.6: Graphical representation of the implemented POMDP model to execute the task.

Actions in A_{Move} are either moving or doing nothing. The actions of moving are named after the four cardinal directions. The directions in the environment correspond to the orientation in the map in Figure 4.3 and therefore also to the orientation in Figure 4.5. This means that if the robot is at 9 and the action is "North", the robot will move to 10. If the action is "West", it will move to 11.

The velocity observation variable has both speeds and directions. Direction is one of the four cardinal directions. Speed is one of three possible speed classes: between 0 and 0.2 meters per second; between 0.2 and 0.7 meters per second; and over 0.7 meters per second. There is also the possibility of the person not moving or not being in the environment, in both cases an observation of "Not moving" is made.

The reward function assigns rewards depending on the robot's position, the person's behaviour and the executed action. For every navigation movement, the agent receives a reward of -0.2, in order to avoid unnecessary navigation. Other values of rewards for specific situations are presented in Table 4.1.

Robot	Person's Behaviour	Reward
7	———	0
Restricted cell	At the restricted cell	30
Restricted cell	At an adjacent cell from the restricted one and not moving away from it	30
Restricted cell	At a distance of two cells from the restricted one and moving towards it	30
Restricted cell	Different from the previous three	-10
Out of the map	———	-30
Other cells	———	-3

Table 4.1: Reward assignment according to the robot's position and the person's behaviour. Lines symbolize cases where the person's behaviour has no influence on rewards.

The agent receives negative rewards at most positions. This forces the robot to move to the equidistant place as it will stop receiving negative rewards once it is there. However, when a person is seen as moving to a restricted/dangerous region or is already there, the robot will also move to that region. While the agent will receive penalties for not being at the equidistant spot nor a restricted/dangerous place and for moving, once it reaches the desired place and alerts the person, the agent will receive a high positive reward. In summary, the agent is acting in a way that maximizes its cumulative reward in the long term which is one of the benefits of using the POMDP framework.

In the POMDP model, the chosen value for the discount factor γ is 0.99. Therefore, by the inserting this value in the expression (2.1), the cumulative function of rewards becomes,

$$E\left[\sum_{t=0}^{h} 0.99^t R(s_t, a_t)\right], \tag{4.1}$$

where $E[\cdot]$ denotes the expectation operator and h is the planning horizon.

A high discount factor ensures that rewards received in a distant future are still valuable which is important in problems with potentially long planning horizons.

While actual reward values are presented, the problem has substantial sized transition and observation models. For that reason, values assigned to all transition and observation probabilities in the model are not presented in the main body of the thesis. These models are fully exposed in Appendix A. The values chosen in the observation models were based on multiple tests in people detection and laser readings to evaluate how accurate the observations were. Rewards were chosen in a way that guaranteed the completion of the task by the robot.

4.3 POMDP formulation, solving and ROS integration

With the model in Figure 4.6 established, it is necessary to formulate it in as a POMDP text file that can be read by a solver. Formulating a POMDP as a text file can be quite difficult and time-constraint. In problems with a reasonable amount of variables, files can easily include up to thousands of lines to define all aspects of a POMDP. One better solution is to use a tool for editing and evaluating probabilistic graphical models, in particular POMDPs, with the help of a graphical user interface. The chosen tool to implement the model described in Chapter 3 as a POMDP file was OpenMarkov[3]. OpenMarkov is a software tool for editing and evaluating several types of probabilistic graphical models, learning Bayesian networks and performing cost-effectiveness analysis. Besides OpenMarkov, there are multiple software packages for designing and editing probabilistic graphical models, most presented in Kevin Murphy's list[4]. While many of these packages have graphical user interfaces, most of them are limited to building Bayesian networks and influence diagrams, not supporting POMDPs [28]. That is not the case with OpenMarkov which allows to build POMDPs. Using the OpenMarkov tool, the model in Figure 3.1 was implemented with the characteristics explained in Chapter 3. Files generated with OpenMarkov are in the ProbModelXML format.

Once the model is built as a POMDP, it is important to solve it in order to perform decision-theoretic planning and learning. OpenMarkov allows editing POMDPs but not their evaluation which is why a solver is required. The solver chosen was Symbolic Perseus, as mentioned in Chapter 2. The code of Symbolic Perseus was written in MATLAB and Java. The Symbolic Perseus algorithm, available online[5], is not compatible with the ProbModelXML format but the code was modified in order to also read that format.

Symbolic Perseus is used to perform offline planning. Offline planning consists of finding and evaluating solutions to a problem or task before actually execute them. In known environments, offline planning is preferable. In unknown environments, where unpredictable situations can easily occur, it is better to

[3]`http://www.openmarkov.org/`
[4]`http://www.cs.ubc.ca/~murphyk/Software/bnsoft.html`
[5]`https://cs.uwaterloo.ca/~ppoupart/software.html`

perform online planning. Online planning consists of revising a course of action during the execution of a task. Offline planning may take some time to compute prior to the execution of a task but requires no further computations while the task is carried out. Online planning does not require previous computation but since a course of action is revised while a task is carried out, the agent may take some time to know how to act while the task is executed. Since the proposed task occurs in a known indoor environment and the HRI occurs in real-time, offline planning is a better solution.

The created file with the POMDP problem is first parsed in Symbolic Perseus. The components generated from parsing consist of the information that describes the POMDP, such as all the variables and their possible states. Afterwards, an initial policy is determined using Q_{MDP}. Using this initial policy, it is possible to sample a number of reachable belief states. In the case study model, 300 belief points were sampled. Following the belief sampling, the Perseus algorithm, with ADDs, is used to perform point-based value iteration using the Perseus backup operator. At each Bellman backup update, the values of the belief points are improved and therefore so it is the value function and the optimal policy. This improvement, and the use of ADDs, enables the usage of a small amount of α vectors, instead of having an α vector for every belief point. In fact, while 300 belief points were initially sampled, most iterations only needed 10 to 30 α vectors. Having a lower number of α vectors means that the computation is faster. When solving the case study model, 50 Perseus iterations were performed.

Once the POMDP has been solved, it is possible to evaluate the policy graph generated by Symbolic Perseus. This can be achieved with the evalPOMDPpolicy function granted by the MATLAB code of Symbolic Perseus, provided online. The function simulates multiple belief states for all variables and executes the optimal action in each scenario. Finally, actions are executed with the tracePOMDPpolicy. The function receives observations, in the format of strings, inserted manually and outputs policies.

While the decision system, established using Symbolic Perseus, can be used to obtain results in MATLAB's simulation environment, with observations being manually entered in the system, that is not the goal of the thesis since it does not allow real world HRI. To accomplish human-aware planning, it is necessary to integrate the sensing and acting systems provided when using the NRS. Thus, the overall system used in experiments with the NRS can be divided into three subsystems: information perceived as observations; the solver Symbolic Perseus which provides decisions on how to act at each time step; and the execution of actions by the NRS. A description of the integrated system is displayed in Figure 4.7.

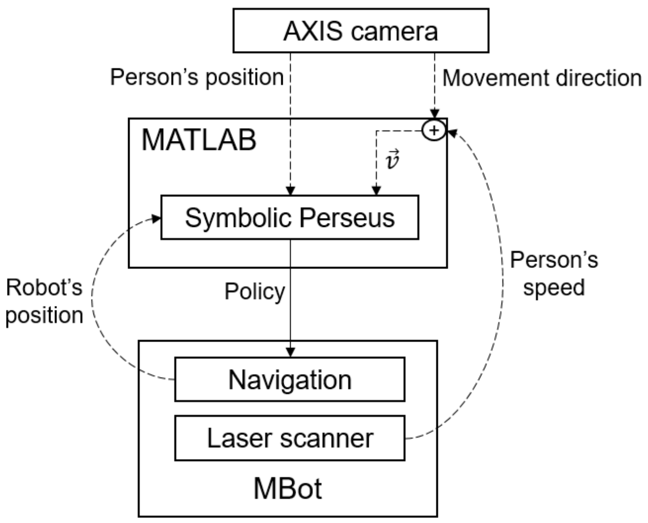


Figure 4.7: Connection between all subsystems. The observations (dashed arrows) perceived are sent to the solver which generates a policy (line arrow) and sends it to the robot as a navigation goal.

The three subsystems work independently but they have to communicate information. Just as Figure 4.7 shows, observations perceived by the NRS have to be sent to the decision system. The decision system generates a policy and sends it back to the NRS which acts accordingly. This means that a connection must exist between MATLAB and the NRS. The behaviour and functionality of the NRS occurs using ROS packages. The integration of the decision system with the NRS can be performed using the Robotics System Toolbox[6] which provides an interface between MATLAB and ROS setting up applications on ROS enabled robots and robot simulators. Using ROS topics it is possible to communicate between interfaces. ROS topics are named buses over which nodes exchange messages. Four topics were created for observations and one topic for the action. The observations are obtained in the sensing system. When each observation is obtained, it is published to a topic. In MATLAB, the four topics are subscribed and their content is provided as strings to the solver (the strings are inserted when the function tracePOMDPpolicy is executed). However the solver only takes into account three observations instead of four. This is due to the fact that the solver considers the velocity of a person and that observation has two components, speed and direction. These two components are published in two separate topics and then joined as a string sent to the solver. After receiving the three observations, the solver returns a policy that is published in a topic. That topic is then subscribed outside of MATLAB and the action is selected and executed by the robot according to the content of the subscribed topic. Only when the robot finishes its action will the solver read new observations and determine the next optimal policy.

Symbolic Perseus behaviour has been explained previously. The way the robot perceives observations and executes actions is explained in detail hereinafter.

4.4 Executing actions

Actions are executed according to a navigation system existing in the robot. The solver outputs an optimal action based on perceived observations that influence the belief state of variables. This action may require the robot to move. The robot moves between cells in the map, each cell containing a waypoint to where the robot should move. This waypoint is the navigation goal in each action and is defined by the x, y coordinates and the orientation in the world frame. The navigation system must be capable of choosing the best possible path to reach a waypoint. The system receives information about map data, the robot's pose in the map and sensor readings. From that data, it builds a costmap with information about obstacles in the world. A global planner using the A^* search algorithm finds the optimal path between the robot's current position and the navigation goal. Knowing the costmap and the global plan, a local planner yields velocity commands that should be executed by the robot. This results in a trajectory for the robot. As the robot moves, the costmap is updated and so is the global navigation plan. This overall procedure is attained with the ROS package navigation. In theory, when the robot reaches a waypoint, it should rotate in order to match the orientation of the waypoint. Since this takes some time and the orientation is not important in the developed task as a POMDP, the orientation of the robot when it reaches a waypoint is discarded.

When the robot reaches a restricted/dangerous area, it should inform the person that they should not be there. That is achieved using eSpeak[7]. eSpeak is an open source software that is able to synthesize text to speech.

[6]http://www.mathworks.com/hardware-support/robot-operating-system.html
[7]http://espeak.sourceforge.net/

4.5 Receiving observations

Observations are received from MBot's onboard laser scanner and from the surveillance camera. The laser is used to detect people's speed and for the robot's self localization. In the robot's self localization, odometry is also used. The camera is used to detect the position of people and to check their moving direction.

When the robot is moving, it uses Adaptive Monte Carlo Localization (AMCL) to check its position in the environment. AMCL is a probabilistic localization system for a robot navigating in a 2-D environment. Knowing the dimensions of a map and the robot's pose in it, AMCL uses a particle filter to track the robot's pose. During the robot's navigation, readings given by the laser scanner and the distance covered over time given by odometry, update the particles position estimating the robot's pose in the known map. While this contains an associated localization error, this error is quite small and the robot is always seen very close from its actual pose. Consequently, while the actual position may not be the waypoint's exact one, the robot will stay inside the correct cell once it finishes its navigation action. This makes the robot's observation deterministic. If the robot is unable to navigate, it will stay in the same position and, in that case, the observation of the robot's position will be the same as in the previous time step.

The task requires for the agent to predict future behaviour of a person, based on previous observations of the person's movement which have an influence on the person's belief states, and act accordingly. The NRS requires to check both the position and movement of the person at each time step. As mentioned in the observations of the POMDP model, in Chapter 3, two observations are obtained to analyse movements. The observations correspond to the person's position and velocity. Velocity is a vector with two components that require two observations. One for speed (velocity's magnitude) and one for direction. In summary, while the model considers two observations for every person, the implementation actually requires three which are position, speed and movement direction of a person.

A person's position in the environment can be achieved by perceiving images of the environment and then distinguish people in those images. In order to perceive images of the environment, a connection to the camera is established. The camera provides a live video feed of a section of the LRM. By analysing each video frame, the detection of a person can be done using a computer vision approach. The computer vision approach used in the experimental work of this thesis is the Histogram of Oriented Gradients (HOG) with a trained Support Vector Machine (SVM). Both the extraction of each video frame and the consequential people detection are performed using the OpenCV[8] library which contains computer vision techniques with real-time applications.

HOG is a feature descriptor used in computer vision to accomplish people and object detection. A feature descriptor is an algorithm that extracts useful information from an image and ignores irrelevant one. The HOG technique was first presented in [29] and became popular in people detection due to its descriptors outperforming other methods. One of the advantages of HOG and the reason why it is commonly used in people detection is the fact that individual body movements of people can be ignored as long as they maintain a roughly upright position. HOG is similar to edge orientation histograms, scale-invariant feature transform (SIFT) descriptors and shape contexts. The difference between HOG and the other mentioned methods stands on the way HOG descriptors are computed. The computation of descriptors occurs according to the upcoming list of steps:

- Evaluate pixels of the image to perform colour normalization and gamma correction. In an image, the distribution of colour values depends on illumination and shadowing, which may vary. Colour normalization permits to compensate those illumination/shadowing variations. Gamma correction on an image can be used to adjust its brightness. Depending on how dark or bright an image is,

[8]`http://opencv.org`

making tones lighter or darker helps improving people detection. However, in HOG, colour normalization and gamma correction have little impact in the end result due to the subsequent descriptor block normalization achieving similar results [29]. Therefore, this first step is not essential and can be excluded.

- Calculate the horizontal (G_H) and vertical (G_V) gradients of the image. When computing the gradients, the image's colour is filtered using centred $[-1, 0, 1]$ and $[-1, 0, 1]^T$ masks with no Gaussian smoothing applied. Tests conducted in [29] showed that using more complex masks or smoothing worsens results in people detection. At every pixel, the gradient has a magnitude and a direction, which can be obtained from the corresponding horizontal and vertical gradients. The gradient magnitude and orientation are obtained from (4.2) and (4.3), respectively [30].

$$|G| = \sqrt{G_H^2 + G_V^2} \tag{4.2}$$

$$arg(G) = atan\left(\frac{G_H}{G_V}\right) \tag{4.3}$$

- Compute histograms of gradients with spatial and orientation binning. One advantage in using histograms is that this representation is less sensitive to noise. The image is divided into multiple spatial areas called cells, each composed of a certain amount of pixels. These cells can be of rectangular or radial shape. For each cell, an histogram is determined. The gradient magnitude and orientation at each pixel are essential in determining the histogram of each cell. The orientation, in people detection, only assumes values between 0 and 180 degrees, which means that the gradient is unsigned. Using an unsigned gradient implies that the gradient and its negative have the same value. This essentially means that a certain direction and its opposite are seen as the same. Unsigned gradients are used because the wide range of people clothing and background colours make a contrast that disseminates few information. In fact, using unsigned gradients displays a better performance than using signed gradients (values between 0 and 360 degrees). The magnitude in a pixel corresponds to the weighted vote of that pixel's orientation in the cell. The histogram represents the weight of each orientation in a cell. The histogram's dominant direction catches the shape of a person. The values from 0 to 180 degrees are evenly distributed along 9 orientation bins. The reason why 9 bins are used stands on the fact that conducted tests in [29] showed that increasing the number of orientation bins rises performance up to 9 bins, making little difference beyond that.

- Define descriptor blocks and perform block normalization. The computed gradients of the image are conditioned by variations in illumination and contrast. In order to make descriptors independent from these variations, gradients are locally normalized. This normalization consists of grouping cells thus creating a large spatial block. Each block is then normalized separately. Block normalization occurs by concatenating the histograms of the block (since the block is composed by a group of cells and each cell has an associated histogram of orientations) thus forming a non-normalized vector. This vector is then normalized using a block normalization scheme. The HOG descriptor corresponds to the vector of all components of the normalized cell histograms provenient from all blocks. Blocks usually overlap since that allows for some cells to belong to more than one block and therefore contribute to the HOG descriptor more than once. Blocks assume one of two geometries: rectangular and circular. Rectangular blocks are named R-HOG and contain a grid of square or rectangular spatial cells. Circular blocks are called C-HOG and contain log-polar shape cells.

- Define an over detection window. While in [29] a 64x128 detection window was used, as this was the size of the images in the training data, the HOG descriptor can be calculated for multi-scale windows. One way to increase performance and decrease the number of false positives is by including a border with pixels of margin around a person. This way, it is easier to distinguish histograms associated with the shape of a person and histograms associated with the background of the image. The training set used in [29] had a margin of 16 pixels around each person, on all four sides.

- Use a supervised learning classifier. The previous steps allow to extract the HOG descriptor from the image. The final step consists of analysing the information provided by the HOG descriptor using a supervised learning model. In the case of HOG, SVMs are used. In a SVM, a training set of images of two classes are provided, with knowledge on each image being of one type or the other also being given to the model. The images are trained and from that point, the SVM model is capable of designating new data as belonging to one of the two classes. In this case, two training sets were used, one from the MIT CBCL pedestrian database[9] and the other from the INRIA person dataset[10]. From the two databases, a total 2478 images with people and 1218 without were selected. Once the training set has been trained, the SVM classifier gains the ability to distinguish people in images. HOG uses a linear SVM classifier since it has a good combination between performance and run time.

Since detecting people with a high degree of confidence is important to achieve the proposed task, another method was tested. This method, introduced in [31], consists of using Haar feature-based cascade classifiers. This method was created to perform object detection using a boosted cascade of simple features. While Haar cascade detection is most commonly used in the detection of objects and faces, it can also be used to detect people and in fact, the OpenCV library provides a Haar cascade training set for full body detection.

In Haar feature detection, images are classified according to information extracted from features. A detection window is used to select a portion of the image. In locations of the detection window, simple features are positioned, each feature consisting of a rectangle divided in rectangular regions of equal size. When computing features, pixel intensities of each rectangular region are summed up and then it is calculated the difference between the sums of each region. The method uses three types of features: two-rectangle features, three-rectangle and four-rectangle features. The value of two-rectangle features corresponds to the difference between the sums of pixel intensities of the two regions. The value of the three-rectangle features is the difference between the sums on the outside rectangles and the sum of the middle rectangle. Finally, the value of four-rectangular features is obtained from the difference between the sums of diagonal pairs of rectangles. These rectangular features can be computed using an integral image, which assures a reduced computation time. The values of the computed features are used to classify the analysed section of the detection window. In summary, multiple sections of the detection window are analysed using rectangular features and their computed value allows to classify each section. The detection window is moved through the image with Haar features being calculated for every position of the detection window. Once all features are computed, it is possible to distinguish body shapes or objects from the background. This is accomplished by using a threshold, provenient from a training set, to establish if a feature is more likely to belong to one class or not. In people detection in particular, the training set contains information about features that are classified as belonging to a body shape or not.

While the computation of Haar features is fast, a large number of features is necessary, to obtain

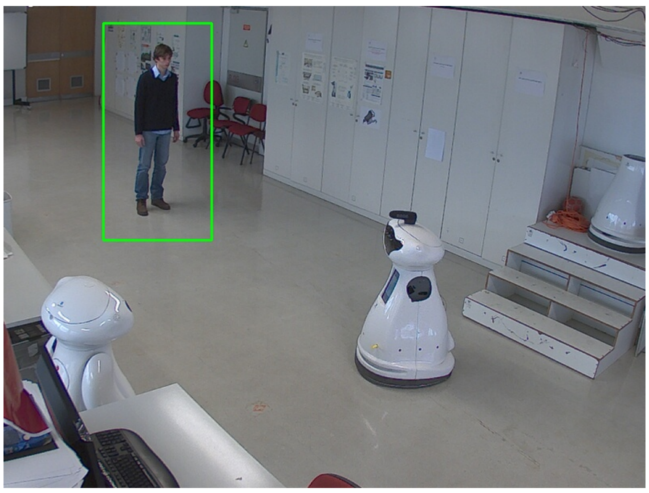
[9]http://cbcl.mit.edu/software-datasets/PedestrianData.html

[10]http://pascal.inrialpes.fr/data/human/

good results in object and people detection, which ends up increasing the overall computation time. The features are organized as a cascade classifier to avoid using a large amount of features and keeping the low computation time advantage of the method. The way the cascade classifier works is by classifying a section of the detection window and if that section returns a bad outcome, that is, if its classified as not being of the intended class (in this case a person) then that section will not be analysed again. If a section provides a good result, that is, if its classified as being of the intended class, then that section is re-evaluated until there is a negative result or until no further processing is required.

Tests were conducted using both methods, HOG with a trained SVM and Haar features. With both methods, the OpenCV library was used and parameters in both of the algorithm's functions were adjusted in order to achieve the best possible results. In both methods, when a person is detected, a bounding box is displayed. This bounding box should contain the entire area where a person is detected.

Tests performed using HOG with a trained SVM displayed good results with a person being correctly detected in over 90% of the frames in a video and few false positives occurring. However, tests conducted with Haar feature-based cascade classifiers performed poorly with only partial detections of a person's body shape and a high number of false positives. A comparison between the two methods can be made by visualizing the outcomes of people detection in Figure 4.8.


(a) HOG with a trained SVM.

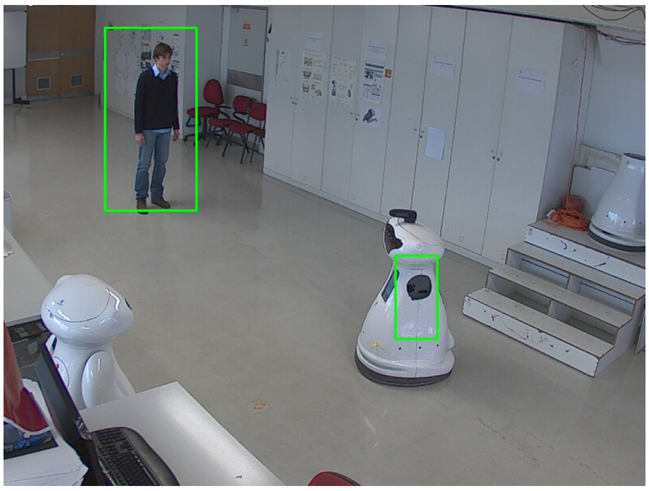

(b) Haar feature-based cascade classifiers.

Figure 4.8: Detection of a person with two different methods. The bounding box represents the area that contains the person.

Since HOG with a trained SVM provides better results, the observations of a person's position in the experimental work conducted were received using this technique.

In Figure 4.8a, a person is seen in an image and a bounding box is inserted. The HOG detector of OpenCV returns a bounding box which is larger than the actual person. The size of the bounding box was shrunken in order to assure that the the bottom line of the box intersects the person's feet. This way, the middle bottom pixel of the generated bounding box is considered to contain the x, y coordinates (the pixels coordinates) of the person in the image.

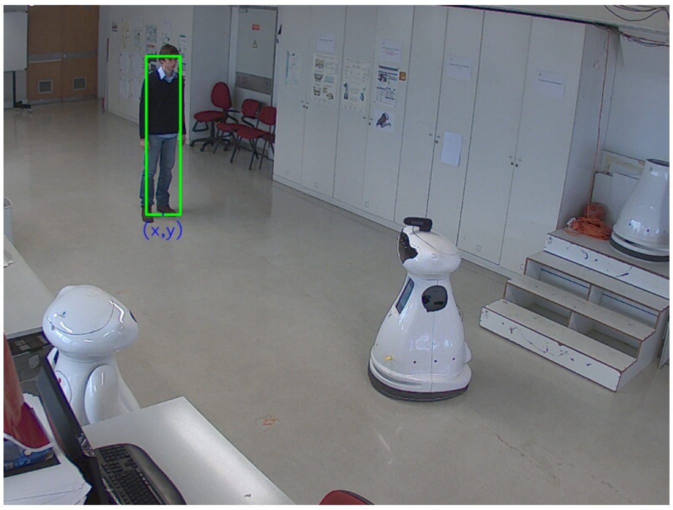


Figure 4.9: Detection of a person using HOG with a trained SVM. The box's bottom middle pixel's Cartesian coordinates indicate the x, y position of a person in the image.

Once the detection of a person in an image is made, it is necessary to find the person's position in the environment's map presented in Figure 4.3. The middle bottom pixel of the generated bounding box needs to be converted into a point on the map. This can be achieved by performing a projective transformation, also known as homography, which maps one plane into another while preserving straight lines [32]. Since the position of the AXIS camera relatively to the environment's map plane is known, the transformation can be applied to convert points in the camera plane, $\tilde{x}$, into the map plane, $\tilde{x}'$. The homography consists of an homogeneous 3×3 matrix $\tilde{H}$. The values for the homography matrix were determined using OpenCV, which allows to find an homography between planes. OpenCV does this computation, given a set of points in one plane and the matching set of points in the other plane. The transformation is determined by,

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{4.4}$$

Since $\tilde{H}$ is homogeneous, it is necessary to normalize $\tilde{x}'$ in order to find the actual x', y' coordinates of a person in the map. From the expression in 4.4, it is possible to determine the normalization factor λ,

$$\lambda = h_{20}x + h_{21}y + h_{22} \tag{4.5}$$

Finally, the inhomogeneous coordinates of a person in the map are given by,

$$x' = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + h_{22}} \tag{4.6}$$

$$y' = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + h_{22}} \tag{4.7}$$

The person's map coordinates are then seen as belonging to one of the twelve regions of the environment and that region corresponds to the person's position observation that is sent to the Symbolic Perseus solver.

Movement direction of a person consists of comparing the person's position at two consecutive moments and finding the correspondent direction vector. This means that the current map position of a person is compared to the previous one. There are four possible directions, these being the four cardinal directions. The directions are relative to the map in Figure 4.3, meaning that, for example, north is upwards. The person's position in the map is represented in x', y' coordinates. By comparing values of x', y' in consecutive moments, it is possible to check if the current coordinates are north, west, east or south to the previous ones. This way the movement direction of a person is determined. If a person is stopped or is not found, then an observation of "Not moving" is obtained. When a person is not walking, that does not necessarily mean it displays no movement. A person may stay in the same place but it is unlikely that they are completely immobile. Therefore, when a person is not walking, the exact pixel where the person is located still has slight variations. This means that when a person is not moving, the middle bottom pixel in the detection bounding box usually slightly oscillates and consequentially so will the person's position in the map. To prevent this small variations to be seen as moving to a direction, a threshold is used. If the absolute difference between x', y' values in consecutive moments is not higher than that threshold, the person is observed as not moving. Note that in movement direction, x', y' values are used and not cell numbers because using cells would mean that a person is stopped unless they transition between cells.

It is important to note that the movement direction is calculated continuously during the person's behaviour and not just between observations received by the solver. The solver receives observations and then returns a policy which is executed by the robot. At this moment, the solver receives once again observations and returns another policy. However, the movement direction of a person cannot be computed by only checking the person's position when the robot finishes its action. This is due to the fact that the robot takes some time to carry out its action and to plan the following one. In that time, a person could be moving in a direction and then turn back and walk to the opposite direction. By only considering positions before and after robot actions, the behaviour of continuous movement of a person would be lost.

A video of the observations of the person's position and movement direction is included in footnote[11].

A person's speed is perceived using the robot's laser reading to detect legs. Legs are detected at positions relative to the laser, and therefore the robot. These positions have x, y values which may be positive or negative depending on the location of the robot's world referential. Once there are positive readings, the distance between leg positions at consecutive readings is determined. The mean value of the leg positions corresponds to the centre of the legs and therefore, that could be seen as the position of the person, relatively to the robot. Since the leg distance is determined over time, it is possible to see how much the person's legs move and therefore the distance covered by the person during that time, that is, the person's velocity. The person's velocity is determined as the velocity in the x coordinate, v_x, and in the y coordinate, v_y. This information is obtained using the ROS package people_velocity_tracker.

Speed can be computed using the expression in 4.8.

$$|\overrightarrow{v}| = \sqrt{v_x^2 + v_y^2} \tag{4.8}$$

[11]https://www.youtube.com/watch?v=lEt500v44ek

Speed is then discretized in three possible classes: $0 - 0.2m/s$, $0.2 - 0.7m/s$ and $> 0.7m/s$. These values were chosen as test conducted showed that when a person walks to a place at a relatively high speed, they usually move at over 0.7 meters per second and at constant speed. The middle class represents a person walking at a medium or low speed and therefore less likely to keep moving in the same direction. Finally, the class with lower speeds considers a person that will likely stop moving or is already standing still.

When a person is not detected in two consecutive frames, it is considered that there is no person and therefore no movement. In that case, the observation "Not moving" is published and the laser readings are ignored, since leg detection could occur due to a false positive (e.g., leg of chair). This is due to tests conducted showing that information obtained by the camera and analysed with HOG is more reliable than the one obtained by the laser scanner. When a person is detected but the laser scanner displays no data, a speed of $0 - 0.2m/s$ is published. In general, the laser scanner displays correct readings often, with no data only if a person is out of range despite the fact that they may be in the environment.

The laser readings provide information of a person's speed. However, they could also determine the position and moving direction. Using the laser instead of the surveillance camera to check a person's position and moving direction could seem a possibility. However, since this information is relative to the robot's referential, the robot's orientation pose would influence both observations. Therefore, the robot's orientation would also have to be included in the POMDP model which would unnecessarily, since the camera is already a valid option, increase its complexion. The camera also provides better results in people detection than the laser scanner which makes it preferable.

Chapter 5

Experiments and Results

With the implementation concluded in Chapter 4, the next step is finding the optimal behaviour of the agent in achieving its goal. This can be done by putting the agent in multiple situations, both in simulation and in real environments, providing observations and checking the actions chosen by the planner. Chapter 5 demonstrates the experiments conducted in order to check if the agent acts according to the task proposed in Chapter 3 and achieves the goals indicated in Chapter 1.

The experiments conducted present two type of scenarios: one and two restricted/dangerous areas. In all experiments, the person's position and movement direction are analysed and the robot acts accordingly in order to accomplish the main goal of keeping people away from a specific place. In each trial, the behaviour of a person is different as that is essential to observe the agent's actions in multiple situations. Each of the experiments presented in this chapter was repeated a few times to ensure that the behaviour of the robot is always the same. The results displayed are representative of the common results in each trial. When examining the agent's selected actions it is important to survey the cumulative reward over time since the agent should maximize it in the long-term. It is also important to monitor the evolution of the belief states in the model. A POMDP uses the values of belief states to consider the degree of certainty of a variable being in a specific state. Actions are only chosen when that degree of certainty is high enough. Essentially, this means that actions are chosen according to the update values of belief states, at each time step, depending on perceived observations. Each state variable contains a matrix with each state and the corresponding probability of that state.

Policies were generated with the use of the tracePOMDPpolicyNonStationary function of Symbolic Perseus. This is due to the fact that, while that function may require a lot of memory, it is also the one in Symbolic Perseus that provides better policies. The optimal action is selected by finding the best α vector computed at each iteration of the solver.

In all of the conducted experiments, an action executed in step n was chosen according to the belief update after the observations perceived in step $n-1$. The first action was selected from the initial conditions of the model. In every experiment, a table and images with results are displayed and the relevant aspects of the evolution of belief states are described. The cumulative reward over time is also displayed. All experiments assumed initial states with complete certainty. The robot's position in fact should always be in a specific node with total certainty. Before starting the trials, the robot should be positioned in the environment, ensuring its self-localization. The position and movement direction of a person could have other initial conditions, including an uniform distribution over all possible states. In uncontrolled scenarios, where it would not be possible to assume initial conditions, assuming them with complete certainty would not be preferable. However, using an uniform distribution is not completely realistic either. It is safe to assume that in every environment, there is always a higher probability of people staying in a given place. In a hospital, for example, people waiting on a queue or being positioned in a waiting area are the most probable scenarios. The initial conditions of the model should always be defined depending on the environment where this task is taking place. Since in the experiments conducted the starting behaviour of the person is known, it was considered that having initial conditions with complete certainty is the best option. From that point forward, the POMDP model is capable of adapting the values of the belief states as it perceives observations.

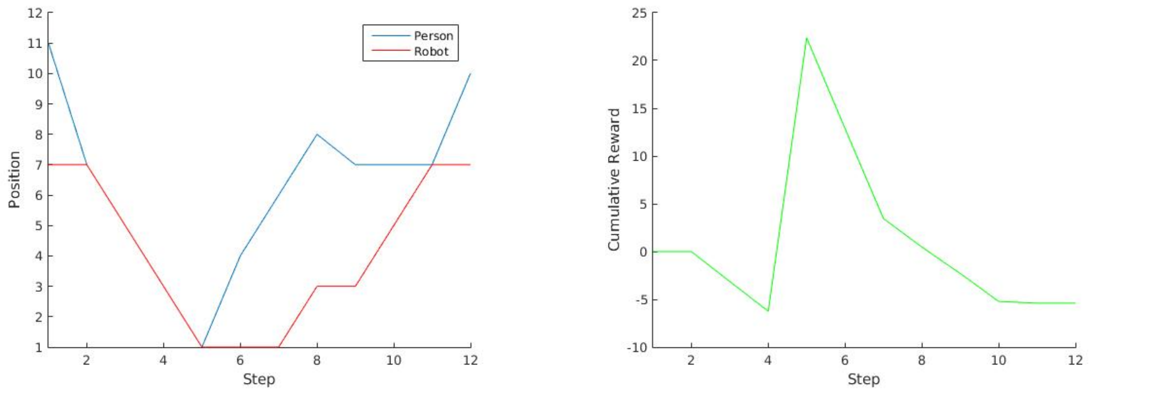
5.1 Experiments with one restricted/dangerous area

The first two experiments consider one restricted/dangerous area located at 1. While a person being in this area is considered to have hazardous behaviour, being in an adjacent cell should also be avoided unless the person is moving away from that spot. A person located at cells 5 and 6 and moving in the direction of 1 (east) is also considered as having avoidable conduct.

5.1.1 Experiment in simulation

The first experiment presented considers a simulation in the MATLAB environment. Instead of observations being provided by sensors and actions being executed by the robot, observations were manually input and the evolution of belief states and the selected actions were studied. A simulation is advantageous as it allows to check if the model behaves as intended in an easily controlled scenario.

The experiment considers a person in the environment, continuously moving in the direction of the restricted/dangerous place. The robot should understand that the person is moving there and should also move towards the same location in order to reach it as soon as possible and inform the person not to be there. The robot should not move to the place until there is certainty that the person is in fact going to stay at that restricted/dangerous area. When the person leaves the place and there is a high degree of certainty that they have in fact left and will not be coming back, the robot will move back to its starting position. Figure 5.1 displays the positions of the person and the robot during the experiment, as well as the cumulative reward over time steps. Table 5.1 contains the agent's behaviour and observations received at each step. Initial conditions in this experiment place a person at cell 11, not moving, and the robot at node 7.



(a) Positions of the person and the robot. (b) Cumulative reward.

Figure 5.1: Simulation with one restricted/dangerous area: positions and cumulative reward over time steps.

In the first step, the belief that the movement direction is east increases to 0.75 once that observation is made. The belief of the individual being at 9 is only of 0.22 while the probability of being at 11 is

0.73. These values are coherent with the model implemented. The position's transition model depends on both the position and direction at the previous step. Since the initial conditions considered that the person is at 11 and not moving, the most probable state at the next step would be to stay at the same position. Since 9 is observed as the current position, the two most likely states for the individual become 11 and 9.

Step	Action	Obs Robot	Obs Person	Obs Velocity
1	Do nothing	7	9	East higher
2	Do nothing	7	7	East higher
3	East	5	5	East higher
4	East	3	3	East 0-0.2
5	East	1	1	Not moving
6	Do nothing	1	4	West 0.2-0.7
7	Do nothing	1	6	West higher
8	West	3	8	South 0-0.2
9	Do nothing	3	7	Not moving
10	West	5	7	Not moving
11	West	7	10	North 0-0.2
12	Do nothing	7	10	Not moving

Table 5.1: Simulation with one restricted/dangerous area: policies and observations over time steps.

In the second step, the probabilities of the person being at 7 is 0.85 and of moving to east increases to 0.89. The fact that the movement direction observed in step one is east increases the possibility of a person having moved from 9 or 11 to an eastern position. The observation of the individual's position contributes to the high belief of being at 7.

The third step shows the robot move to east and its position changing from 7 to 5. This policy is selected since the transition model for the person's position (based on what is described in step two) increased the probability of being in a position east of 7 even before that observation was made. A high probability exists that the individual is moving to east.

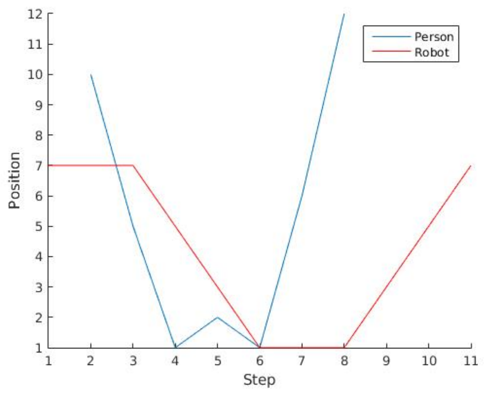
Steps three to five see both the person and the robot moving to 1. The belief of the person being in the observed position is always above 0.9. In order to reach 1, the robot moves and those actions have a penalty. The robot stays outside of its resting position 7 (where it should be) when no person is moving to a restricted/dangerous area. The agent receives negative rewards until its position is 1. At that point, it receives a high positive reward. Actions that did not seem the best option at the time, proved valuable later. This means that the agent acted to maximize its reward in the long-term, in this case a planning horizon of three.

In step six, the person moves away from the restricted/dangerous place, following the robot's verbal alert. The belief that the person is at 4 is 0.61 while the belief of being at 1 is 0.31. The belief of the person's movement direction being west is 0.69 and the probability of moving to north is 0.17. The model considers that there is not a high level of certainty that the person is moving to a far away west position. It is still very probable that the person is at either 1 or 4. Thus, the action chosen in step seven is not to move.

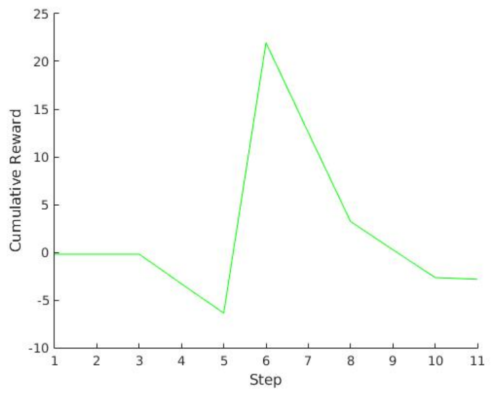
In step seven, the probability of moving to west increases to 0.89 and being in 6 is 0.9. These beliefs have a degree of certainty high enough to ensure that the policy in the next step is move to west. From this point forward, the robot returns to its initial position since there is never another observation of the person moving to east or being in a cell near the restricted/dangerous area.

5.1.2 Experiment with the NRS

The second experiment with one restricted/dangerous area was performed using the actual robot and the surveillance camera. The person's behaviour was similar to the one in the simulation. The initial conditions are however different. The robot starts at 8 and the person is not seen in the environment. Figure 5.2 and Table 5.2 contain the results of this experiment. A video of the experiment is included in footnote[1].



(a) Positions of the person and the robot.



(b) Cumulative reward.

Figure 5.2: Experiment with the NRS and one restricted/dangerous area: positions and cumulative reward over time steps.

Step	Action	Obs Robot	Obs Person	Obs Velocity
1	South	7	No person	Not moving
2	Do nothing	7	10	East 0-0.2
3	Do nothing	7	5	East 0-0.2
4	East	5	1	Not moving
5	East	3	2	Not moving
6	East	1	1	Not moving
7	Do nothing	1	6	East 0-0.2
8	Do nothing	1	12	Not moving
9	West	3	No person	Not moving
10	West	5	No person	Not moving
11	West	7	No person	Not moving

Table 5.2: Experiment with the NRS and one restricted/dangerous area: policies and observations over time steps.

The initial policy makes the robot move to the its predefined place. Since the person is not seen, the robot does not navigate in the next step. In step two, the person is observed moving to east and the belief that the direction is east is 0.49. The person is observed as being at 10 with a certainty of 0.58. In step three, the belief of moving to east increases to 0.8 and being at 5 is 0.72. Since previously the person was at 10 and moving to east, the transition model considers that the most likely position in the next step would be 8. However, since the person is observed at 5, this position ends up being the one with the highest probability, with 8 being the second most likely state with 0.09. Other most probable states, but with smaller rates are 6 and 7.

From steps four to six the robot moves to 1. The probability of the person being at either 1 or 2 is of 0.8 at step four increasing to 0.99 by step six. In step six, the agent receives a high reward, once again

[1]`https://www.youtube.com/watch?v=lgGp0Frtbvc`

maximizing it by choosing previous actions that resulted in negative rewards.

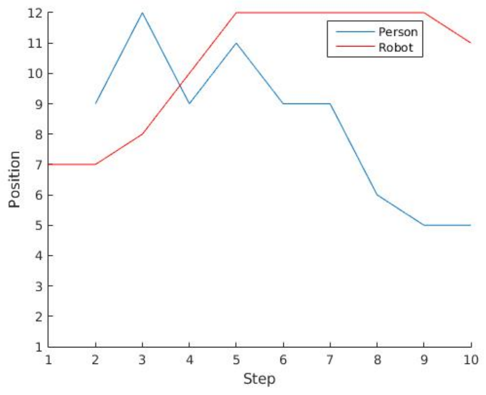
In step seven, the probability of the person moving to east is 0.49 due to the observation made. The model assumes the person could be coming back and, for that reason, the generated policy is of not moving. Only when the person is seen at 12 in step eight, the robot moves to west. The person is not seen again in the environment and the robot returns to 7.

5.2 Experiments with two restricted/dangerous areas

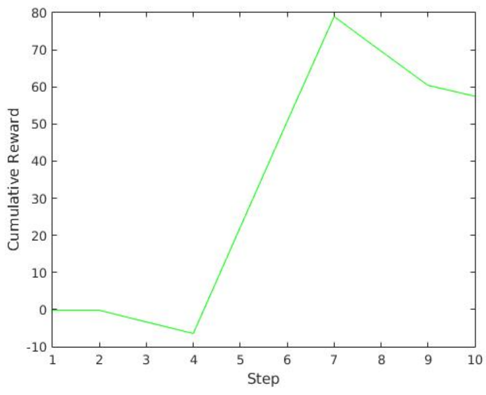
The previous experiments only considered one restricted/dangerous area. In the following two tests, two areas exist. One in 1 and the other in 12. Both experiments were conducted using the robot and the surveillance camera. The reward function provides a high reward when the robot reaches a restricted/dangerous area and an individual is at that area or nearby or moving towards it. The initial conditions in both experiments are identical. The robot starts at 9 and no person is detected in the environment. Videos of both experiments are included in footnote[2][3].

5.2.1 Experiment 1

The experiment, with results in Figure 5.3 and Table 5.3, explores a situation where a person moves to a restricted/dangerous place and does not leave for some time after being told by the robot to avoid that area. The most important aspect in this experiment stands on analysing how the belief state changes once the person finally listens to the robot and leaves. The belief state may take some time to decrease the probability of the person not being in a cell nearby the restricted/dangerous area. This should be avoided as the robot should not stay at that area for a long time after the person's departure. The reason for this is that the person could visit another restricted/dangerous area (and therefore be exposed to a hazardous situation) and the robot would then take a longer time to reach that spot. However, it is also important that the agent does not leave immediately a restricted/dangerous place until there is a high certainty that the person is in fact no longer there. The trial is also significant in analysing how important is to observe the direction to which a person is moving, since most observations showed no movement.



(a) Positions of the person and the robot.



(b) Cumulative reward.

Figure 5.3: Experiment 1 with two restricted/dangerous areas: positions and cumulative reward over time steps.

[2]https://www.youtube.com/watch?v=2S4e5QvYDNM
[3]https://www.youtube.com/watch?v=PzihCS99Mhw

Step	Action	Obs Robot	Obs Person	Obs Velocity
1	East	7	No person	Not moving
2	Do nothing	7	9	South 0-0.2
3	North	8	12	Not moving
4	West	10	9	North 0-0.2
5	West	12	11	Not moving
6	Do nothing	12	9	Not moving
7	Do nothing	12	9	Not moving
8	Do nothing	12	6	Not moving
9	Do nothing	12	5	Not moving
10	South	11	5	Not moving

Table 5.3: Experiment 1 with two restricted/dangerous areas: policies and observations over time steps.

The first step shows the robot move to position 7. This is due to the fact that the 7 is the equidistant place to both restricted/dangerous areas.

In step two, an observation of a person being at 9 is made. Owing to the initial conditions and the observations of the person's variables in step one, the belief that the person is at 9 is only of 0.57 and the movement direction to south has a degree of certainty of 0.47. Not moving 0.28. West and north have a summed probability of 0.15. This degree of certainty resulting from of the combination of a person being at 9 and either not moving or moving closer to 12 (north or west) is enough to make the robot start its movement towards 12 in step three.

Between steps three and five the person is always observed at one of the cells 9 to 12, that is, the restricted/dangerous cell or one of the adjacent ones. The belief state during this interval indicates that the probability of being in one of those four cells is always above 0.9. The robot reaches the restricted/dangerous place at step five and receives a high reward. Since the person does not leave the area despite the robot's verbal request, the agent continues to receive high rewards until step seven. By step seven, the probability of a person being at 9 is 0.99 and the probability of not moving is 0.96.

In step eight, an observation of the person being at 6 is received. However, the degree of certainty is only of 0.57 with a 0.3 probability that the person is still at 9. Only in the next step, when the observation received is 5, the probability of the person's position being 9 drops severely to the value of 0.03. At this moment, the robot moves to south and would then presumably return to 7.

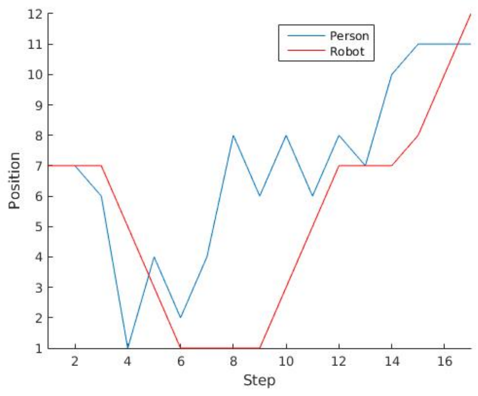
5.2.2 Experiment 2

The final experiment tests the case where a person visits both restricted/dangerous places. Figure 5.4 and Table 5.4 contain the results of the experiment.

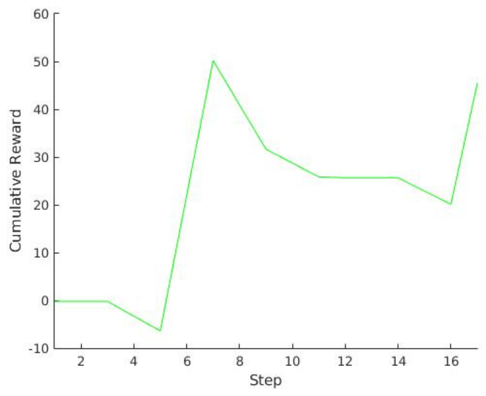
During the first three steps, the robot is at 7, as this is the equidistant spot where the robot receives no penalties. At step four, the robot moves to east since the person is observed at 6 with a probability of being in that state of 0.88 and the person's movement direction has a 0.61 probability of being east.

While the robot travels to 1, the probability of the person being in a cell between 1 and 4 is always above 0.85. By step six, the agent receives a high reward which compensates the penalties received in previous steps. At step nine, an observation of moving to west was perceived. That, combined with the person's positions in previous observations, makes the robot move to west. While the robot navigates, the person never abandons the middle region of the map space. For that reason, probabilities of being in cells in the left or right sides of the map are constantly low.

Finally, at step fourteen, the person is observed in 10. Since in step thirteen the person was observed at 7, the model considers that the person must have moved to west between steps, besides also moving to north. For this reason, while north has a probability of 0.65, moving to west has a probability of 0.15. The belief state of both the person's position and moving direction causes the robot to start moving to 12. Between steps fifteen and seventeen the individual is always observed at 11, a position that guarantees a reward to the agent once it reaches 12 and therefore it continues its movements to 12 receiving another high reward at step seventeen.



(a) Positions of the person and the robot.



(b) Cumulative reward.

Figure 5.4: Experiment 2 with two restricted/dangerous areas: positions and cumulative reward over time steps.

Step	Action	Obs Robot	Obs Person	Obs Velocity
1	East	7	No person	Not moving
2	Do nothing	7	7	North higher
3	Do nothing	7	6	East 0-0.2
4	East	5	1	Not moving
5	East	3	4	East 0-0.2
6	East	1	2	Not moving
7	Do nothing	1	4	Not moving
8	Do nothing	1	8	Not moving
9	Do nothing	1	6	West 0-0.2
10	West	3	8	Not moving
11	West	5	6	Not moving
12	West	7	8	South 0.2-0.7
13	Do nothing	7	7	Not moving
14	Do nothing	7	10	North 0.2-0.7
15	North	8	11	Not moving
16	West	10	11	East 0-0.2
17	West	12	11	Not moving

Table 5.4: Experiment 2 with two restricted/dangerous areas: policies and observations over time steps.

5.3 Analysis

Results evidenced that the model can be used to accomplish the proposed task. The usage of a POMDP in this task is an advantage since the policies are chosen according to the belief states whose values evolve over time. Policies are only selected when there is a high degree of confidence of the person's position and movement direction. The reward models also prove to be useful as the robot always took the shortest path between areas of interest and executed actions that resulted in penalties but proved better in the long run.

When the robot is at the restricted/dangerous place, it will only leave it once the person is seen outside of it for two consecutive times. Results show that two belief updates are necessary to increase the degree of certainty that the person is no longer at the spot. This can be considered as a positive outcome since it means that two potentially negative situations where the robot would be leaving too soon can be discarded: a false positive placing the person in another position; and the person leaving and immediately coming back. The fact that it takes two observations for the robot to leave is something that would require the usage of memory, should a heuristic be used instead of a POMDP.

In a simulation an assumption that the action is carried out instantly can be made which is the most advisable measure to test the model's behaviour. Gathering observations of the world, providing these observations to the planning system and sending the resulting policy to the acting system can be considered duties that take barely no time. For this reason, in the initial steps, a person is observed continuously moving in the same direction and transitioning between cells at the same pace as the robot. The robot only initiates movement once there is a high probability of the person moving to east. The robot and the person reach the restricted/dangerous area at the same time step. Thus, the person is advised not to be there immediately at their arrival. The results in simulations demonstrate how the task should be performed by the robot in the real world. However, when comparing results between experiments in 5.1, the limitations of using an actual robot are clear. In both experiments the person exhibits the same type of behaviour. However, in the experiment using the robot, the person is already at the dangerous zone when the robot starts navigating. Experiments with the robot showed that the person reaches the restricted/dangerous place seconds before the robot. This means that while the model can be used for this task, there will always be a delay between the time person and robot reach an area of interest. Despite the delay, the robot will still act according to the observations perceived and will request for the person to leave the area which was the goal in the execution of the task.

Experiments with two restricted/dangerous zones showed that the robot will not move to one of the zones unless the person does so. While the person is between both areas, the robot will stay in the equidistant spot. Another important conclusion derived from the experiments is that the transition model has a significant influence in the belief states. The direction a person is moving influences the position of the person in the next step. If the observation received is not consistent with the previous moving direction, the belief that the person is in the state observed will not be that high. For this reason, it is important to guarantee that a very short time passes during the robot's navigation as it is important that the person's movement is observed continuously over time instead of at separated moments.

Chapter 6

Conclusions and Future Work

The work carried out is considered positive. It was possible to perform optimal planning and apply it to interact with a person in an indoor space using a NRS. The evolution of technology and robotics will likely increase the usage of NRSs in common daily tasks. For that reason, planning is an important area of study as it may be used to improve HRI in both the present and specially in the future. The model developed proved that the POMDP framework is useful in task planning. POMDPs deal with uncertainty in a way that most frameworks do not. It also uses belief states to understand how the system evolves over time instead of only considering present observations.

The model was considered for the specific environment where tests occurred. The values chosen in the transition and observation models as well as in the reward function would have to be adjusted to other locations. The number of possible states could also be different. The fact that the model could be successfully implemented in the test environment shows that it could have the same result in other indoor spaces. The main objective in studying planning and applying it in real world conditions consists in understanding the extend of its utility and the limitations that still exist in NRSs. From that, it is concluded that while this task may be applied in the conditions of the experimental work of this thesis, it will be more suitable for larger open spaces. In open spaces it is easier to observe the person's movement over time. Also, robots are able to navigate and perform global and local path planning in low time.

The conclusive results of this thesis are positive but there is still plenty of work that needs to be done. NRSs still have various limitations. Observations are usually perceived in discontinuous fashion and robots usually act much slower than humans, particularly in navigation tasks. In the implemented task, the number of frames per second captured by the surveillance camera and analysed by the people detection method was always below twenty, with the actual number oscillating depending on the number of processes being executed by the computer used.

Navigation between waypoints in cells took some time due to both the robot's physical limitations and the existing navigation system. When the robot moves, it needs to update a costmap of the environment. This costmap contains obstacles that the robot should avoid. In the experiments conducted, when the person walked near the robot, the costmap indicated an obstacle. This prevented immediate navigation when a goal was received. It took some time for the costmap to be updated and that obstacle to disappear from the robot's path. There was also the need to create a global and a local path between waypoints. While creating a path takes little time, it is still another cause for the delay between each of the robot's moving actions. These limitations prevent a more realistic HRI.

Instead of using fixed waypoints inside each cell, another possibility could have been the robot travelling a certain fixed distance, without the need of waypoints, in the direction provided by the policy. Navigating without waypoints and without path planning may be troublesome though as it would not take into account possible obstacles. The model could also be changed to consider a variable indicating the existence of obstacles. Their existence could be perceived by sensors and their presence influence the robot's transition model as actions of moving in a direction would only be taking place if no obstacle existed in the robot's path. This would however increase the complexity of the model. This addition to the model could be implemented in the future.

The POMDP model took into account one person in the environment. This could be useful in monitoring one particular person, such as a child's behaviour in a home. But in a large open space, where the robot would need to keep a high number of people away from multiple places, the model would have to consider extensive state and observation spaces. This would increase the computation time of the model exponentially. For this reason, while the model can be designed with a higher number of state and observation variables, it is limited to the current computation power. The best solutions consist in finding ways of changing the model to decrease its dimension. One possible solution could be having multiple robots in the environment. Each robot would be responsible for specific regions or particular people. This would mean that the model could be divided into multiple smaller models, each computed separately, thus decreasing computation time. Another solution could be decreasing the number of possible positions for a person. Positions in the environment would have a low number of possible states: in, near or far from a restricted/dangerous area existing in the model.

The movement direction implemented in the model took advantage of the fact that a person would necessarily move to east or west to reach a dangerous place. In a larger environment, that information would not be enough. It would be necessary to use further directions, possibly using orientations with an associated angle.

The fact that POMDPs have a fixed number of variables and that their models cannot change is not always recommendable. The task in the thesis can be accomplished when the environment is known. However, situations such as a higher number of people than the one considered in the model or the level of uncertainty in sensing changing due to an unpredictable circumstance may occur. One elegant solution of adapting to unplanned occurrences is online planning which allows to change the number of variables and existing models according to each situation. Applying this model in unknown environments with other methods of planning under uncertainty could be valuable.

The environment used had several limitations. The open area was not very large and it was not possible to use the entire lab environment since some areas were not completely observable by the surveillance cameras. It would be interesting to test the model in a different environment, with a larger space, to compare results with the experiments in this thesis.

Due to the fact that the model considers only one person, performing person detection was enough to determine the person's position. However, in cases where multiple people are considered, to analyse a person's behaviour over time, a tracking system would need to be implemented. This would be the only way of associating each person in the environment with their correspondent person state variables. Mixing other components of computer vision, such as people tracking or recognition, with planning under uncertainty is a path that would be interesting to follow.

The graphical tool of OpenMarkov is quite useful as it enables a faster and more understandable design of POMDPs. However, file types generated using this tool are not compatible with all existing solvers and software. In fact, POMDPs can be designed in multiple formats but only some are compatible with certain solvers. It is important to assure a higher compatibility and easier and faster integrations between ROS and POMDP solvers to increase the usage of this framework in planning.

Machines processing power have been increasing over time and it is safe to assume that current complex POMDP models will be solvable in a shorter period of time in the future. It is also pertinent to assume that NRSs will take less time executing actions and sensing their surroundings in the future. For that reason, there is a lot of potential of this task being applied in multiple scenarios and of POMDPs becoming of regular use in human-aware planning.

Bibliography

[1] A. Sanfeliu, N. Hagita, and A. Saffiotti, “Network robot systems,” *Robotics and Autonomous Systems*, vol. 56, no. 2, pp. 793–797, 2008.

[2] R. Alterovitz, S. Koenig, and M. Likhachev, “Robot Planning in the Real World: Research Challenges and Opportunities,” *Artificial Intelligence Magazine*, vol. 37, pp. 76–84, 2016.

[3] I. Leite, C. Martinho, and A. Paiva, “Social Robots for Long-Term Interaction: A Survey,” *International Journal of Social Robotics*, vol. 5, no. 2, pp. 291–308, 2013.

[4] K. M. Tsui, H. A. Yanco, D. J. Feil-Seifer, and M. J. Matarić, “Survey of domain-specific performance measures in assistive robotic technology,” *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems - PerMIS ’08*, 2008.

[5] K. Wada, “Social and physiological influences of living with seal robots in an elderly care house for two months,” *Gerontechnology*, no. June 2005, pp. 235–235, 2008.

[6] J. Y. Sung, R. E. Grinter, and H. I. Christensen, “Domestic robot ecology: An initial framework to unpack long-term acceptance of robots at home,” *International Journal of Social Robotics*, vol. 2, no. 4, pp. 417–429, 2010.

[7] J. Hoey, P. Poupart, C. Boutilier, and A. Mihailidis, “POMDP models for assistive technology,” *Proc AAAI Fall Symposium on Assistive Technologies*, 2014.

[8] J. Boger, P. Poupart, J. Hoey, C. Boutilier, G. Fernie, and A. Mihailidis, “A decision-theoretic approach to task assistance for persons with dementia,” *IJCAI International Joint Conference on Artificial Intelligence*, pp. 1293–1299, 2005.

[9] M. T. J. Spaan, T. S. Veiga, and P. U. Lima, “Active cooperative perception in network robot systems using POMDPs,” *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, pp. 4800–4805, 2010.

[10] J. Pineau, M. Montemerlo, M. Pollack, N. Roy, and S. Thrun, “Towards robotic assistants in nursing homes: Challenges and results,” *Robotics and Autonomous Systems*, vol. 42, no. 3-4, pp. 271–281, 2003.

[11] H. Kurniawati, Y. Du, D. Hsu, and W. S. Lee, “Motion planning under uncertainty for robotic tasks with long time horizons,” *Springer Tracts in Advanced Robotics*, vol. 70, pp. 151–168, 2011.

[12] S. Seuken and S. Zilberstein, “Memory-Bounded Dynamic Programming for DEC-POMDPs,” *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2009–2015, 2007.

[13] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, “A survey of monte carlo tree search methods,” *IEEE Transactions on Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–49, 2012.

[14] D. Silver and J. Veness, “Monte-Carlo Planning in Large POMDPs,” *Advances in neural information processing systems (NIPS).*, pp. 1–9, 2010.

[15] D. Wierstra, A. Foerster, J. Peters, and J. Schmidthuber, “Solving Deep Memory POMDPs with Recurrent Policy Gradients,” *Artificial Neural Networks – ICANN*, vol. 1, no. 1, pp. 697–706, 2007.

[16] S. Russell and P. Norvig, “Artificial Intelligence A Modern Approach (Third Edition),” pp. 645–666, 2013.

[17] L. Pack, M. L. Littman, and A. R. Cassandra, “Planning and acting in partially observable stochastic domains,” *Artificial Intelligence*, vol. 101, pp. 99–134, 1998.

[18] M. T. J. Spaan, “Partially Observable Markov Decision Processes,” *Reinforcement Learning - State-of-the-Art*, pp. 387–414, 2012.

[19] M. T. Izadi and D. Precup, “Using rewards for belief state updates in partially observable markov decision processes,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3720 LNAI, pp. 593–600, 2005.

[20] P. Lison, “Towards Relational POMDPs for Adaptive Dialogue Management,” *Proceedings of the ACL 2010 Student Research Workshop*, no. July, pp. 7–12, 2010.

[21] K. Hsiao, L. P. Kaelbling, and T. Lozano-Pérez, “Grasping POMDPs,” *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 4685–4692, 2007.

[22] R. D. Smallwood and E. J. Sondik, “The Optimal Control of Partially Observable Markov Decision Processes over a Finite Horizon,” *Operations Research*, vol. 21, no. 5, pp. 1071 – 1088, 1973.

[23] G. Hollinger, “Partially Observable Markov Decision Processes (POMDPs) Outline for POMDP Lecture,” *Artificial Intelligence*, 2007.

[24] M. T. J. Spaan and N. Vlassis, “A point-based POMDP algorithm for robot planning,” *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2399–2404, 2004.

[25] J. Pineau, G. Gordon, and S. Thrun, “Point-based value iteration: An anytime algorithm for POMDPs,” *IJCAI International Joint Conference on Artificial Intelligence*, pp. 1025–1030, 2003.

[26] R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, a. Pardo, and F. Somenzi, “Algebraic decision diagrams and their applications,” *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*, vol. 206, pp. 171–206, 1993.

[27] P. Poupart, “Exploiting Structure to Efficiently Solve Large Scale POMDPs,” *Ph.D. dissertation, University of Toronto*, 2005.

[28] M. Arias, F. Diez, M. A. Palacios-Alonso, M. Yebra, and J. Fernandez, “POMDPs in OpenMarkov and ProbModelXML,” *The Seventh Annual Workshop on Multiagent Sequential Decision-Making Under Uncertainty (MSDM-2012)*, no. June, pp. 1–8, 2012.

[29] N. Dalal and B. Triggs, “Histograms of Oriented Gradients for Human Detection,” *Coference on Computer Vision and Pattern Recognition (CVPR)*, 2005.

[30] M. Bertozzi, A. Broggi, M. D. Rose, M. Felisa, A. Rakotomamonjy, and F. Suard, “A Pedestrian Detector Using Histograms of Oriented Gradients and a Support Vector Machine Classifier,” *Proceedings of the 2007 IEEE Intelligent Transportation Systems Conference*, pp. 143–148, 2007.

[31] P. Viola and M. Jones, “Rapid Object Detection using a Boosted Cascade of Simple Features,” *Coference on Computer Vision and Pattern Recognition (CVPR)*, 2001.

[32] R. Szeliski, “Computer Vision: Algorithms and Applications,” *Springer, Online Book*, 2010.

Appendix A

Task POMDP Model

The purpose of this appendix is to display the values used in the construction of the POMDP model. The Figures in this appendix show the reward function and the transition and observation models represented as ADDs as they are displayed in OpenMarkov. Note that [0] and [1] represent the current and the following time steps respectively.

P(Direction [1]) = {Not moving, South, East, West, North}

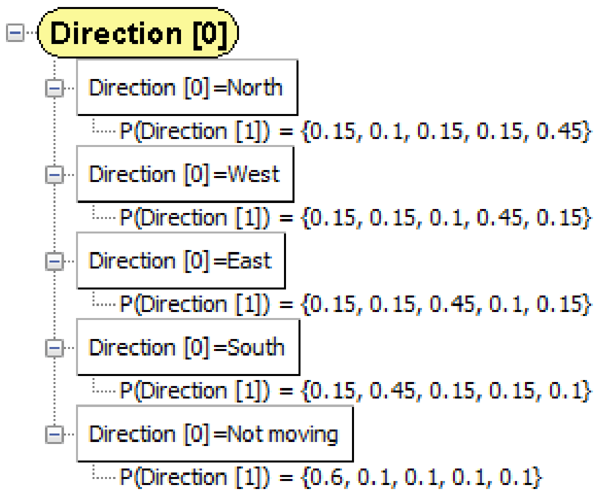
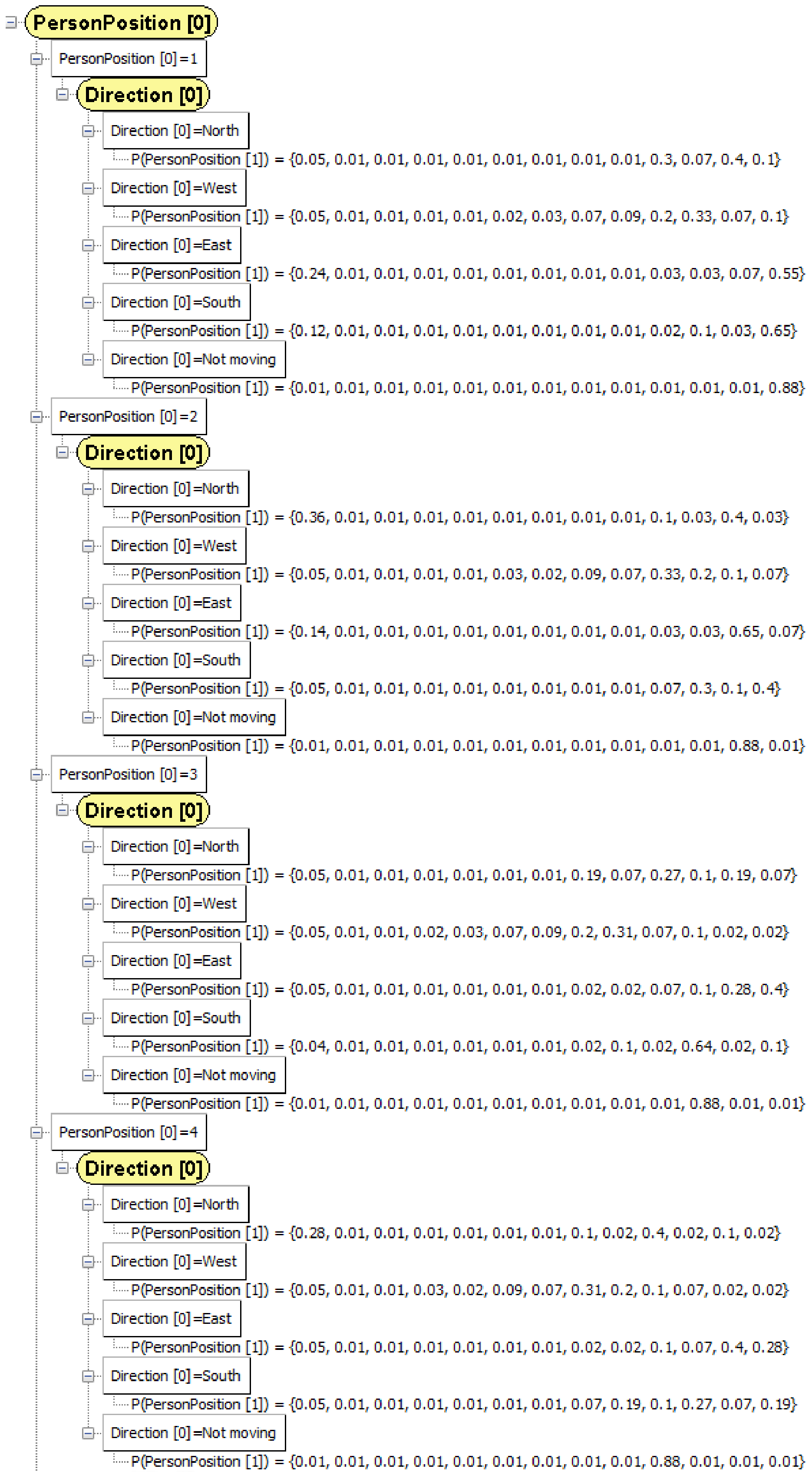

```
Direction [0]
  Direction [0]=North
    P(Direction [1]) = {0.15, 0.1, 0.15, 0.15, 0.45}
  Direction [0]=West
    P(Direction [1]) = {0.15, 0.15, 0.1, 0.45, 0.15}
  Direction [0]=East
    P(Direction [1]) = {0.15, 0.15, 0.45, 0.1, 0.15}
  Direction [0]=South
    P(Direction [1]) = {0.15, 0.45, 0.15, 0.15, 0.1}
  Direction [0]=Not moving
    P(Direction [1]) = {0.6, 0.1, 0.1, 0.1, 0.1}
```

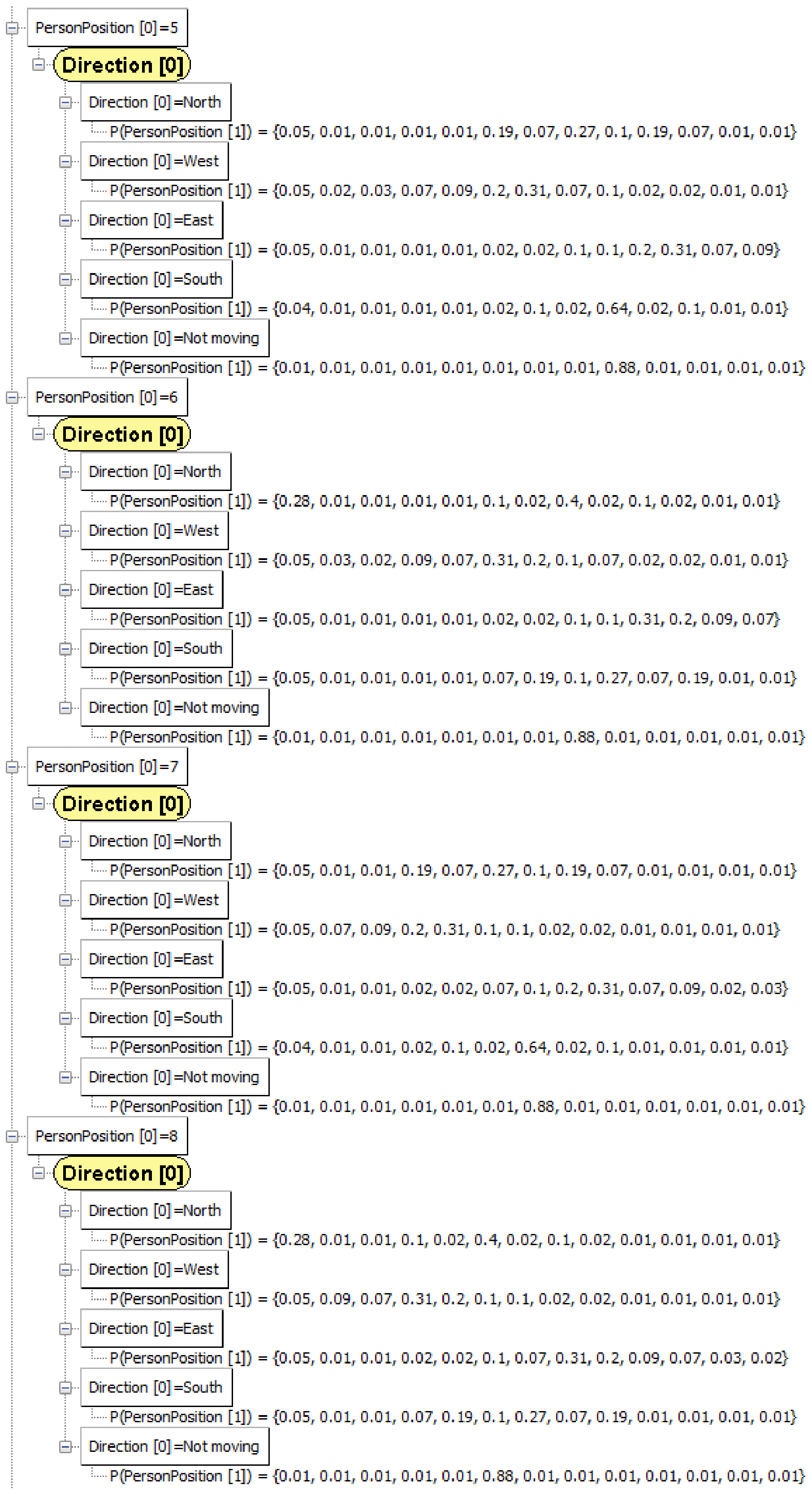
Figure A.1: Transition model of the person's movement direction.

P(PersonPosition [1]) = {No person, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1}



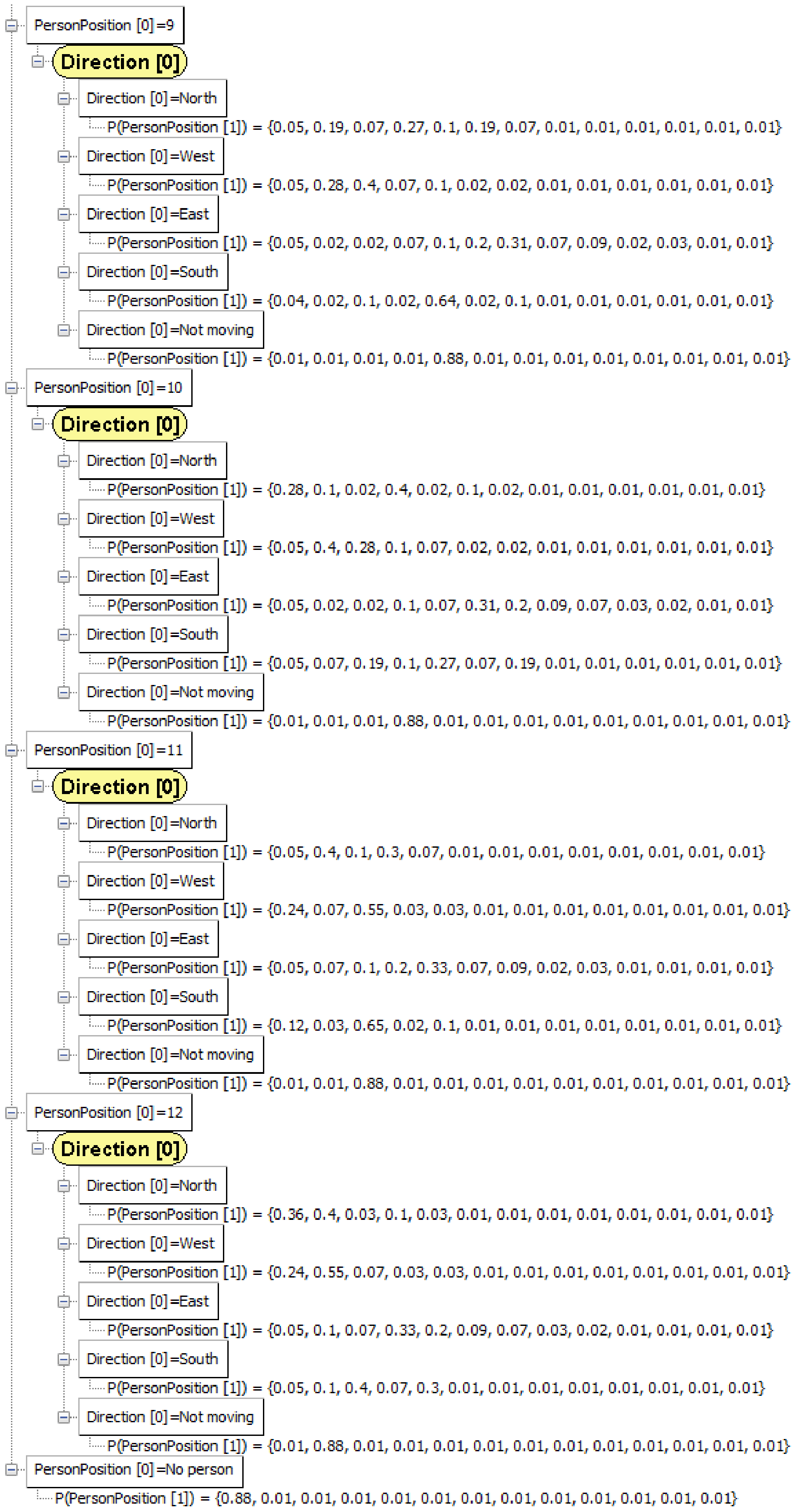
```
PersonPosition [0]
  PersonPosition [0]=1
    Direction [0]
      Direction [0]=North
        P(PersonPosition [1]) = {0.05, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.3, 0.07, 0.4, 0.1}
      Direction [0]=West
        P(PersonPosition [1]) = {0.05, 0.01, 0.01, 0.01, 0.01, 0.02, 0.03, 0.07, 0.09, 0.2, 0.33, 0.07, 0.1}
      Direction [0]=East
        P(PersonPosition [1]) = {0.24, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.03, 0.03, 0.07, 0.55}
      Direction [0]=South
        P(PersonPosition [1]) = {0.12, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.02, 0.1, 0.03, 0.65}
      Direction [0]=Not moving
        P(PersonPosition [1]) = {0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.88}
  PersonPosition [0]=2
    Direction [0]
      Direction [0]=North
        P(PersonPosition [1]) = {0.36, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.1, 0.03, 0.4, 0.03}
      Direction [0]=West
        P(PersonPosition [1]) = {0.05, 0.01, 0.01, 0.01, 0.01, 0.03, 0.02, 0.09, 0.07, 0.33, 0.2, 0.1, 0.07}
      Direction [0]=East
        P(PersonPosition [1]) = {0.14, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.03, 0.03, 0.65, 0.07}
      Direction [0]=South
        P(PersonPosition [1]) = {0.05, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.07, 0.3, 0.1, 0.4}
      Direction [0]=Not moving
        P(PersonPosition [1]) = {0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.88, 0.01}
  PersonPosition [0]=3
    Direction [0]
      Direction [0]=North
        P(PersonPosition [1]) = {0.05, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.19, 0.07, 0.27, 0.1, 0.19, 0.07}
      Direction [0]=West
        P(PersonPosition [1]) = {0.05, 0.01, 0.01, 0.02, 0.03, 0.07, 0.09, 0.2, 0.31, 0.07, 0.1, 0.02, 0.02}
      Direction [0]=East
        P(PersonPosition [1]) = {0.05, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.02, 0.02, 0.07, 0.1, 0.28, 0.4}
      Direction [0]=South
        P(PersonPosition [1]) = {0.04, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.02, 0.1, 0.02, 0.64, 0.02, 0.1}
      Direction [0]=Not moving
        P(PersonPosition [1]) = {0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.88, 0.01, 0.01}
  PersonPosition [0]=4
    Direction [0]
      Direction [0]=North
        P(PersonPosition [1]) = {0.28, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.1, 0.02, 0.4, 0.02, 0.1, 0.02}
      Direction [0]=West
        P(PersonPosition [1]) = {0.05, 0.01, 0.01, 0.03, 0.02, 0.09, 0.07, 0.31, 0.2, 0.1, 0.07, 0.02, 0.02}
      Direction [0]=East
        P(PersonPosition [1]) = {0.05, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.02, 0.02, 0.1, 0.07, 0.4, 0.28}
      Direction [0]=South
        P(PersonPosition [1]) = {0.05, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.07, 0.19, 0.1, 0.27, 0.07, 0.19}
      Direction [0]=Not moving
        P(PersonPosition [1]) = {0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.88, 0.01, 0.01, 0.01}
```

Figure A.2: Transition model of the person's position (1 of 3).



```
PersonPosition [0]=5
  Direction [0]
    Direction [0]=North
      P(PersonPosition [1]) = {0.05, 0.01, 0.01, 0.01, 0.01, 0.19, 0.07, 0.27, 0.1, 0.19, 0.07, 0.01, 0.01}
    Direction [0]=West
      P(PersonPosition [1]) = {0.05, 0.02, 0.03, 0.07, 0.09, 0.2, 0.31, 0.07, 0.1, 0.02, 0.02, 0.01, 0.01}
    Direction [0]=East
      P(PersonPosition [1]) = {0.05, 0.01, 0.01, 0.01, 0.01, 0.02, 0.02, 0.1, 0.1, 0.2, 0.31, 0.07, 0.09}
    Direction [0]=South
      P(PersonPosition [1]) = {0.04, 0.01, 0.01, 0.01, 0.01, 0.02, 0.1, 0.02, 0.64, 0.02, 0.1, 0.01, 0.01}
    Direction [0]=Not moving
      P(PersonPosition [1]) = {0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.88, 0.01, 0.01, 0.01, 0.01}
PersonPosition [0]=6
  Direction [0]
    Direction [0]=North
      P(PersonPosition [1]) = {0.28, 0.01, 0.01, 0.01, 0.01, 0.1, 0.02, 0.4, 0.02, 0.1, 0.02, 0.01, 0.01}
    Direction [0]=West
      P(PersonPosition [1]) = {0.05, 0.03, 0.02, 0.09, 0.07, 0.31, 0.2, 0.1, 0.07, 0.02, 0.02, 0.01, 0.01}
    Direction [0]=East
      P(PersonPosition [1]) = {0.05, 0.01, 0.01, 0.01, 0.01, 0.02, 0.02, 0.1, 0.1, 0.31, 0.2, 0.09, 0.07}
    Direction [0]=South
      P(PersonPosition [1]) = {0.05, 0.01, 0.01, 0.01, 0.01, 0.07, 0.19, 0.1, 0.27, 0.07, 0.19, 0.01, 0.01}
    Direction [0]=Not moving
      P(PersonPosition [1]) = {0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.88, 0.01, 0.01, 0.01, 0.01, 0.01}
PersonPosition [0]=7
  Direction [0]
    Direction [0]=North
      P(PersonPosition [1]) = {0.05, 0.01, 0.01, 0.19, 0.07, 0.27, 0.1, 0.19, 0.07, 0.01, 0.01, 0.01, 0.01}
    Direction [0]=West
      P(PersonPosition [1]) = {0.05, 0.07, 0.09, 0.2, 0.31, 0.1, 0.1, 0.02, 0.02, 0.01, 0.01, 0.01, 0.01}
    Direction [0]=East
      P(PersonPosition [1]) = {0.05, 0.01, 0.01, 0.02, 0.02, 0.07, 0.1, 0.2, 0.31, 0.07, 0.09, 0.02, 0.03}
    Direction [0]=South
      P(PersonPosition [1]) = {0.04, 0.01, 0.01, 0.02, 0.1, 0.02, 0.64, 0.02, 0.1, 0.01, 0.01, 0.01, 0.01}
    Direction [0]=Not moving
      P(PersonPosition [1]) = {0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.88, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01}
PersonPosition [0]=8
  Direction [0]
    Direction [0]=North
      P(PersonPosition [1]) = {0.28, 0.01, 0.01, 0.1, 0.02, 0.4, 0.02, 0.1, 0.02, 0.01, 0.01, 0.01, 0.01}
    Direction [0]=West
      P(PersonPosition [1]) = {0.05, 0.09, 0.07, 0.31, 0.2, 0.1, 0.1, 0.02, 0.02, 0.01, 0.01, 0.01, 0.01}
    Direction [0]=East
      P(PersonPosition [1]) = {0.05, 0.01, 0.01, 0.02, 0.02, 0.1, 0.07, 0.31, 0.2, 0.09, 0.07, 0.03, 0.02}
    Direction [0]=South
      P(PersonPosition [1]) = {0.05, 0.01, 0.01, 0.07, 0.19, 0.1, 0.27, 0.07, 0.19, 0.01, 0.01, 0.01, 0.01}
    Direction [0]=Not moving
      P(PersonPosition [1]) = {0.01, 0.01, 0.01, 0.01, 0.01, 0.88, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01}
```

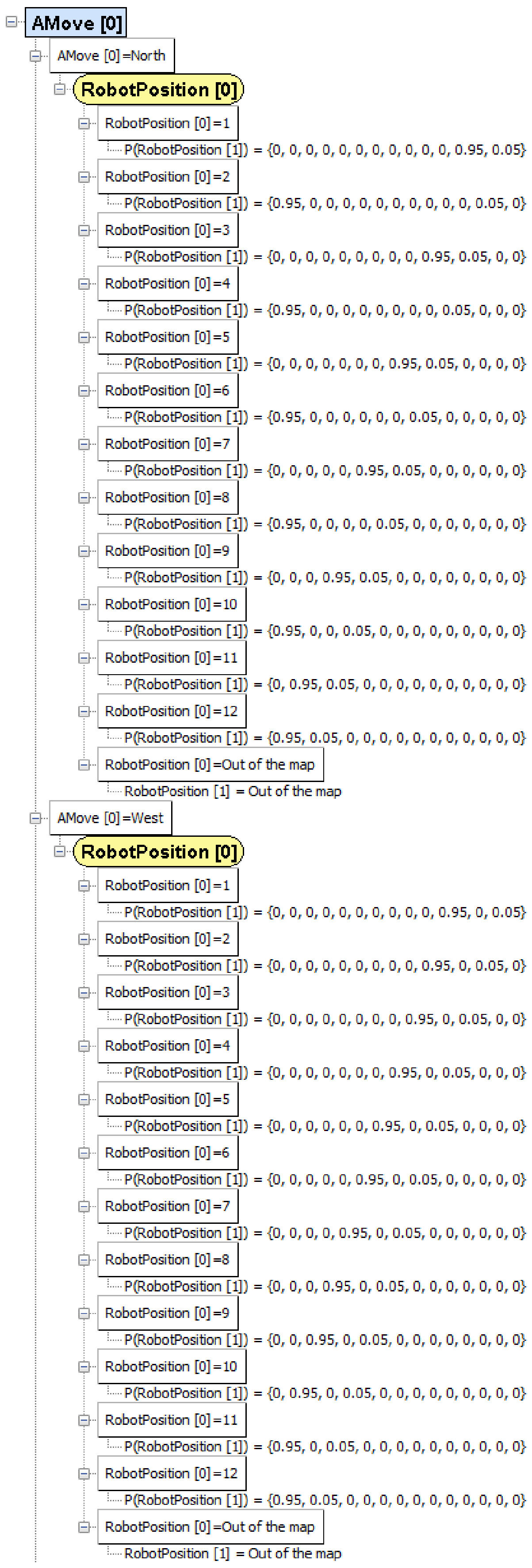
Figure A.3: Transition model of the person's position (2 of 3).



```
PersonPosition [0]=9
  Direction [0]
    Direction [0]=North
      P(PersonPosition [1]) = {0.05, 0.19, 0.07, 0.27, 0.1, 0.19, 0.07, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01}
    Direction [0]=West
      P(PersonPosition [1]) = {0.05, 0.28, 0.4, 0.07, 0.1, 0.02, 0.02, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01}
    Direction [0]=East
      P(PersonPosition [1]) = {0.05, 0.02, 0.02, 0.07, 0.1, 0.2, 0.31, 0.07, 0.09, 0.02, 0.03, 0.01, 0.01}
    Direction [0]=South
      P(PersonPosition [1]) = {0.04, 0.02, 0.1, 0.02, 0.64, 0.02, 0.1, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01}
    Direction [0]=Not moving
      P(PersonPosition [1]) = {0.01, 0.01, 0.01, 0.01, 0.88, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01}
PersonPosition [0]=10
  Direction [0]
    Direction [0]=North
      P(PersonPosition [1]) = {0.28, 0.1, 0.02, 0.4, 0.02, 0.1, 0.02, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01}
    Direction [0]=West
      P(PersonPosition [1]) = {0.05, 0.4, 0.28, 0.1, 0.07, 0.02, 0.02, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01}
    Direction [0]=East
      P(PersonPosition [1]) = {0.05, 0.02, 0.02, 0.1, 0.07, 0.31, 0.2, 0.09, 0.07, 0.03, 0.02, 0.01, 0.01}
    Direction [0]=South
      P(PersonPosition [1]) = {0.05, 0.07, 0.19, 0.1, 0.27, 0.07, 0.19, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01}
    Direction [0]=Not moving
      P(PersonPosition [1]) = {0.01, 0.01, 0.01, 0.88, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01}
PersonPosition [0]=11
  Direction [0]
    Direction [0]=North
      P(PersonPosition [1]) = {0.05, 0.4, 0.1, 0.3, 0.07, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01}
    Direction [0]=West
      P(PersonPosition [1]) = {0.24, 0.07, 0.55, 0.03, 0.03, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01}
    Direction [0]=East
      P(PersonPosition [1]) = {0.05, 0.07, 0.1, 0.2, 0.33, 0.07, 0.09, 0.02, 0.03, 0.01, 0.01, 0.01, 0.01}
    Direction [0]=South
      P(PersonPosition [1]) = {0.12, 0.03, 0.65, 0.02, 0.1, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01}
    Direction [0]=Not moving
      P(PersonPosition [1]) = {0.01, 0.01, 0.88, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01}
PersonPosition [0]=12
  Direction [0]
    Direction [0]=North
      P(PersonPosition [1]) = {0.36, 0.4, 0.03, 0.1, 0.03, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01}
    Direction [0]=West
      P(PersonPosition [1]) = {0.24, 0.55, 0.07, 0.03, 0.03, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01}
    Direction [0]=East
      P(PersonPosition [1]) = {0.05, 0.1, 0.07, 0.33, 0.2, 0.09, 0.07, 0.03, 0.02, 0.01, 0.01, 0.01, 0.01}
    Direction [0]=South
      P(PersonPosition [1]) = {0.05, 0.1, 0.4, 0.07, 0.3, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01}
    Direction [0]=Not moving
      P(PersonPosition [1]) = {0.01, 0.88, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01}
PersonPosition [0]=No person
  P(PersonPosition [1]) = {0.88, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01}
```

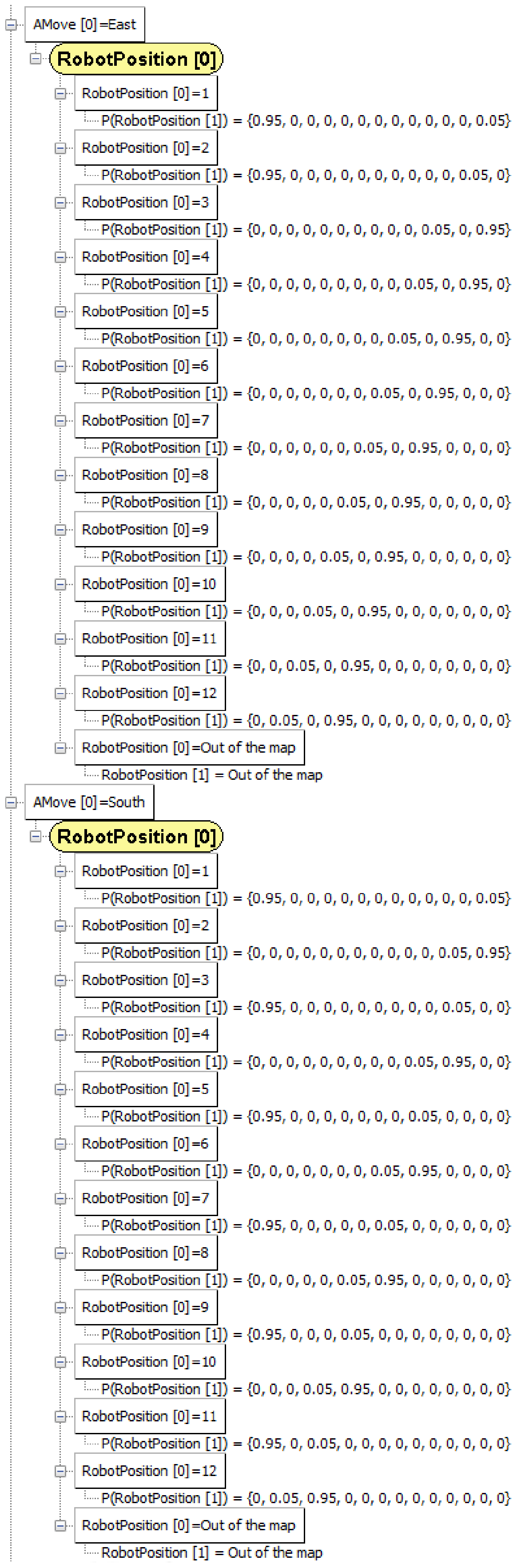
Figure A.4: Transition model of the person's position (3 of 3).

P(RobotPosition [1]) = {Out of the map, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1}



```
AMove [0]
  AMove [0]=North
    RobotPosition [0]
      RobotPosition [0]=1
        P(RobotPosition [1]) = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.95, 0.05}
      RobotPosition [0]=2
        P(RobotPosition [1]) = {0.95, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.05, 0}
      RobotPosition [0]=3
        P(RobotPosition [1]) = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0.95, 0.05, 0, 0}
      RobotPosition [0]=4
        P(RobotPosition [1]) = {0.95, 0, 0, 0, 0, 0, 0, 0, 0, 0.05, 0, 0, 0}
      RobotPosition [0]=5
        P(RobotPosition [1]) = {0, 0, 0, 0, 0, 0, 0, 0.95, 0.05, 0, 0, 0, 0}
      RobotPosition [0]=6
        P(RobotPosition [1]) = {0.95, 0, 0, 0, 0, 0, 0, 0.05, 0, 0, 0, 0, 0}
      RobotPosition [0]=7
        P(RobotPosition [1]) = {0, 0, 0, 0, 0, 0.95, 0.05, 0, 0, 0, 0, 0, 0}
      RobotPosition [0]=8
        P(RobotPosition [1]) = {0.95, 0, 0, 0, 0, 0.05, 0, 0, 0, 0, 0, 0, 0}
      RobotPosition [0]=9
        P(RobotPosition [1]) = {0, 0, 0, 0.95, 0.05, 0, 0, 0, 0, 0, 0, 0, 0}
      RobotPosition [0]=10
        P(RobotPosition [1]) = {0.95, 0, 0, 0.05, 0, 0, 0, 0, 0, 0, 0, 0, 0}
      RobotPosition [0]=11
        P(RobotPosition [1]) = {0, 0.95, 0.05, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
      RobotPosition [0]=12
        P(RobotPosition [1]) = {0.95, 0.05, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
      RobotPosition [0]=Out of the map
        RobotPosition [1] = Out of the map
  AMove [0]=West
    RobotPosition [0]
      RobotPosition [0]=1
        P(RobotPosition [1]) = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.95, 0, 0.05}
      RobotPosition [0]=2
        P(RobotPosition [1]) = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0.95, 0, 0.05, 0}
      RobotPosition [0]=3
        P(RobotPosition [1]) = {0, 0, 0, 0, 0, 0, 0, 0, 0.95, 0, 0.05, 0, 0}
      RobotPosition [0]=4
        P(RobotPosition [1]) = {0, 0, 0, 0, 0, 0, 0, 0.95, 0, 0.05, 0, 0, 0}
      RobotPosition [0]=5
        P(RobotPosition [1]) = {0, 0, 0, 0, 0, 0, 0.95, 0, 0.05, 0, 0, 0, 0}
      RobotPosition [0]=6
        P(RobotPosition [1]) = {0, 0, 0, 0, 0, 0.95, 0, 0.05, 0, 0, 0, 0, 0}
      RobotPosition [0]=7
        P(RobotPosition [1]) = {0, 0, 0, 0, 0.95, 0, 0.05, 0, 0, 0, 0, 0, 0}
      RobotPosition [0]=8
        P(RobotPosition [1]) = {0, 0, 0, 0.95, 0, 0.05, 0, 0, 0, 0, 0, 0, 0}
      RobotPosition [0]=9
        P(RobotPosition [1]) = {0, 0, 0.95, 0, 0.05, 0, 0, 0, 0, 0, 0, 0, 0}
      RobotPosition [0]=10
        P(RobotPosition [1]) = {0, 0.95, 0, 0.05, 0, 0, 0, 0, 0, 0, 0, 0, 0}
      RobotPosition [0]=11
        P(RobotPosition [1]) = {0.95, 0, 0.05, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
      RobotPosition [0]=12
        P(RobotPosition [1]) = {0.95, 0.05, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
      RobotPosition [0]=Out of the map
        RobotPosition [1] = Out of the map
```

Figure A.5: Transition model of the robot's position (1 of 2).

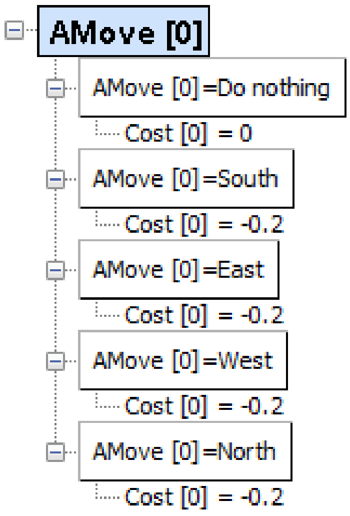


```
AMove [0]=East
  RobotPosition [0]
    RobotPosition [0]=1
      P(RobotPosition [1]) = {0.95, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.05}
    RobotPosition [0]=2
      P(RobotPosition [1]) = {0.95, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.05, 0}
    RobotPosition [0]=3
      P(RobotPosition [1]) = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.05, 0, 0.95}
    RobotPosition [0]=4
      P(RobotPosition [1]) = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0.05, 0, 0.95, 0}
    RobotPosition [0]=5
      P(RobotPosition [1]) = {0, 0, 0, 0, 0, 0, 0, 0, 0.05, 0, 0.95, 0, 0}
    RobotPosition [0]=6
      P(RobotPosition [1]) = {0, 0, 0, 0, 0, 0, 0, 0.05, 0, 0.95, 0, 0, 0}
    RobotPosition [0]=7
      P(RobotPosition [1]) = {0, 0, 0, 0, 0, 0, 0.05, 0, 0.95, 0, 0, 0, 0}
    RobotPosition [0]=8
      P(RobotPosition [1]) = {0, 0, 0, 0, 0, 0.05, 0, 0.95, 0, 0, 0, 0, 0}
    RobotPosition [0]=9
      P(RobotPosition [1]) = {0, 0, 0, 0, 0.05, 0, 0.95, 0, 0, 0, 0, 0, 0}
    RobotPosition [0]=10
      P(RobotPosition [1]) = {0, 0, 0, 0.05, 0, 0.95, 0, 0, 0, 0, 0, 0, 0}
    RobotPosition [0]=11
      P(RobotPosition [1]) = {0, 0, 0.05, 0, 0.95, 0, 0, 0, 0, 0, 0, 0, 0}
    RobotPosition [0]=12
      P(RobotPosition [1]) = {0, 0.05, 0, 0.95, 0, 0, 0, 0, 0, 0, 0, 0, 0}
    RobotPosition [0]=Out of the map
      RobotPosition [1] = Out of the map
AMove [0]=South
  RobotPosition [0]
    RobotPosition [0]=1
      P(RobotPosition [1]) = {0.95, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.05}
    RobotPosition [0]=2
      P(RobotPosition [1]) = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.05, 0.95}
    RobotPosition [0]=3
      P(RobotPosition [1]) = {0.95, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.05, 0, 0}
    RobotPosition [0]=4
      P(RobotPosition [1]) = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0.05, 0.95, 0, 0}
    RobotPosition [0]=5
      P(RobotPosition [1]) = {0.95, 0, 0, 0, 0, 0, 0, 0, 0.05, 0, 0, 0, 0}
    RobotPosition [0]=6
      P(RobotPosition [1]) = {0, 0, 0, 0, 0, 0, 0, 0.05, 0.95, 0, 0, 0, 0}
    RobotPosition [0]=7
      P(RobotPosition [1]) = {0.95, 0, 0, 0, 0, 0, 0.05, 0, 0, 0, 0, 0, 0}
    RobotPosition [0]=8
      P(RobotPosition [1]) = {0, 0, 0, 0, 0, 0.05, 0.95, 0, 0, 0, 0, 0, 0}
    RobotPosition [0]=9
      P(RobotPosition [1]) = {0.95, 0, 0, 0, 0.05, 0, 0, 0, 0, 0, 0, 0, 0}
    RobotPosition [0]=10
      P(RobotPosition [1]) = {0, 0, 0, 0.05, 0.95, 0, 0, 0, 0, 0, 0, 0, 0}
    RobotPosition [0]=11
      P(RobotPosition [1]) = {0.95, 0, 0.05, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
    RobotPosition [0]=12
      P(RobotPosition [1]) = {0, 0.05, 0.95, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
    RobotPosition [0]=Out of the map
      RobotPosition [1] = Out of the map
```

Figure A.6: Transition model of the robot's position (2 of 2).

Tolerance = 0.01

Discount Factor = 0.99


AMove [0]

AMove [0]=Do nothing
Cost [0] = 0

AMove [0]=South
Cost [0] = -0.2

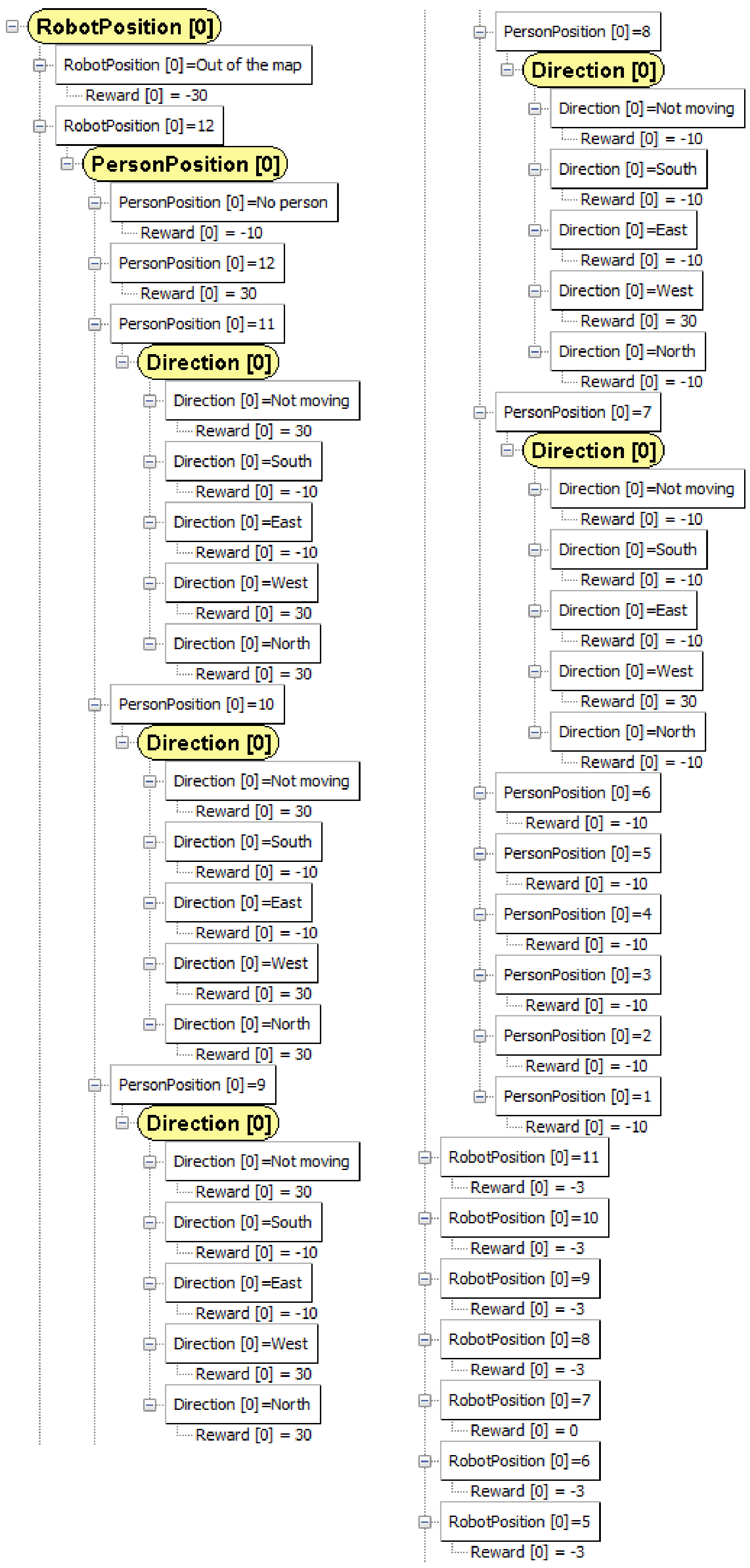
AMove [0]=East
Cost [0] = -0.2

AMove [0]=West
Cost [0] = -0.2

AMove [0]=North
Cost [0] = -0.2

Figure A.7: Reward function (1 of 3).

Figure A.8 is the reward function when two restricted/dangerous places exist. If there is only one, RobotPosition [0] = 12 $\rightarrow$ Reward [0] = -3



```
RobotPosition [0]
    RobotPosition [0]=Out of the map
        Reward [0] = -30
    RobotPosition [0]=12
        PersonPosition [0]
            PersonPosition [0]=No person
                Reward [0] = -10
            PersonPosition [0]=12
                Reward [0] = 30
            PersonPosition [0]=11
                Direction [0]
                    Direction [0]=Not moving
                        Reward [0] = 30
                    Direction [0]=South
                        Reward [0] = -10
                    Direction [0]=East
                        Reward [0] = -10
                    Direction [0]=West
                        Reward [0] = 30
                    Direction [0]=North
                        Reward [0] = 30
            PersonPosition [0]=10
                Direction [0]
                    Direction [0]=Not moving
                        Reward [0] = 30
                    Direction [0]=South
                        Reward [0] = -10
                    Direction [0]=East
                        Reward [0] = -10
                    Direction [0]=West
                        Reward [0] = 30
                    Direction [0]=North
                        Reward [0] = 30
            PersonPosition [0]=9
                Direction [0]
                    Direction [0]=Not moving
                        Reward [0] = 30
                    Direction [0]=South
                        Reward [0] = -10
                    Direction [0]=East
                        Reward [0] = -10
                    Direction [0]=West
                        Reward [0] = 30
                    Direction [0]=North
                        Reward [0] = 30
            PersonPosition [0]=8
                Direction [0]
                    Direction [0]=Not moving
                        Reward [0] = -10
                    Direction [0]=South
                        Reward [0] = -10
                    Direction [0]=East
                        Reward [0] = -10
                    Direction [0]=West
                        Reward [0] = 30
                    Direction [0]=North
                        Reward [0] = -10
            PersonPosition [0]=7
                Direction [0]
                    Direction [0]=Not moving
                        Reward [0] = -10
                    Direction [0]=South
                        Reward [0] = -10
                    Direction [0]=East
                        Reward [0] = -10
                    Direction [0]=West
                        Reward [0] = 30
                    Direction [0]=North
                        Reward [0] = -10
            PersonPosition [0]=6
                Reward [0] = -10
            PersonPosition [0]=5
                Reward [0] = -10
            PersonPosition [0]=4
                Reward [0] = -10
            PersonPosition [0]=3
                Reward [0] = -10
            PersonPosition [0]=2
                Reward [0] = -10
            PersonPosition [0]=1
                Reward [0] = -10
    RobotPosition [0]=11
        Reward [0] = -3
    RobotPosition [0]=10
        Reward [0] = -3
    RobotPosition [0]=9
        Reward [0] = -3
    RobotPosition [0]=8
        Reward [0] = -3
    RobotPosition [0]=7
        Reward [0] = 0
    RobotPosition [0]=6
        Reward [0] = -3
    RobotPosition [0]=5
        Reward [0] = -3
```

Figure A.8: Reward function (2 of 3).

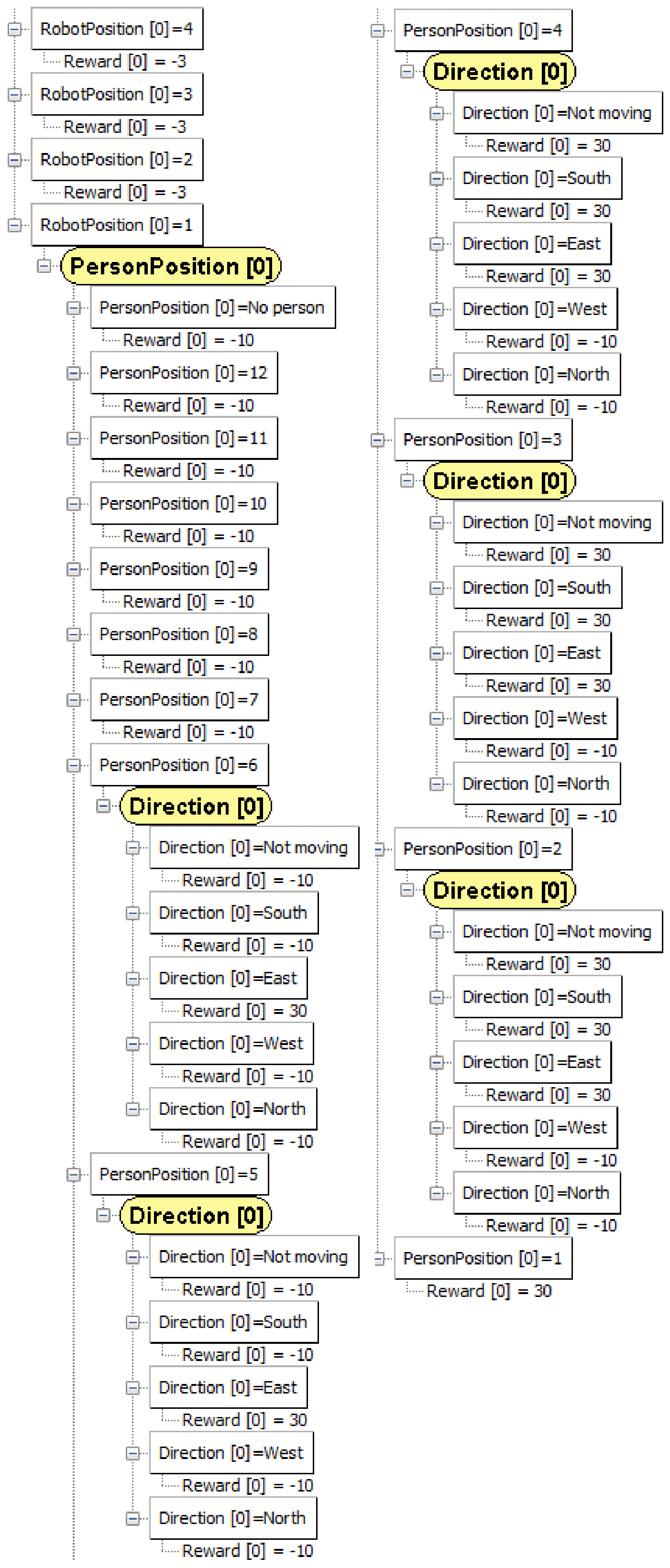
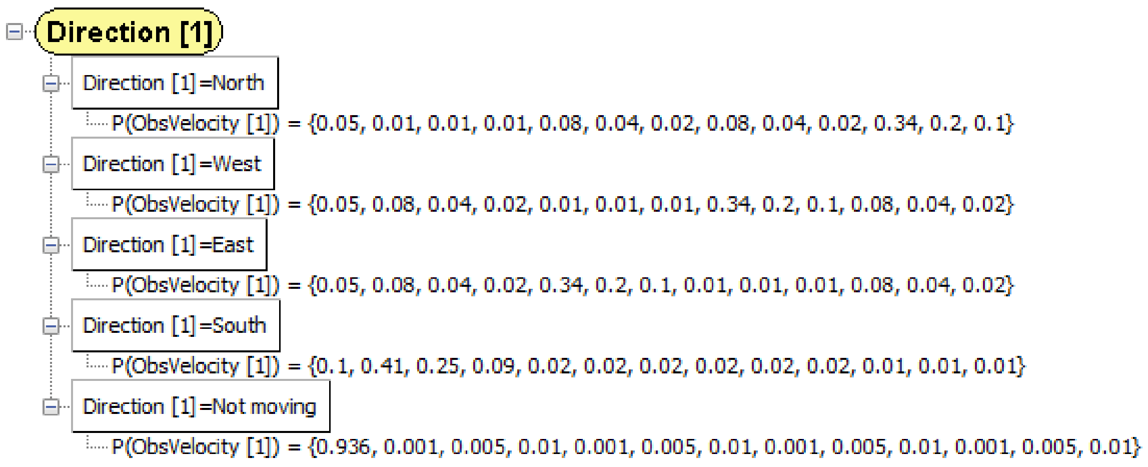

```
RobotPosition [0]=4
    Reward [0] = -3
RobotPosition [0]=3
    Reward [0] = -3
RobotPosition [0]=2
    Reward [0] = -3
RobotPosition [0]=1
    PersonPosition [0]
        PersonPosition [0]=No person
            Reward [0] = -10
        PersonPosition [0]=12
            Reward [0] = -10
        PersonPosition [0]=11
            Reward [0] = -10
        PersonPosition [0]=10
            Reward [0] = -10
        PersonPosition [0]=9
            Reward [0] = -10
        PersonPosition [0]=8
            Reward [0] = -10
        PersonPosition [0]=7
            Reward [0] = -10
        PersonPosition [0]=6
            Direction [0]
                Direction [0]=Not moving
                    Reward [0] = -10
                Direction [0]=South
                    Reward [0] = -10
                Direction [0]=East
                    Reward [0] = 30
                Direction [0]=West
                    Reward [0] = -10
                Direction [0]=North
                    Reward [0] = -10
        PersonPosition [0]=5
            Direction [0]
                Direction [0]=Not moving
                    Reward [0] = -10
                Direction [0]=South
                    Reward [0] = -10
                Direction [0]=East
                    Reward [0] = 30
                Direction [0]=West
                    Reward [0] = -10
                Direction [0]=North
                    Reward [0] = -10
        PersonPosition [0]=4
            Direction [0]
                Direction [0]=Not moving
                    Reward [0] = 30
                Direction [0]=South
                    Reward [0] = 30
                Direction [0]=East
                    Reward [0] = 30
                Direction [0]=West
                    Reward [0] = -10
                Direction [0]=North
                    Reward [0] = -10
        PersonPosition [0]=3
            Direction [0]
                Direction [0]=Not moving
                    Reward [0] = 30
                Direction [0]=South
                    Reward [0] = 30
                Direction [0]=East
                    Reward [0] = 30
                Direction [0]=West
                    Reward [0] = -10
                Direction [0]=North
                    Reward [0] = -10
        PersonPosition [0]=2
            Direction [0]
                Direction [0]=Not moving
                    Reward [0] = 30
                Direction [0]=South
                    Reward [0] = 30
                Direction [0]=East
                    Reward [0] = 30
                Direction [0]=West
                    Reward [0] = -10
                Direction [0]=North
                    Reward [0] = -10
        PersonPosition [0]=1
            Reward [0] = 30
```

Figure A.9: Reward function (3 of 3).

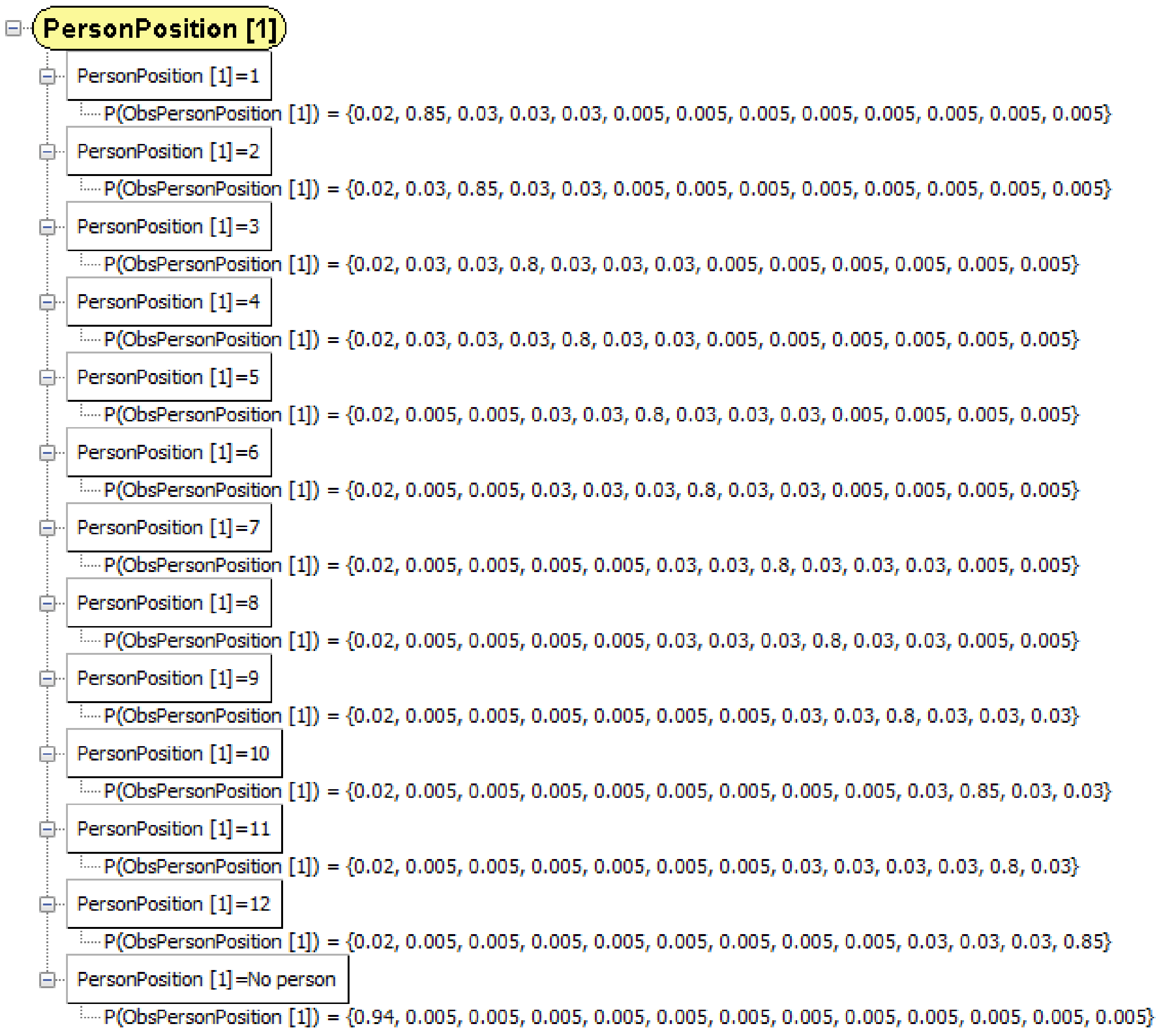
P(ObsVelocity [1]) = {Not moving, South higher, South 0.2-0.7, South 0-0.2, East higher, East 0.2-0.7, East 0-0.2, West higher, West 0.2-0.7, West 0-0.2, North higher, North 0.2-0.7, North 0-0.2}



```
Direction [1]
    Direction [1]=North
        P(ObsVelocity [1]) = {0.05, 0.01, 0.01, 0.01, 0.08, 0.04, 0.02, 0.08, 0.04, 0.02, 0.34, 0.2, 0.1}
    Direction [1]=West
        P(ObsVelocity [1]) = {0.05, 0.08, 0.04, 0.02, 0.01, 0.01, 0.01, 0.34, 0.2, 0.1, 0.08, 0.04, 0.02}
    Direction [1]=East
        P(ObsVelocity [1]) = {0.05, 0.08, 0.04, 0.02, 0.34, 0.2, 0.1, 0.01, 0.01, 0.01, 0.08, 0.04, 0.02}
    Direction [1]=South
        P(ObsVelocity [1]) = {0.1, 0.41, 0.25, 0.09, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.01, 0.01, 0.01}
    Direction [1]=Not moving
        P(ObsVelocity [1]) = {0.936, 0.001, 0.005, 0.01, 0.001, 0.005, 0.01, 0.001, 0.005, 0.01, 0.001, 0.005, 0.01}
```

Figure A.10: Observation model of the person's velocity.

P(ObsPersonPosition [1]) = {No person, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1}



```
PersonPosition [1]
    PersonPosition [1]=1
        P(ObsPersonPosition [1]) = {0.02, 0.85, 0.03, 0.03, 0.03, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005}
    PersonPosition [1]=2
        P(ObsPersonPosition [1]) = {0.02, 0.03, 0.85, 0.03, 0.03, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005}
    PersonPosition [1]=3
        P(ObsPersonPosition [1]) = {0.02, 0.03, 0.03, 0.8, 0.03, 0.03, 0.03, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005}
    PersonPosition [1]=4
        P(ObsPersonPosition [1]) = {0.02, 0.03, 0.03, 0.03, 0.8, 0.03, 0.03, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005}
    PersonPosition [1]=5
        P(ObsPersonPosition [1]) = {0.02, 0.005, 0.005, 0.03, 0.03, 0.8, 0.03, 0.03, 0.03, 0.005, 0.005, 0.005, 0.005}
    PersonPosition [1]=6
        P(ObsPersonPosition [1]) = {0.02, 0.005, 0.005, 0.03, 0.03, 0.03, 0.8, 0.03, 0.03, 0.005, 0.005, 0.005, 0.005}
    PersonPosition [1]=7
        P(ObsPersonPosition [1]) = {0.02, 0.005, 0.005, 0.005, 0.005, 0.03, 0.03, 0.8, 0.03, 0.03, 0.03, 0.005, 0.005}
    PersonPosition [1]=8
        P(ObsPersonPosition [1]) = {0.02, 0.005, 0.005, 0.005, 0.005, 0.03, 0.03, 0.03, 0.8, 0.03, 0.03, 0.005, 0.005}
    PersonPosition [1]=9
        P(ObsPersonPosition [1]) = {0.02, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.03, 0.03, 0.8, 0.03, 0.03, 0.03}
    PersonPosition [1]=10
        P(ObsPersonPosition [1]) = {0.02, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.03, 0.85, 0.03, 0.03}
    PersonPosition [1]=11
        P(ObsPersonPosition [1]) = {0.02, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.03, 0.03, 0.03, 0.03, 0.8, 0.03}
    PersonPosition [1]=12
        P(ObsPersonPosition [1]) = {0.02, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.03, 0.03, 0.03, 0.85}
    PersonPosition [1]=No person
        P(ObsPersonPosition [1]) = {0.94, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005}
```

Figure A.11: Observation model of the person's position.

Since no uncertainty is considered when observing the robot's position,
ObsRobotPosition [1] = RobotPosition [1]