Statistical Task Modeling of Activities of Daily Living for Rehabilitation

Émilie Michèle Déborah Jean-Baptiste

School of Engineering

University of Birmingham

A thesis submitted for the degree of

Doctor of Philosophy

February 2016

Acknowledgements

I would like to thank my supervisor, Prof. Martin Russell, for his supervision throughout my PhD. His guidance has been essential to my research. He has always found the time to meet me and to answer my questions. I am very grateful to him. I am also grateful to Dr. Pia Rotshtein for her support and her insights on cognitive decision models.

I also would like to thank Prof. Alan Wing for his excellent team management. The success of the final CogWatch prototype was the result of a joint effort. For that I would like to thank the CogWatch team, which has provided a highly motivating environment for me to carry out this research.

Finally, my thanks go to my family and my unwavering source of inspiration, for their continuous encouragement and support.

Abstract

Stroke survivors suffering from cognitive deficits experience difficulty completing their daily self-care activities. The latter are referred to as activities of daily living (ADL) [54]. The resulting loss of independence makes them rely on caregivers to help them go through their daily routine. However, such reliance on caregivers may conflict with their need for privacy and willingness to keep a control over their life. A possible solution to tackle this issue is the development of an assistive or rehabilitation system.

Ideally, the aim of such a system would be to deliver the same services as a human caregiver. For example, the system could provide meaningful recommendations or hints to stroke survivors during a task, so they have a higher probability of successfully continuing or completing it. In order to fulfill such an aim, an assistive or rehabilitation system would need to monitor stroke survivors' behavior, constantly keep track of what they do during the task, and **plan** the strategies they should follow to increase their task completion.

The module in charge of planning is really important in this process. Indeed, this module interacts with stroke survivors or any users during the task, analyzes how far they might be in the completion of this task, and infers what they should do to succeed it. To do so, the planning module needs to receive information about users' behavior, and be trained to "learn" how to take decisions that could guide them. In the case where the information it receives are incorrect, the main challenge of the planning module is to cope with the uncertainty in its inputs, and still be able to take the right decisions as far as users are concerned.

Different decision theory models exist and could be implemented, for example cognitive models [22; 23] or statistical models such as Markov Decision Process (MDP) [86] or Partially Observable Markov Decision Process (POMDP) [52]. The MDP assumes full observability as far as the system's environment is concerned, while the POMDP provides a rich and natural framework to model sequential decision-making problems under uncertainty. Hence, it is potentially a good candidate for a system whose aim is to guide stroke survivors during ADL, even if the information it receives is potentially erroneous.

Since a POMDP-based system acknowledges the fact that the information it receives about a user may be incorrect, it maintains a probability distribution over all potential situations this user might be in. These probability distributions are referred to as "belief states", and the belief state space containing all belief states is infinite.

Many methods can be implemented in order to solve a POMDP. In the case of a system in charge of guiding users, to solve a POMDP means to find what are the optimal recommendations to send to a user during a task. Exact POMDP solution methods are known to be intractable, due to their aim of computing the optimal recommendation for all possible belief states contained in the belief state space [103]. A way to sidestep this intractability is to implement approximation algorithms by considering only a finite set of belief points, referred to as "belief subspace".

In the work presented in this thesis, a belief state representation based on the MDP reduced state space is explained. We will show how restricting the growth of the MDP state space helps maintain the belief state's dimensionality at a relatively small size. The thesis also analyzes the potential for improving the strategy selection process during execution. In the case of a POMDP-based system, since strategies are found only for a subspace of belief states, this may lead the system to face the challenge of deciding what strategy to take in a situation it has not been trained for. In this case, we investigated the effect of different methods, which can be used during execution to approximate an unknown belief state to a belief state the system has seen during training.

Overall, this work represents an important step forward in the development of an artificial intelligent planning system designed to guide users suffering from cognitive deficits during their activities of daily living.

Nomenclature

General

- AI - Artificial intelligence
- MDP - Markov Decision Process
- $POMDP$ - Partially Observable Markov Decision Process
- MC - Monte Carlo
- NL - Numerical label
- NNS - Nearest Neighbor Search
- $\mathbb{N}$ - Set of natural numbers
- $P(.)$ - Probability
- $P(.|.)$ - Conditional probability

Rehabilitation

- $AADS$ - Apraxia or action disorganization syndrome
- ADL - Activity of daily living
- EF - Errorfull
- EL - Errorless

CogWatch system

- CW - CogWatch
- $SimU$ - Simulated User
- ARS - Action recognition system
- TM - Task Manager
- APM - Action policy module
- ERM - Error recognition module
- λ - SimU's compliance probability
- δ - SimU's probability to forget
- a_u - User's action
- o - ARS's output
- ω - Task Manager's prompt
- μ - Task Manager's recommendation (i.e., system's action)
- e - Task Manager's interpretation of user's error
- Θ - Signal from virtual Cue Selector
- ζ - Cue from Cue Selector
- r_s - User's state representation
- s_d - User's history of action

Task formalism

- BT - Black tea
- BTS - Black tea with sugar
- WT - White tea
- WTS - White tea with sugar
- BTr - Button trigger
- AD - Addition error
- AN - Anticipation error
- OM - Omission error
- PE - Perplexity error
- PsE - Perseveration error
- QT - Quantity error
- FE - Fatal error
- NFE - Non fatal error
- NE - Not an error

Markov Decision Process

- A - Set of recommendations (i.e., set of system's actions)
- μ_t - TM's recommendation at step t (i.e., system's action at step t)
- S - Set of states
- s_t - State at step t
- $c(s, \mu)$ - Cost incurred when taking μ in state s

- π - Policy, with π: $S \rightarrow A$
- ϕ - Geometric discount factor
- $V^{\pi}(s)$ - Value function for policy π
- $V^{*}(s)$ - Optimal value function
- $Q^{\pi}(s, \mu)$ - Q-function for policy π
- $Q^{*}(s, \mu)$ - Optimal Q-function
- $N(s, \mu)$ - Number of times μ is taken in state s

Partially Observable Markov Decision Process

- O - Set of observations
- Z - Observation probability
- B - Set of belief states
- b - Belief state (i.e., probability distribution over states)
- b_t - Belief state at step t
- $b(s)$ - Probability to be in state s
- $c(b, \mu)$ - Cost for taking μ in belief state b
- $V^{\pi}(b)$ - Value function for policy π
- $Q^{\pi}(b, \mu)$ - Q-function for policy π

Contents

Nomenclature **v**

Contents **ix**

List of Figures **xiv**

1 Introduction **1**

1.1 Overview 1

1.1.1 General problem definition 4

1.1.2 Applications 5

1.1.3 Assistive Systems' Architecture 9

1.1.3.1 Monitoring module 10

1.1.3.2 Action Recognition System 11

1.1.3.3 Task Manager 12

1.1.3.4 Cue Generation Module 14

1.1.4 Similarities with a Spoken Dialogue System 14

1.2 Contribution 16

1.2.1 State representation in the Task Manager 16

1.2.2 MDP and POMDP implementation in CogWatch 17

1.2.3 Policy optimization 18

1.2.4 Users' error detection 18

1.2.5 User Simulation and virtualization of CogWatch 18

1.2.6 Nearest neighbor search under uncertainty 19

1.3 Thesis structure 20

2 Decision Theory Models 21

2.1 Introduction 21

2.2 Cognitive models 22

2.2.1 Hierarchical approach 22

2.2.2 Contention Scheduling 24

2.2.3 Neural Network 26

2.3 Statistical Models 27

2.3.1 Markov Decision Process 28

2.3.1.1 MDP policy optimization 30

2.3.1.2 Value iteration algorithm 33

2.3.1.3 Monte Carlo algorithm 34

2.3.1.4 MDP-based task management 36

2.3.2 Partially Observable Markov Decision Process 40

2.3.2.1 Exact POMDP optimization 42

2.3.2.2 Approximate POMDP optimization 44

2.3.2.3 POMDP-based task management 48

2.4 Summary 50

3 The CogWatch Project 51

3.1 Introduction 51

3.2 Task definition 52

3.2.1 Actions tree 52

3.2.2 Error types 53

3.3 CogWatch architecture 54

3.3.1 Sensors and sensorized objects 56

3.3.2 Action Recognition System 58

3.3.3 Task Manager 59

3.3.4 Cue Selector 60

3.3.5 Interactive buttons 62

3.4 Implementation and definition of errors in the tea-making task . . 63

3.5 Summary 64

4 Simulation 68
4.1 Introduction . . . 68
4.2 Simulated User ($SimU\alpha$) . . . 68
4.2.1 Simulated user's architecture . . . 69
4.2.2 Simulated User's process (example) . . . 72
4.3 Simple actions generator ($SimU\ \beta$) . . . 73
4.4 Virtualization of CogWatch . . . 74
4.5 Training via user simulation . . . 75
4.6 Evaluation via user simulation . . . 77
4.7 Summary . . . 78

5 MDP-based Task Manager 79
5.1 Introduction . . . 79
5.2 CogWatch MDP-based Task Manager . . . 79
5.2.1 Task formalism . . . 79
5.2.2 State representation . . . 80
5.2.2.1 APM reduced state representation . . . 81
5.2.2.2 ERM state representation . . . 83
5.2.3 Policy representation . . . 84
5.2.4 Error recognition . . . 85
5.3 Training . . . 85
5.3.1 MDP framework adaptation to CogWatch . . . 86
5.3.2 APM Policy optimization . . . 88
5.4 Evaluation . . . 88
5.4.1 Trials with stroke survivors . . . 88
5.4.2 Trials with simulated users . . . 91
5.5 Results . . . 94
5.5.1 Results obtained with stroke survivors . . . 94
5.5.2 Results obtained via simulation . . . 97
5.6 Summary . . . 103

6 POMDP-based Task Manager 104
6.1 Introduction . . . 104

6.2 CogWatch POMDP-based Task Manager . . . 104
6.2.1 Task formalism . . . 105
6.2.2 Belief state representation . . . 106
6.2.2.1 APM perspective . . . 106
6.2.2.2 ERM perspective . . . 107
6.2.3 Belief update . . . 107
6.3 Training and evaluation . . . 107
6.3.1 POMDP-based APM . . . 108
6.3.1.1 POMDP framework adaptation to CogWatch . . 108
6.3.1.2 APM Policy optimization . . . 109
6.3.2 ERM for POMDP-based Task Manager . . . 110
6.3.3 Training at varying ARS error rates . . . 111
6.3.4 Evaluation . . . 113
6.3.4.1 Trials with simulated users . . . 113
6.4 Results . . . 116
6.4.1 APM performance only . . . 116
6.4.1.1 First scenario . . . 116
6.4.1.2 Second scenario . . . 120
6.4.2 ERM performance only . . . 125
6.4.3 Joint performance . . . 127
6.5 Summary . . . 128

7 Nearest Neighbor Search in the POMDP-based APM 129

7.1 Introduction . . . 129
7.1.1 Classifier . . . 131
7.1.1.1 Support Vector Machine (SVM) . . . 131
7.1.2 Metrics . . . 131
7.1.3 Tree based approach structure . . . 132
7.1.3.1 K-d tree . . . 132
7.2 SciMK . . . 133
7.3 Evaluation . . . 137
7.3.1 Trials with a simulated user . . . 137
7.3.2 Evaluation of the APM . . . 137

7.4 Results 138
7.4.1 APM performance only 138
7.4.1.1 First scenario 138
7.4.1.2 Second scenario 140
7.4.1.3 Third scenario 147
7.5 Summary 150

8 Conclusion **151**
8.1 Thesis summary 151
8.2 Open problems and Future work 154

References **158**

List of Figures

1.1 Rehabilitation system's architecture 9

1.2 Spoken dialogue system architecture 15

2.1 Hierarchical representation of a tea making task 23

2.2 Principal components of the contention schedulling [22] 25

2.3 Reccurent network architecture for action selection [15; 22] 26

2.4 Backup diagram for V^π [104] - Note that μ could be written $\pi(s)$. 31

2.5 Backup diagram for Q^π [104] - Note that μ' could be written $\pi(s')$. 32

2.6 Partial backup diagram of the belief state update 41

2.7 POMDP exact value function representation 43

2.8 MC policy optimisation algorithm 47

3.1 CogWatch architecture, APM: Action Policy Module, ERM: Error Recognition Module 55

3.2 A jug fitted with a CogWatch Instrumented Coaster (CIC) and an 'open' CIC, showing the accelerometer, PIC microcontroller, Bluetooth module and battery [33] 56

3.3 Kinect^{TM} user interface with skeleton view of arms and hands [33] 57

3.4 Screen shot showing the output of the real-time Action Recognition System [73] 59

3.5 Screen shot showing the CogWatch patient's interface 61

4.1 The architecture of the Simulated User ($SimU\alpha$). 69

4.2 Architecture of the Virtualization of CogWatch. 74

5.1 Structure of CogWatch with the MDP-based Task Manager. . . . 80

5.2 Example of state representation in the MDP-based Task Manager 83
5.3 MDP-based APM state mapping, with $A_{1,4,7}$ some of the actions the user can perform, and μ^* the recommendation selected by the APM. 84
5.4 Representation of the MDP state space for Black tea. 87
5.5 Structure of CogWatch with the MDP-based Task Manager during evaluation with stroke survivors. 89
5.6 Diagram of the system used to evaluate the MDP-based Task Manager 91
5.7 Virtual ARS confusion matrices. (a - ARS error rate 10%, (b - ARS error rate 20%, (c - ARS error rate 30%. Class 1: "Fill kettle", Class 2: "Boil water", Class 3: "Add teabag", Class 4: "Pour kettle", Class 5:"Add sugar", Class 6:"Add milk", Class 7: "Stir", Class 8:"Remove teabag", Class 9:"Pour water from jug to cup", Class 10:"Toying with boiling water", Class 11: ∅ (no action performed or detected). 93
5.8 Number of specific types of errors made by users over 96 trials. . . 95
5.9 Total number of errors made by users over 96 trials. 95
5.10 *SimUα* success rate at varying levels of compliance during "Black tea", "Black tea with sugar" and "White tea with sugar". Virtual ARS error rate: 0%. 97
5.11 *SimUα* and *SimUβ*'s success rate at varying ARS error rate - "Black tea". 100
5.12 *SimUα* and *SimUβ*'s success rate at varying ARS error rate - "Black tea with sugar". 100
5.13 *SimUα* and *SimUβ*'s success rate at varying ARS error rate - "White tea with sugar". 101

6.1 Structure of CogWatch with the POMDP-based Task Manager. . 105
6.2 Diagram of the system used to evaluate the POMDP-based Task Manager 114

6.3 SimU success rate at varying ARS error rate during “Black tea”. POMDP-based APM using euclidean distance. ARS confusion matrix observable. 117

6.4 SimU success rate at varying ARS error rate during “Black tea with sugar”. POMDP-based APM using Euclidean distance. ARS confusion matrix observable. 118

6.5 SimU success rate at varying ARS error rate during “White tea with sugar”. POMDP-based APM using Euclidean distance. ARS confusion matrix observable. 119

6.6 Confusion matrices as implemented in the virtual ARS. (a - ARS error rate 7.5% (approximation used: 8%), (b - ARS error rate 20%, (c - ARS error rate 29%. Class 1: “Fill kettle”, Class 2: “Boil water”, Class 3: “Add teabag”, Class 4: “Pour kettle”, Class 5:“Add sugar”, Class 6:“Add milk”, Class 7: “Stir”, Class 8:“Remove teabag”, Class 9:“Pour water from jug to cup”, Class 10:“Toying with boiling water”, Class 11: ∅ (no action performed or detected). 121

6.7 Confusion matrices as seen by the POMDP-based APM. (a - ARS error rate 7.7%, (b - ARS error rate 20%, (c - ARS error rate 26.5%. Class 1: “Fill kettle”, Class 2: “Boil water”, Class 3: “Add teabag”, Class 4: “Pour kettle”, Class 5:“Add sugar”, Class 6:“Add milk”, Class 7: “Stir”, Class 8:“Remove teabag”, Class 9:“Pour water from jug to cup”, Class 10:“Toying with boiling water”, Class 11: ∅ (no action performed or detected). 122

6.8 *SimUα* success rate at varying ARS error rate during “Black tea”. POMDP-based APM using Euclidean distance. POMDP-based TM has an approximation of ARS confusion matrix. 123

6.9 *SimUα* success rate at varying ARS error rate during “Black tea with sugar”. POMDP-based APM using Euclidean distance. POMDP-based TM has an approximation of ARS confusion matrix. 124

6.10 *SimUα* success rate at varying ARS error rate during “White tea with sugar”. POMDP-based APM using Euclidean distance. POMDP-based TM has an approximation of ARS confusion matrix. 124

6.11 ERM evaluation: percentage of correct detection, false positive and false negative. ARS error rate: 30%. Task: “Black tea” 126
6.12 ERM evaluation: percentage of semantically correct prompts delivered to a simulated user. ARS error rate: 30%. Task: “Black tea”. 127

7.1 Detailed diagram of the system used to evaluate the POMDP-based Task Manager 130
7.2 Belief state transformation. 134
7.3 SimU α success rate at varying ARS error rate and NNS techniques. ARS confusion matrix is observable. Task: “Black tea”. . 139
7.4 SimU α success rate at varying ARS error rate and NNS techniques. ARS confusion matrix is observable. Task: “Black tea with sugar”. 139
7.5 SimU α success rate at varying ARS error rate and NNS techniques. ARS confusion matrix is observable. Task: “White tea with sugar”. 140
7.6 SimU α success rate at varying ARS error rate and NNS techniques. POMDP-based TM has an approximation of ARS confusion matrix. Task: “Black tea”. 141
7.7 SimU α success rate at varying ARS error rate and NNS techniques. POMDP-based TM has an approximation of ARS confusion matrix. Task: “Black tea with sugar”. 142
7.8 SimU α success rate at varying ARS error rate and NNS techniques. POMDP-based TM has an approximation of ARS confusion matrix. Task: “White tea with sugar”. 142
7.9 Comparison of neighbors selected by different NNS techniques. (1) and (3): Comparison of probabilistic distributions. (2) and (4) : Comparison of NLs vectors. 146

7.10 Confusion matrix as implemented in the virtual ARS. ARS error rate 24.1%. Class 1: "Fill kettle", Class 2: "Boil water", Class 3: "Add teabag", Class 4: "Pour kettle", Class 5: "Add sugar", Class 6: "Add milk", Class 7: "Stir", Class 8: "Remove teabag", Class 9: "Pour water from jug to cup", Class 10: "Toying with boiling water", Class 11: ∅ (no action performed or detected). 148

7.11 Confusion matrix seen by the POMDP-based APM. ARS error rate 19.7%. Class 1: "Fill kettle", Class 2: "Boil water", Class 3: "Add teabag", Class 4: "Pour kettle", Class 5: "Add sugar", Class 6: "Add milk", Class 7: "Stir", Class 8: "Remove teabag", Class 9: "Pour water from jug to cup", Class 10: "Toying with boiling water", Class 11: ∅ (no action performed or detected). 148

7.12 *SimU* success rate. ARS confusion matrix implemented Figure 7.10. ARS confusion matrix seen by POMDP-based APM: Figure 7.11. Task: "Black tea with sugar". 149

Chapter 1

Introduction

1.1 Overview

Each year, there are more than 100,000 new stroke cases in the UK [4], with over half of all stroke survivors depending on others to carry out Activities of Daily Living (ADL). Such activities can be defined as sequences of actions related to specific tasks that one may need to go through in order to live independently (for example cooking, grooming, teeth-brushing or making a drink). Stroke survivors face difficulties due to the loss of physical and cognitive functions caused by Apraxia or Action Disorganization Syndrome (AADS) [6; 76; 117]. In [11], it has been estimated that such cognitive deficits affect 46% of stroke survivors during ADL. For example, when aiming to prepare a cup of tea, they may perform a wrong sequence of actions: forget to pour water in the kettle then switch the latter on, skip steps, or misuse objects with possible safety implications. It also has been observed that they could make errors such as hesitating a long period of time trying to decide what to do next, repeatedly verifying if the water has boiled, putting ingredients over the cup without adding them, or adding water into the cup then immediately putting it back into the kettle. These errors can relate to the defective use of real tools and objects [81], the inability to correctly select appropriate tools for a task [31], or the inability to complete sequences of actions [72].

These impairments are typically associated with loss of action knowledge [30], attention and executive function deficits [43], and/or loss of object knowledge [101]. Such acquired neurological deficits have a direct impact on stroke survivors' daily life, relatives, caregivers and society as a whole.

In order to mitigate the effects of AADS and improve individuals' conditions, rehabilitation interventions can be provided [21; 57]. In this case, rehabilitation interventions can be defined as processes that *enable people who are disabled by injury or disease to achieve their optimum physical, psychological and social well-being* [66]. During such interventions, skilled clinicians are required to provide appropriate assistance, guide individuals through ADL by observing their behavior, and prompting them when necessary. Two main schools of thought exist as to how rehabilitation should be provided. In the case of ADL, when prompting cognitively impaired persons, clinicians can follow an errorless (EL) or errorfull (EF) technique. In the case of EL technique, the choice is made to prompt them at each step of the task, and prevent the occurrence of any error. This technique was promoted by Werd et al. in the case of dementia, which is an umbrella term for symptoms such as loss of memory, language or other intellectual capacities [112]. When the EF technique is applied, the individuals receiving rehabilitation take all the initiatives during the task, and receive cues only when they make mistakes. Middleton et al. and van Heugten promoted this technique in the case of stroke [68; 108]. The choice to follow an EL or EF technique when providing rehabilitation is a clinical decision that only specialists can take.

However, hiring such specialists comes at a price: an economical and human one. In the UK, the cost of stroke to society is estimated to be £8.9 billion a year, with about half linked to indirect costs of ongoing support [18]. Beyond this economical aspect, the loss of independence affecting stroke survivors' personal privacy also needs to be highlighted. Indeed, the difficulties they face on a daily basis increase their reliance on caregivers, who may directly go to their home to deliver rehabilitation and recovery care. Some cognitively impaired persons may perceive this situation as an invasion of their personal space, and be unwilling to accept this over-reliance on caregivers as a long-term solution [85].

Therefore, there is a need for technology that can take decisions, and provide assistance automatically; with the overall goal of reducing AADS related disabilities, and help stroke survivors regain self-sufficiency and independence while keeping their dignity.
Could this be met by the use of computer technology? Could we design a system that could easily be installed in the home of stroke survivors, and act as a rehabilitation or assistive system? Could we make such a system *artificially intelligent*, so it could automatically track their progress through ADL, detect if they make errors, and provide meaningful recommendations to help them properly complete a task? To achieve this, the rehabilitation system may be composed of different modules, each having a specific aim to be fulfilled for the overall goal to be reached.

The objective of the research presented in this thesis is to describe how such an artificial intelligent (AI) rehabilitation system can be conceptualized. The focus will be put on the module in charge of identifying stroke survivors' error during ADL, and planning the optimal strategies they should follow at each step to properly continue or succeed a task. More precisely, the work and analysis presented will provide answers and suggestions to the following questions:

- How can activities of daily living be modeled by an AI-based rehabilitation system?
- How can pre-existing AI-based techniques be adapted, in order to fit the characteristics of such a system?
- How can the system be trained, in order to "learn" how to take decisions and guide stroke survivors during a task?
- What search technique can be applied, in order to allow the system to reuse what it has "learnt", and select the best recommendation stroke survivors should follow in order to succeed at a task?
- What method could be used to enable automatic error detection in stroke survivors' behaviors during a task?

To be able to address those challenges is the key which, one day, will allow stroke survivors to receive assistance and rehabilitation at home, without compromising their privacy and their need to be independent.

1.1.1 General problem definition

To understand how a system could take decisions by itself, we might begin by analyzing the common decision-making processes we go through on a daily basis. When involved in an activity that can be completed in different ways, we go through a pattern of actions, modifying our environment (i.e., collection of surrounding elements and objects) until we reach our goal. But how do we choose the sequence of actions we believe will make us complete the task properly? Goal, environment, rules and consequences of our actions, are parameters that are often taken into account when decisions must be taken. The plan we follow to go through a task is generally driven by what we want to achieve (i.e., goal), what is at our disposal to achieve it (i.e., elements from the environment, objects), what can be done and how (i.e., rules) and the long-term impact of our present actions on the future.

For example, when playing chess, to move a chess piece without thinking ahead of the potential risks and opportunities generated by our current action may result in a poor game. The relationship between present and future decisions and their related outcomes is a key element in decision-making. Hence, when given a goal, context, rules and notion of potential consequences of actions, we can imagine a system able to follow a similar model by implementing Artificial Intelligence (AI) techniques. A technology able to do so would then have the ability to "learn" how to take decisions by itself and plan strategies to guide stroke survivors.

In the next sections, we will go through the background related to different applications for which assist systems have been designed. Then we will explain and compare these systems' functionality and main components.

1.1.2 Applications

When systems are able to monitor users with cognitive deficits, they have the potential to reduce the number of consultations with specialists, assess disease progression and evaluate medication effects [74]. In the arena of home-based monitoring of chronic diseases, several studies have been carried out, in particular [25; 51]. In [51], the authors proposed a monitoring device capable of providing self-assessments and motor tests for individuals with cognitive impairments. Cued and un-cued tests were designed to collect data on upper limb conditions. The data were then analyzed to evaluate users' compliance and the usability of the device.

Based on similar methods, another home-based assessment tool was presented by Cunningham et al. in [25]. Their novel approach ensured that users did not need to wear any sensorized objects to be monitored. The computer-based device was designed to collect data on hands and fingers movements. Their results showed that the data collected could distinguish between the states where the users' medication was working at its best and when it had worn off.

Monitoring home-based systems can be enhanced with Artificial Intelligence (AI) methods and automatic planning techniques. In the field of assistive and rehabilitation technology, several systems have been designed to increase independent completion of ADL by users with cognitive deficits. In such a case, these systems do not only collect data on users' behavior but can also provide reminders or guidance in order to help them during ADL. They aim to output recommendations in order to help the user complete a sequence of actions that will lead to task completion. The key challenges related to this field were described by Kautz et al. in [55]. In this technical report, the authors highlight the main features an assistive system should implement in order to properly guide users with cognition deficits during ADL. Among the different prototypes the authors designed, they focused on the development of an *ADL prompter* and *ADL monitor*. Taking into account the fact that some users may have difficulties performing ADL by themselves, the aim of the *ADL prompter* was to help such users complete multi-step tasks such as cooking, by providing them with appropriate prompts and guidance.

The *ADL monitor* was designed to track users' performance during different ADL (for example grooming and socializing), and detect users' errors and abnormal patterns of behaviors during the tasks. Extensive literature reviews focusing on assistive technology for cognition have been published and updated through time [29; 64]. In 2011, Gillespie et al. identified 89 publications, where 91 studies of an assistive system were reported [29]. Given the proliferation of such devices, they highlighted the impracticality to conduct large-scale efficacy studies for each of them. They also found that 63% of the reviewed studies focused on assistive systems designed to provide reminding and prompting interventions to users. This specific interest supports Hart et al. study's results, which drew attention to the fact that clinicians saw more potential for devices in the areas of learning/memory, planning/organization and initiation [32].

The increased interest in this area led to the development of complex systems such as Autominder [84; 94] and COACH [14]. Implemented as a scheduling aid, the Autominder reminds older adults what activities should be done through the day, by providing personalized prompts. To achieve its goal, the system tries to maintain a correct representation of the user's daily plan, monitor its execution, and plan reminders accordingly. After being specified, the user's plan is updated trough the day. High quality reminders are then generated by an intelligent planning system [3]. However, compared to the COACH system [14], the Autominder does not help users to correctly go through any of the monitored activities.
The COACH system was designed to provide instructional cueing in order to guide users with dementia during one specific task: hand-washing. In [14], the focus is put on the COACH Markov Decision Process (MDP) based planning system. The latter is in charge of providing cues corresponding to the user's needs during hand-washing. In order to do so, the whole system is composed of a monitoring module (i.e., a vision-based agent) [70], which is used to monitor the user during the task, and provide meaningful information about the user's environment to the planning system. An efficacy study based on participants' comments was then run to evaluate their point of view about the system's performance.

A new COACH system was designed few years later, and implemented a Partially Observable Markov Decision Process (POMDP) based planning system [36; 71]. Contrary to an MDP, a POMDP can model the uncertainties related to the environment of the user (for example the actions done by the user during the task, what he or she has achieved so far), and potentially cope with the errors the monitoring module can make while observing the user's behavior. In [36], an extended description of the main modules composing COACH is given. We learn that the whole system is composed of a video-based hand-and-towel tracker (i.e., a monitoring module), a POMDP-based planning system, and a cue selector. Similarly to the previous prototype, the COACH video-based tracker allows the system to monitor a user during ADL in a non-invasive way. Three experiments were run in order to evaluate the performance of COACH's components, and the overall ability of the system to provide assistance to six users suffering from dementia. Sequences from a user trial were used to evaluate the tracker, and results showed that the latter is robust and can recover from failures. The POMDP-based planning system was evaluated via simulations of hand-washing, and compared with heuristic policies and the MDP. Results showed that the POMDP-based planning system performed best, but not significantly better than the heuristic policies. The global evaluation of the system was run during an eight week user trial, and during these trials, a caregiver or technician could intervene and input information to the system. The results obtained indicated that the POMDP model is a fairly good model for the domain.

Using the same system, the authors of COACH ran another efficacy study described in [71]. In this study, six older adults with moderate dementia were asked to wash their hands while being assisted by the COACH system. The performance of the system was calculated based on its ability to provide the right prompt when necessary (i.e., when the user made mistakes during the task). The results showed that 78% of the COACH system's strategies were correct. In this study, the authors also analyzed the user's performance, and found that when COACH was used, an average of 11% more steps were performed independently by users during the task, which required 60% fewer interactions with a human caregiver.

More recently, Peters et al. [79; 80] developed the TEBRA system, which has been designed to support mildly impaired people during teeth-brushing. Similarly to COACH, TEBRA is composed of a sensing module dedicated to the monitoring of users, a behavior recognition module and a planning system, which provides prompts when necessary. In order to evaluate the system's influence on users' behavior, a user study was performed with seven participants suffering from cognitive deficits. The participants were assessed through two scenarios referred to as the *caregiver* (CG) scenario, and the *system* (SYS) scenario. In the CG scenario, users were asked to brush their teeth as usual. As the toothbrush is sensorized, the TEBRA system could record sensor data and monitor the task progress. In that specific scenario, no system prompts were allowed to be delivered to the users. In the SYS scenario, the system was configured to provide prompts and guide users during the task. The authors highlighted that in the SYS scenario, a caregiver could take over and intervene if the system made fatal errors. The results showed that users made significantly more independent steps in the SYS scenario compared to the CG scenario. In other words, most users gradually showed signs of independent behaviors when they could have access to the system's prompts. In order to evaluate the appropriateness of these prompts, the authors introduced the notion of prompts' *semantic correctness*. Prompts are considered semantically correct when they are delivered at the right time (for example when the user really needs assistance or makes a mistake), and when the information provided by the prompt corresponds to the next step the user should follow to correctly continue or finish the task. Taking into account the trials performed by six participants in the SYS scenario, results showed that 71.3% of the prompts delivered by the TEBRA system were semantically correct. These results are encouraging and show the potential of the TEBRA system. However, as highlighted by the authors, the number of participants involved during the evaluation *does not allow for hypotheses about the impact of the system for people with specific disabilities in general.*

In the next section, we will highlight, from a general perspective, the common framework assistive and rehabilitation technology such as COACH and TEBRA share with each other, and the specific differences that make them unique.

1.1.3 Assistive Systems' Architecture

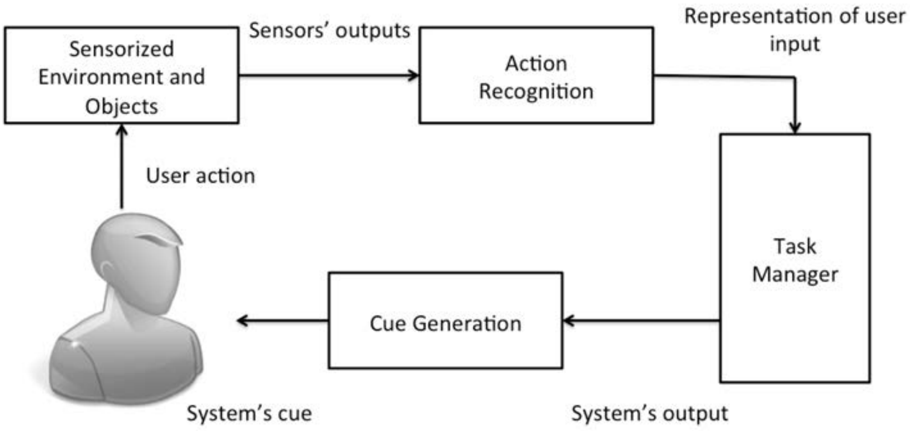


Figure 1.1: Rehabilitation system's architecture

In order to properly assist cognitively impaired users during ADL, systems such as COACH and TEBRA rely on sub-components that work online, in real-time and allow them to:

- **Monitor** users during the task,
- **Detect** and recognize the actions made by users,
- **Plan** the next best actions/strategies/recommendations users should follow in order to properly continue or succeed the task,
- **Display** this information (i.e., what should be done to succeed) in a way users can easily understand.

From a general point of view, as depicted in Figure 1.1, the workflow within such an architecture can be described as follows:

1. The user is given a task to complete,
2. The user moves objects and performs actions related to the given task,
3. Sensors located in the user's environment or smart objects monitor the user's movements and actions,
4. The sensors' outputs are analyzed by an Action Recognition System (ARS) which outputs its interpretation of the user's actions,
5. A Task Manager plans which optimal action the user should do, then passes the information to a cue generation module,
6. The Cue Generation module then selects in which form the Task Manager's output should be delivered to the user.

1.1.3.1 Monitoring module

Assistive and rehabilitation systems designed to remind users what to do during a task, need to be able to monitor the environment of the users and their behavior. The environment corresponds to the direct surrounding or vicinity of the user. For example, if the task to be monitored is cooking, the environment will correspond to the space where the activity is taking place. This space may include the cooking surface, kitchen utensils or any objects within easy reach.
Several researchers have used sensors attached to objects, or placed in the user's environment for task monitoring. In the first version of COACH [69], a tracking bracelet was worn by the user during the hand-washing task, and a digital camera was used to estimate the two-dimensional coordinates of the user's hands [35]. The camera was located above the sink and counter where the task was taking place. The authors of TEBRA also used a camera-based monitoring system [79].

In [79], two cameras were used: one camera observed the environment from an overhead point of view in order to capture the sink and counter; while the second camera was facing the user in order to observe his or her face and upper body during the teeth-brushing task. Contrary to the latest prototype of COACH, the monitoring module of TEBRA is based on the use of smart sensors installed in the user's environment, or integrated in objects that are part of this environment. Indeed, the authors of TEBRA installed a water flow sensor at the water supply (the tap) to determine whether or not the latter was used during the task. Moreover, a sensor module composed of a gyroscope, accelerometer, and magnetometer was attached to a toothbrush, in order to detect its different movements while being held by the user.

Once the monitoring module outputs the information related to the user's environment and behavior, this information can be treated in different ways by the rest of the assistive system. In COACH, the hands' coordinates provided by the monitoring module are directly sent to the module in charge of planning which recommendation the user should follow to succeed the task (i.e., the Task Manager) [71]. Conversely, in TEBRA, the outputs of the monitoring module are analyzed and refined beforehand by an ARS [79] (see Figure 1.1).

1.1.3.2 Action Recognition System

The aim of the ARS is to give a higher level of interpretation of the monitoring module's outputs. To do so, it analyzes the outputs provided by the monitoring module, and infers what actions are being performed by the user during the task. In other words, the ARS uses hands' positions and information about movements of body and objects, to output a more detailed description of the user's behavior. Action recognition is challenging due to the spatial variance in the execution of the task, and the variability with which users can move objects to perform the same actions. In TEBRA [79], action recognition is based on the extraction of a set of features from sensor data. These features are then discretized into a small number of observation variables corresponding to objects' position, movements, and whether the tap is being used or not. In this case, a Bayesian Network is used as a classifier, where the result of the network is a probability distribution over the user's behaviors. From this probability distribution, the most probable behavior is retrieved and sent to the rest of the system. Once the ARS outputs its observation of the user's action, there is no guarantee that this observation corresponds to the real action made by the user. Indeed, the ARS is prone to errors and may send wrong information to the next module. At that point in the architecture of the system, the module which receives the ARS outputs is the Task Manager. It is in charge of planning the best recommendation that the user should follow to successfully continue or complete the task.

1.1.3.3 Task Manager

Different approaches can be implemented as far as the Task Manager (TM) is concerned. The assumption can be made that its inputs directly correspond to the real user's behavior, or that the information it receives may be erroneous. The second case is the most *realistic* one. By *realistic* we mean that there is often a non-zero probability for complex monitoring modules and the ARS, to make errors at one point in their process. Hence, it is the goal of the Task Manager to cope with the potential uncertainty in its inputs and maintain the ability of the whole system to fulfill its overall aim.

Various prototypes of COACH were engineered through time. The earliest prototypes implemented a Markov Decision Process (MDP) based Task Manager [14]. The MDP has been widely used in artificial intelligence (AI) [16; 86] to model and solve decision-planning problems. It provides a model of a system's interaction with its environment, allowing us to find the optimal strategies which can guide that system within its environment. If we take the example of ADL management, the system's environment corresponds to the user and the user's environment that the system monitors. Indeed, this user's environment is modeled by the information captured by the monitoring module and the ARS. Hence, the aim of an MDP-based Task Manager, is to find the optimal strategies that can guide a user within his or her environment.

For example, these strategies can be actions the users should perform at a specific point in the task, or prompts providing a hint in the users' actions and denoting how they should correct their behavior, so in both cases they can properly complete the task. However, the definition of the MDP does not allow it to cope well with uncertainties in its inputs. Indeed, the MDP-based Task Manager will tend to "believe" all information that it receives even if this information is erroneous.

In order to take into account uncertainties related to the user's actions, COACH's planning framework was improved by implementing a Partially Observable Markov Decision Process (POMDP) based Task Manager [36]. Contrary to the MDP, the POMDP can model a decision planning process in the case where the system cannot directly observe the *state* of the user. Here, the *state* of the user can correspond to what he or she has achieved so far (i.e, sequence of his or her actions) as far as the task is concerned, or any relevant information about his or her behavior during the task.

The aim of the Task Manager is not only to find how to properly guide a user during a task. It is also in charge of detecting whether or not the user makes mistakes while going through the task. This ability to detect errors is an important feature which is found in TEBRA and COACH. Indeed, the performance of both systems can be measured by their ability to provide appropriate prompts to users when they need help or have made a mistake. In TEBRA [79], a Finite State Machine (FSM) models timing of a user's behaviors and the different phases in the latter. It then performs a consistency check to verify the user's behavior validity. If the user's behavior is considered to be inconsistent, then the system will provide a prompt. The ability of COACH to properly detect when users make mistakes is also a fundamental element the authors took into account in their efficacy study [71].

Whether an error is detected or not, when the Task Manager outputs a prompt, the latter contains abstract information about what is the next best action to be followed by the user. At this level, the Task Manager's output is in a form of a code. Hence, it is sent to a "cue generation module", which is in charge of conveying assistive information to users in a way that can be easily understood and catch their attention.

1.1.3.4 Cue Generation Module

The implementation of the cue generation module can differ based on the architecture of the system it is part of. For example, in COACH [14], it is a stand-alone component of the system, while in TEBRA, it has been designed as a sub-component of the Task Manager [79; 80].

In both cases, the cue generation module selects the appropriate pictograms, prerecorded videos or sounds to be sent to the users during the task. These elements can be vocal commands, videos showing an actor performing the action the user should mirror to continue the task, a picture giving a hint about the next best step to take, or a written message displayed on a screen stating what should be done. The cue generation module decides the form in which the cue is shared. However, the core of the information about what should be done by the user directly comes from the Task Manager. When the appropriate form of cue is selected, it is displayed via a screen and speakers located where the task is taking place.

1.1.4 Similarities with a Spoken Dialogue System

Now that the main modules composing AI-based assistive systems such as COACH and TEBRA are known; it is interesting to explore their similarities with spoken dialogue systems as far as their architecture is concerned. All systems are intrinsically designed to enable human-machine interaction. However, in a spoken dialogue system, the primary input and output, from and to the user, is speech. Similarly to AI-based assistive or rehabilitation systems, a spoken dialogue system generally consists of three main components, as shown in Fig 1.2. The first component is the speech-understanding module, which transforms the user's speech into text using automatic speech recognition, then converts it into a semantic representation of the user's intention. This information is then passed to the dialogue manager, which selects the most appropriate system output. Its role is similar to the Task Manager in a AI-based assistive system. The last component is the speech generation module, which transforms the dialogue manager's output into text before converting it back to speech [50; 67].

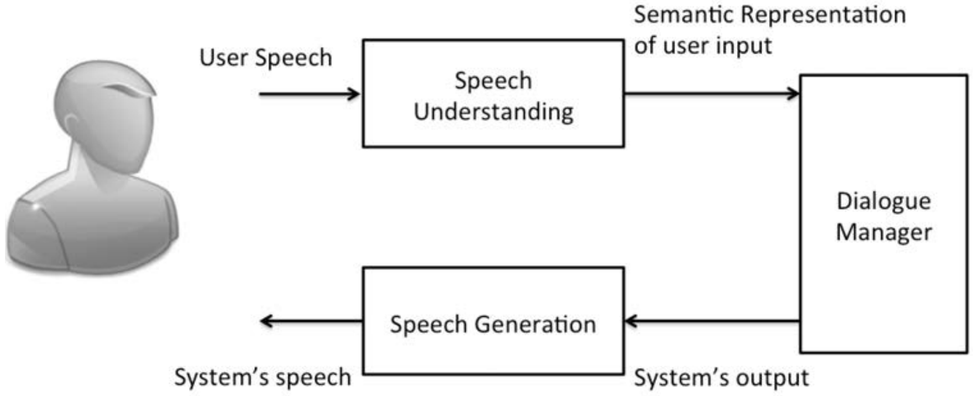


Figure 1.2: Spoken dialogue system architecture

The aim of a task oriented spoken dialogue system is to complete a given task while interacting with a user. In the case of a complex task like booking a flight, the aim of a spoken dialogue system would be to help the user complete the various sub-actions and hence the global task at the end (for example booking the right ticket). To do so, the system tries to engage the user in a spoken dialogue, as a human helper would, retrieving meaningful information which could help the user achieve his or her goal. In such systems, the speech understanding module will make errors, which means that what the user truly says is hidden from the dialogue manager. This draws a parallel with the Action Recognition System and the Task Manager in AI-based assistive systems: the ARS might output wrong information about the user's behavior, so what the user actually does is only partially observable from the Task Manager's point of view. As explained in Section 1.1.3.3, COACH implemented a POMDP-based Task Manager to cope with uncertainty. In the field of spoken dialogue systems, the use of the POMDP is also common. For example, the Hidden Information State (HIS) system [115] is a POMDP-based spoken dialogue system that has successfully been designed to provide tourist information.

Note that the state of the art related to spoken dialogue systems will often be cited in order to highlight its potential benefits to assistive and rehabilitation technologies research.

1.2 Contribution

The focus of this thesis is an assistive and rehabilitation system named CogWatch (CW), and more specifically the research that we have undertaken as far as its Task Manager (TM) is concerned [44; 45; 46; 47; 48; 49]. Taking into account the fact that CogWatch follows the architecture detailed in Figure 1.1, the contributions related to this research are presented in the following subsections.

1.2.1 State representation in the Task Manager

For a successful human-machine interaction, the representation of the user's state in the Task Manager is crucial. In CogWatch, the representation of this state intrinsically depends on the model on which the Task Manager is based.

In the case of the MDP-based system, the user's state is modeled by the Task Manager as a sequence of actions (i.e., actions presumably performed by the user during the task - see Section 1.1.3.3). This simplified model of the user's state corresponds to the Task Manager's state. Thus, each time the ARS outputs an observation of the user's action, this observation is used by the Task Manager to update its current representation of the user's state. In other words, each observation received leads to an update of the Task Manager's state. When the user makes an action, he or she passes from one state to another, and so should the Task Manager. Indeed, one of its goals is to constantly recover a compact model of the user's behavior in order to provide him or her with the right information about what they should do next. Hence, taking into account the fact that a user can go through any sequence of actions during a task, one challenge for the Task Manager is to cope with this huge variance in task completion. In this thesis, we propose a leverage method which allows the Task Manager's state to contain enough information about the user's state, while preventing a drastic expansion of its size. When implemented, this method also allows maintaining the system's state space at a manageable size. In the field of spoken dialogue systems, the state of the art methods also rely on reducing the dialogue state space and performing policy optimization on a reduced space [24].

In the case of the POMDP-based system, the user's state is modeled by the Task Manager as a probability distribution over user's states (i.e., belief state), and each time that the ARS outputs an observation of the user's action, this observation is used to update the belief state. In other words, at any step in the task, the POMDP-based Task Manager takes into account the fact that the ARS may have made a mistake, so it maintains probabilities that the users' action may be different from the one observed by the ARS. This thesis shows how to build the POMDP-based Task Manager's belief state by re-using the representation of the MDP-based Task Manager's reduced state space.

Although this work has been designed to fit CogWatch's specifications, the techniques used are applicable to any POMDP-based assistive systems, which focus on providing guidance during sequential tasks, and have a similar representation of concepts in the user's state.

1.2.2 MDP and POMDP implementation in CogWatch

In this thesis, we will explain how the MDP and POMDP models were defined to take into account the specificities of CogWatch. The impact of these models on the Task Manager will be analyzed and compared within various scenarios. The choice has been made to implement both MDP and POMDP based Task Manager in order to easily run an efficacy study with users. The aims of the evaluations were to measure the ability of the MDP and POMDP based Task Managers to properly guide users during different tasks. This was done in the case where the assumption was made that the ARS was perfect, and also in the case where the uncertainty in the ARS outputs was taken into account. These evaluations were run with real participants, but also via user simulation using a virtualization of the whole system (see Section 1.2.5).

1.2.3 Policy optimization

Policy optimization corresponds to the training phase where the Task Manager "learns" how to optimally act based on its representation of its environment (i.e., the user's environment). Artificial intelligence techniques are known to be promising for improving the performance of spoken dialogue technology [60]. Thus, another contribution of this thesis will be to explain how the POMDP framework designed for statistical spoken dialogue systems can be modified and reused for assistive technology in the case of CogWatch.

1.2.4 Users' error detection

In CogWatch, the Task Manager is not only in charge of planning the next best action the user should follow in order to correctly continue or succeed a given task, it also needs to detect when users make mistakes during this task. We will explain the methods that have been implemented to detect such errors. This has been done in the case where the ARS is considered to be perfect, but also in the case where it is assumed that it may output wrong information. In the first case, the Task Manager uses formal definitions of errors users with AADS can make and systematically verify if they appear in the Task Manager's representation of the user's state during the task. In the second case, an attempt to develop a method for error detection under uncertainty was implemented and will be explained.

1.2.5 User Simulation and virtualization of CogWatch

The development of simulated users based on stroke survivors' behavioral data will be described. Able to display different behaviors, these simulated users were used during evaluation of the MDP-based Task Manager (TM), as well as training and evaluations of the POMDP-based TM. During evaluations, these simulated users were interacting with a virtualization of CogWatch, where the main modules described in Section 1.1.3 were simulated except the Task Manager. In the case of the POMDP-based TM, the choice of using simulated users was justified by the need to understand the effect of the ARS error rates and diverse Nearest Neighbor Search (NNS) techniques on the performance of the TM under uncertainty.

1.2.6 Nearest neighbor search under uncertainty

To plan what should be done by users during a task, the CogWatch Task Manager needs to be trained. During this training phase, it "learns" what recommendation to send to the user, taking into account which state the user may be in during a task. In the case of the CogWatch POMDP-based Task Manager, a grid-based approach is used to solve the POMDP [115], (see Section 2.3.2.2). In other words, the POMDP-based Task Manager does not explicitly associate a recommendation with each belief state contained in the belief state space during training, but does so with a set of belief states only (i.e., belief subspace). During execution or evaluation, when the Task Manager re-uses what it "learned" during training to guide users, it may update a belief state it has never been trained for, and which is not associated to any recommendation to be sent to the user. One solution to tackle this issue is to treat it as a classification problem. Indeed, a classifier can be trained on a set of belief-state/recommendation pairs, which may help it to generalize to the whole belief state space. The particular classifier that we use is based on nearest neighbor search (NNS). Hence, during evaluation, if the Task Manager updates a belief state it has not been trained for, it can find the closest neighbor within the set of belief states for which it has an optimal recommendation associated with. When a neighbor is found, the Task Manager can associate the optimal recommendation associated to this neighbor with the belief state it has just updated. In this thesis, it is shown that the distance function used to identify the element in the belief subspace that is closest to an arbitrary belief state can have a major effect on the Task Manager's performance. A new algorithm, referred to as "SciMK", was designed for CogWatch. Its impact on the system's performance is investigated and compared with other common methods used in the field of NNS (see Chapter 7).

1.3 Thesis structure

Chapter 2 offers an introduction to some decision theory models that could be implemented in an assistive and rehabilitation system. Chapter 3 provides an overview of the system referred to as “CogWatch”, when implemented with the MDP and POMDP based Task Manager. Chapter 4 gives a detailed explanation of the simulated users implemented during training and evaluation of the system. Chapters 5 and 6 discuss policy learning and the challenge related to the implementation of an MDP and POMDP based Task Managers. Chapter 7 discusses the impact of different metrics and classifier on the system’s performance. Finally Chapter 8, presents conclusions and discusses possible future research.

Chapter 2

Decision Theory Models

2.1 Introduction

In the introduction, we described the background related to the difficulties stroke survivors face on a daily basis. We highlighted the need for smart technology that could help them during their activities of daily living (ADL). We then provided an overview of various assistive systems, which have been designed to automatically deliver instructional cues during ADL. We saw that one of the key components of such systems is their planning and decision-making module, referred to as the Task Manager. The latter is in charge of planning the best next actions users should follow in order to correctly continue or complete a given task. Hence, in this chapter, we will focus on different decision theory models, which could be applied to the Task Manager so it can fulfill its goal. For simplification purposes, only topics closely related to the scope of this thesis will be explored.

The first sections of this Chapter give an overview of different cognitive models, which could be used to enable action planning in assistive systems. Some of the limitations and constraints of these methods are highlighted, providing a motive for the implementation of statistical approaches. Section 2.3 focuses on statistical models, providing a detailed description of Markov Decision Process and Partially Observable Markov Decision Process; how they are used to enable decision/action planning, and how they can be applied to task management.

2.2 Cognitive models

As explained in the previous chapter, an ideal assistive/rehabilitation technology can be defined as a system able to deliver appropriate guidance, potentially like a human caregiver would. This raises the question related to the possibility of designing a system able to go through a decision process similar to a human one. In other words, how to make a system select prompts for users like a human would? In this section, we explore the possibility of implementing cognitive decision models, specifically designed to address how people produce sequences of actions during activities of daily living.

2.2.1 Hierarchical approach

Lashley [58] proposed that successful completion of sequential tasks requires activation of a stored action-plan, and that these plans are organized hierarchically. This was supported by the observation that actions performed within a sequence are context dependent. In other words, the actions performed during a task depend on the overall goal pursued, and how it can be reached. If we take the case where the task is to make a cup tea, the right action to perform at each step of the task will depend on the type of tea to make (i.e., overall goal) and what needs to be done to reach this goal (for example adding sugar and milk if the goal is to make a white tea with sugar). Some of these actions can be performed any time during the task, while others need to respect some ordering constraints to be correct. For example, to make a hot cup of tea, the water needs to be boiled before being poured in the cup. Hence, descriptions of the cognitive mechanisms of action selection have often adopted hierarchical structures as we can see in [41]. A representation of such a structure can be seen in Figure 2.1, where a hierarchical approach is applied to describe steps in the tea-making task [39]. In this figure, we see that the task "making a cup of tea" is composed of six basic actions that need to be completed for the cup of tea to be made. Each of these basic actions can be defined by more detailed sub-actions, which will also have to be properly completed for their corresponding higher-level actions to be succeeded.

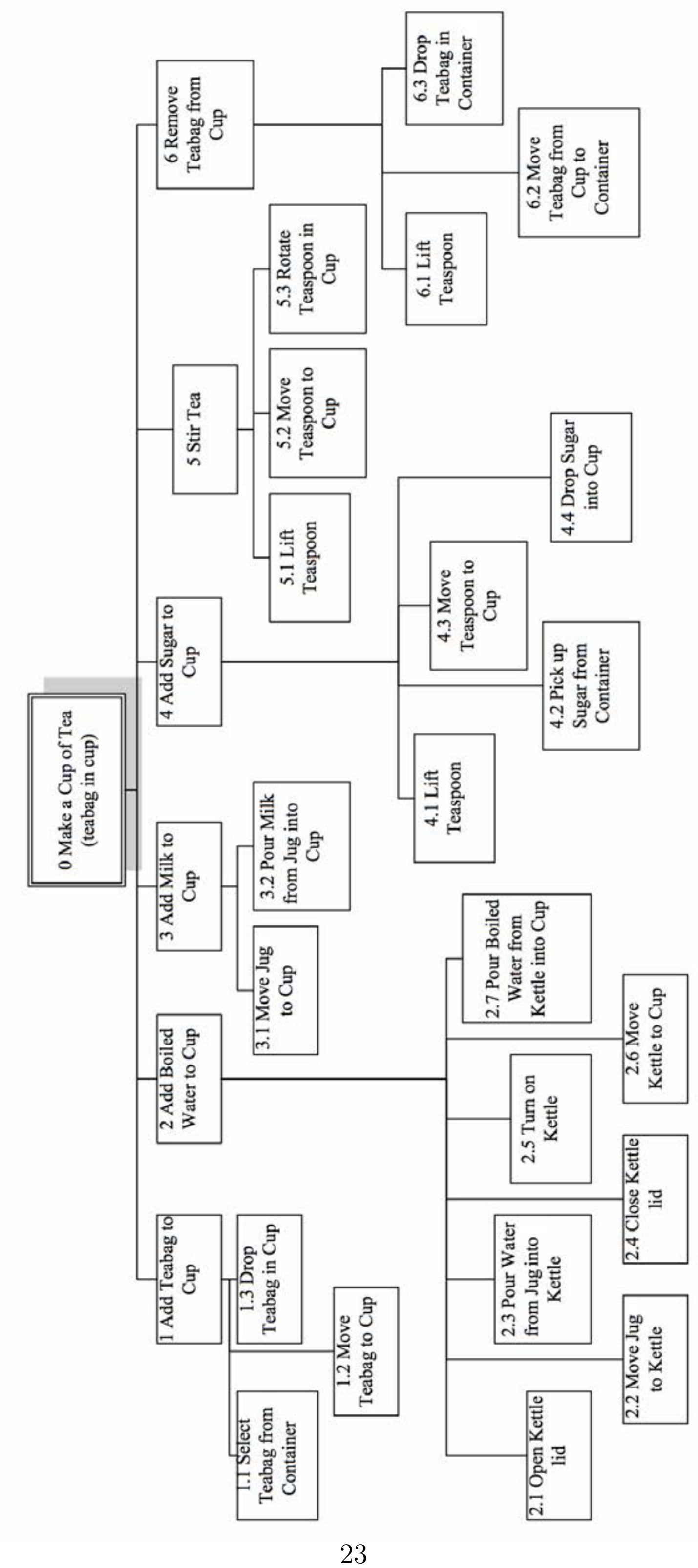


Figure 2.1: Hierarchical representation of a tea making task

Following this lead, new cognitive models were proposed, such as Contention Scheduling by Cooper and Shallice [22; 23], and an implementation of a neural network by Botvinick and Plaut [15]; these are described in the following sections.

2.2.2 Contention Scheduling

Contention scheduling (CS) is an approach designed to model human action selection in routine situations (i.e., situations that are considered to be well known by an individual) [22; 23]. It consists of a hierarchically structured network of well-learned action sequences that are referred to as schemas. In [22; 23], it is argued that when going through a task, an individual can have access to different schemas (i.e., different ways of completing that task). All schemas are assumed to have triggering conditions, activation values and thresholds, which at the end of a selection process allow one schema to be initiated. In other words, schemas compete with each other, and the model proposed by Cooper and Shallice explains how one schema can be selected over the others, so that a sequence of actions is performed by an individual during a task.

Figure 2.2 shows how the CS theory can be computationally implemented. At the center of the figure is the **schema network**, where schemas compete in order to be activated. Schemas receive excitation and inhibition from an **object network** and a **resource network**. The **object network** is another activation network where the nodes correspond to representations of objects an individual may have access to during a task (for example, cup, kettle, spoon in the case of tea-making). The **resource network** is a third activation network, which represents the cognitive resource of the individual. These three networks are connected via feedback loops, so that resource and object nodes excite and are excited by schema nodes. A **selection process** module monitors these interactions and selects a schema when the activation value of the latter reaches a specific threshold. When a schema is selected, its components (i.e., actions) also need to go through a selection process. The chosen action's information is then passed to a **motor system** module, which makes the transition between an action being selected at a cognitive level, and its realization in the physical world.

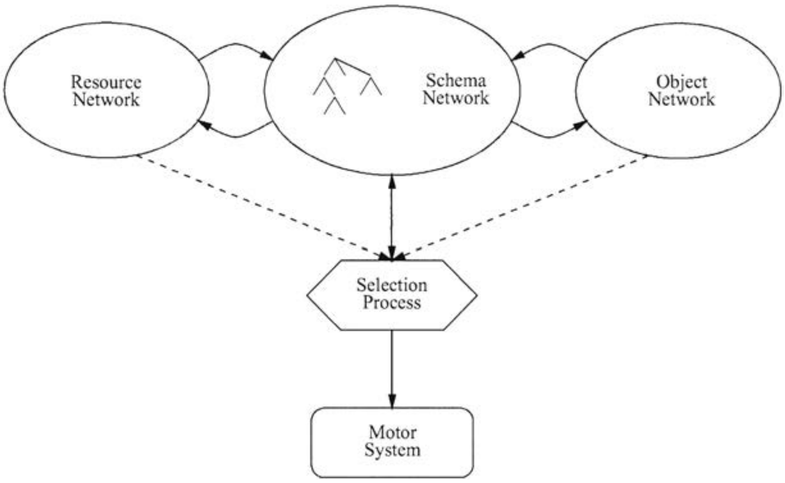


Figure 2.2: Principal components of the contention schedulling [22]

From a computational point of view, the model presented by Cooper and Shallice could be used to design a Task Manager. The latter would then be composed of the **schema network** and the **selection process** module, as seen in Figure 2.2. However, such a Task Manager would not be able to fit the structural specificities of an assistive/rehabilitation system, as described in Figure 1.1, Section 1.1.3. Indeed, a Task Manager following the CS theory would need to communicate in a feedback loop with the ARS, and the latter would need to provide other types of inputs: specific information about the objects (see object network in Figure 2.2), and information about the cognitive resources that the user has access to during the task (see resource network in Figure 2.2). Due to its inputs constraints, the CS theory cannot be applicable to a system like CogWatch (whose Task Manager is the focus of this thesis). CogWatch has a fixed architecture and does not make available to its Task Manager any information about user's resource and object representation. Moreover, it is not clear how CS would cope with uncertainty and deficits within the object and resource networks. In such a case, CS may display an impaired behavior; similar to the one displayed by a person suffering from cognitive deficits. In a system like CogWatch, the Task Manager needs to select appropriate prompts/recommendations for users even if its inputs are erroneous.

2.2.3 Neural Network

In [15], Botvinick and Plaut explain how it is possible to model action selection using a neural network, rather than schema hierarchies. Neural networks are defined as being composed of a set of units, each being related to an activation value [91]. The activation of each unit depends on excitation and inhibition received from other units linked to it, and all units transmit information about their current activation to others through weighted connections. Neural networks are often organized into multiple layers. The first layer is the **input layer**; it is composed of visible units, which transmit features about the environment to the network. The activation of these units propagates through one or more **hidden layers**, which modify the information received from the inputs before sending a pattern of activation to an **output layer**. Similarly to the input layer, the output layer is composed of visible units, which represent the system's response to the input. It is argued that such a network can learn the appropriate set of weights - pattern of activation within its hidden layer - that enables the selection of correct output [91]. Hence, in the case of task management, a neural network could be used to find the appropriate action/recommendation a system should output, taking into account input features related to the user's environment.

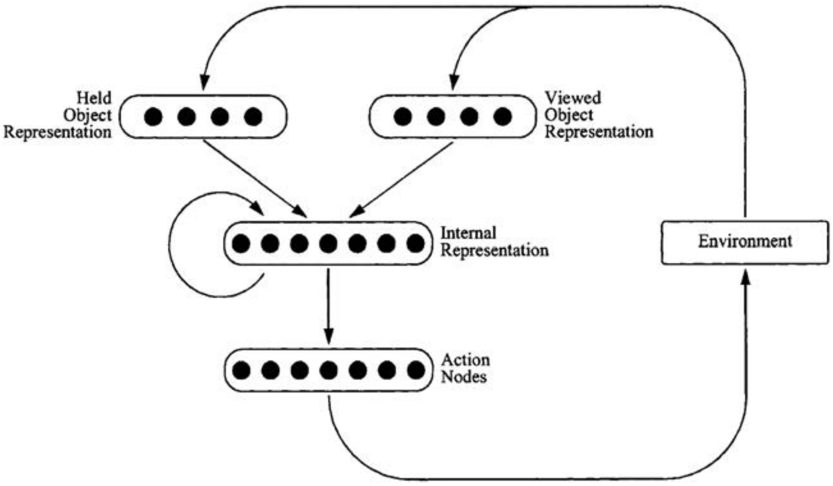


Figure 2.3: Reccurent network architecture for action selection [15; 22]

Figure 2.3 shows the network implemented by Botvinick and Plaut for action selection during ADL. In this network, the input layer carries information about the objects being held by the user (**held object representation**) and those currently in the focus of the user's attention (**viewed object representation**). The output layer relates to the actions that are directed at the objects and environment. From a computational point of view, such a model could also be implemented in a Task Manager. If we take the example of the tea-making task, the network could intensively be trained through supervised learning, with a large amount of labeled examples of users performing the task. In other words, a labeled dataset would be needed. Such a dataset would be composed of feature vectors related to objects being held, and objects being the center of attention of a user during the task. These feature vectors would also need to be associated with the correct next recommendation to output. A Task Manager implementing Botvinick and Plaut's model would then use this dataset to learn, via the network, how specific feature vectors are associated to specific labels (i.e., recommendations). This knowledge would then be reused during execution, where the Task Manager would output the next recommendation to provide to the user for each input it receives. However, similarly to the CS case, a Task Manager based on Botvinick and Plaut's model would not be able to fit the structural specificities of a system like CogWatch. Indeed, a Task Manager following such a model would need to receive inputs (i.e., features about interaction between objects and users) that the ARS does not provide. In CogWatch, the ARS is designed to output observations of actions supposedly made by the users, no information about the objects being the visual focus of users is available.

2.3 Statistical Models

Taking into account the CogWatch architectural specificities, which are similar to spoken dialogue systems (see Section 1.1.4), one possibility to enable decision/action planning is to implement models that have been used in the field of dialogue management, and reapply them to task management; for example, MDP [61], or POMDP [115].

However, it is important to highlight that in the literature [53], both MDPs and POMDPs have initially been defined in the case where the system is a robot navigating in an area trying to reach a goal, not an assistive/rehabilitation system interacting with a user in order to help him or her reach a goal. Thus, we will first go through the definitions of MDPs and POMDPs as they are generally described in the literature, then explain how these frameworks can be implemented in the specific case of assistive/rehabilitation technology.

2.3.1 Markov Decision Process

Suppose that you have a system (for example a robot) located in an area where it can navigate. The system is in a start state and its task is to reach a final state. Each time the system takes an action toward its goal, that action affects the environment and incurs a cost. In order to operate as optimally as possible, the system will have to plan actions that lead to task completion and to the lowest long-term average cost [40]. In the Artificial Intelligent (AI) literature [106], such a system is referred to as an **agent**. When the task that this agent needs to complete is modeled as a Markov Decision Process [86], the framework of the latter is defined as a tuple $\{\mathrm{S, A, P, C}\}$, with:

- **State Space** S:
  The agent's environment is modeled by a set of distinct states S. In general S can be finite, countably infinite or continuous. In this work S is finite and we will write $S = \{s_1, ..., s_N\}$, with $N \in \mathbb{N}$.

- **Action Space** A:
  An agent interacting with its environment seeks to influence the latter by taking actions from its action space A. In this work, A is finite and we will write $A = \{\mu_1, ..., \mu_M\}$, $M \in \mathbb{N}$.

- **Transition model** P:
  The transition model captures the stochastic nature of actions' effects. Thus, $P(s' \mid s, \mu)$ denotes the probability for moving from the current state s to the next s' given that action μ is taken.

- **Cost Function C:**
  The behavior of the agent is encoded in the cost function $C(s, \mu)$. The latter represents the immediate cost for taking action μ when in state s. It is a powerful method the agent uses to model the effectiveness of an action given a specific state.

In the standard implementation of this framework, each time the agent takes an action μ when in a specific state s, the environment stochastically changes according to the transition model $P(s' \mid s, \mu)$ in response to this action. As the agent has access to a finite set of actions A, it uses the cost function $C(s, \mu)$ as a way to judge the quality of its behavior, and make a decision about which action should be taken. In other words, the cost function is composed of the rules governing the agent's environment; it helps the agent to evaluate how good or bad the impact of its actions is when in a specific state.
Previously, we highlighted the potential advantage to have a system that would take into account the long-term impact of its actions. In such a case, the cost function can be formulized in various ways [62]. For example, it can be defined as the expected trial session cost:

$$Cost = \langle \sum_{t=0}^{T_F} C(s_t, \mu_t) \rangle, \tag{2.1}$$

which sums up all the costs experienced by the agent during a trial session (a path in the state space starting in an initial state, and ending in a final state) [61]. In this equation, T_F is the step at which a final state is reached, and $C(s_t, \mu_t)$ is the cost received when action μ_t is taken in state s_t. A variant is the infinite horizon, total discounted cost:

$$Cost = \sum_{t=0}^{\infty} \phi^t C(s_t, \mu_t), \tag{2.2}$$

where ϕ is a geometric discount factor with $\phi \in (0, 1)$. With such a cost function, future costs received at step t are discounted by ϕ^t.

Now that the basics of the MDP framework have been given, and that the way it can be used to model the interaction between an agent and its environment has been described, we need to focus on another of its important concepts: the policy. In decision planning, the problem that an agent faces can be viewed as which actions to take when in a specific state. The choice made by the agent is based on the policy (i.e., the rule that the agent follows in selecting actions, given its state). Intuitively, the policy gives the solution to an MDP; it gives a complete description of what the agent's decision should be for every state $s \in S$. Thus, given an MDP, the task is to find a policy that minimizes the expected sum of costs incurred by the agent's actions.

2.3.1.1 MDP policy optimization

To solve the MDP, two more functions are introduced: the value function $V^{\pi}(s)$ and the Q-function $Q^{\pi}(s, \mu)$, for each state s and action μ [104].

Value Function Suppose that we are given an MDP: a state space S, an action space A, a transition model $P(s' \mid s, \mu)$ and a cost function $C(s, \mu)$. Suppose that we also have a policy π (π: $S \rightarrow A$), such that $\pi(s)$ is the action μ that should be taken by the agent in state s. Concretely, a policy π is a mapping from each state $s \in S$ to action $\mu \in A$. Bellman [7] defines that the value of a state s under policy π, denoted by $V^{\pi}(s)$, is the expected cost when starting at state s and acting according to policy π thereafter. For MDPs, it can be formulized as follows:

$$V^{\pi}(s) = \sum_{s' \in S} P(s' \mid s, \pi(s))[C(s, \pi(s)) + V^{\pi}(s')]. \tag{2.3}$$

One way to visualize this function, is to think ahead from one state to all its possible successors states, as depicted in Figure 2.4. In this diagram, open circles correspond to a state, and solid circles correspond to a state-action pair (for example, $< s, \mu >$). Starting from state s, the agent could take any action from its action space (only three actions are represented). Each of these actions could then be paired with s and correspond to a solid circle. To take an action in a specific state incurs a cost. So if we suppose that when in state s the agent

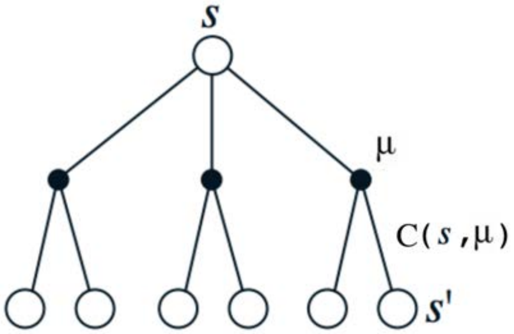


Figure 2.4: Backup diagram for V^{π} [104] - Note that μ could be written $\pi(s)$.

follows the policy π and takes action μ, then an immediate cost is incurred, and the agent moves from state s to potential other next states, such as s'.
In [104], the authors explain that Equation 2.3 *averages over all the possibilities, weighting each by its probability of occurring.* The equation states that the value of state s equals the expected value of the next step s' accumulated along the way, plus the immediate cost (i.e, $C(s, \mu)$ in figure 2.4) [9]. There is at least always one policy π^* that is better or equal than all other policies. This policy is called the optimal policy. The optimal value function based on this optimal policy is given in Bellman's principle of optimality [7]:

$$V^*(s) = \min_{\mu \in A} \sum_{s' \in S} P(s' \mid s, \mu)[C(s, \mu) + V^*(s')], \tag{2.4}$$

The optimal value of s, $V^*(s)$, is the expected cost when starting at state s and acting according to the optimal policy π^*.

Q-function Now, let the Q-value of a state-action pair, denoted by $Q^{\pi}(s, \mu)$, be the expected cost for taking action μ when in state s and acting according to policy π afterward [110]. We see that $Q^{\pi}(s, \mu)$ and $V^{\pi}(s)$ can be defined recursively in terms of each other. This can be formulized as follows:

$$Q^{\pi}(s, \mu) = \sum_{s' \in S} P(s' \mid s, \mu)[C(s, \mu) + V^{\pi}(s')] \tag{2.5}$$

with

$$Q^{\pi}(s, \pi(s)) = V^{\pi}(s) \tag{2.6}$$

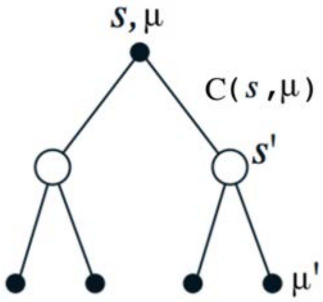


Figure 2.5: Backup diagram for Q^{π} [104] - Note that μ' could be written $\pi(s')$.

This equation states that if the agent is in state s, takes action μ, and moves to state s', it incurs the immediate cost $C(s, \mu)$, plus the expected cost $V^{\pi}(s')$ accumulated along the way. In Figure 2.5, state-action pairs are solid circles and states are open circles. We see that, after taking action μ in state s, the agent could move from s to any potential next state, while incurring the corresponding immediate cost. Suppose that after taking action μ while in state s, the agent incurs a cost (i.e., $C(s, \mu)$ in the Figure 2.5), then reaches the next state s'. From this state, the agent could take any next action contained in its action space (only two actions are represented). Each of these actions could then be paired with s'; the rest of the process following the one explained in Figure 2.4. In order words, following the definition of $Q^{\pi}(s, \mu)$, when the agent reaches state s', it continues by following policy π, and takes action μ', with $\pi(s') = \mu'$.
Thus the optimal Q-function $Q^{*}(s, \mu)$ is defined as the expected cost returned when action μ is taken in state s, and the optimal policy π^{*} is followed afterward:

$$Q^{*}(s, \mu) = \sum_{s' \in S} P(s' \mid s, \mu)[C(s, \mu) + V^{*}(s')]. \tag{2.7}$$

Therefore, the optimal value function can also be expressed by:

$$V^*(s) = \min_{\mu \in A} Q^*(s, \mu). \tag{2.8}$$

The optimal policy π^* can then be derived from the optimal value function with:

$$\pi^*(s) = \arg\min_{\mu \in A} [\sum_{s' \in S} P(s' \mid s, \mu)(C(s, \mu) + V^*(s'))], \tag{2.9}$$

or from the optimal Q-function:

$$\pi^*(s) = \arg\min_{\mu \in A} Q^*(s, \mu). \tag{2.10}$$

2.3.1.2 Value iteration algorithm

Different algorithms can be implemented in order to find the optimal policy π^*. For example, one way to find the optimal policy π^*, is to find the optimal value function V^*. The latter can be determined by an iterative algorithm called *value iteration* defined in [7; 104]. As seen in Algorithm 1, at first the Value iteration algorithm randomly guesses and assigns a value to every state $s \in S$. These values are then iteratively updated using the Bellman Backup operator. By doing so, it creates successively better approximations of the value function (see Equation 2.4) at every time step. The Bellman residual is then calculated as the absolute difference between the previous guess of each value function and the new estimation obtained after update. The algorithm stops when it converges. Convergence is signaled when the Bellman error (i.e., the largest Bellman residual of all states) is less than a pre-defined threshold θ.

In many cases, the exact details of the MDP (i.e., transition model, cost model) are not known, or too complicated to obtain. In such cases, problems are addressed by reinforcement learning algorithms [90], whose main characteristic is the trial-and-error search technique. Indeed, with reinforcement learning, the agent is not told what to do. It first follows an initial policy π, discovers which actions leads to a smaller expected cost by trying many of them, then repeatedly updates the Q value function.

Algorithm 1 Value Iteration

```
$\theta \leftarrow$ arbitrary
for all $s \in S$ do
  $V(s) \leftarrow$ arbitrary
end for
repeat

  for all $s \in S$ do
    $oldV \leftarrow V(s)$
    $V(s) \leftarrow \min_{\mu \in A}[\sum_{s' \in S} P(s' \mid s, \mu)[C(s, \mu) + V(s')]]$
    $Bellman_{residual} \leftarrow |V(s) - oldV|$
    $Bellman_{error} \leftarrow \max(Bellman_{error}, Bellman_{residual})$
  end for
until $Bellman_{error} < \theta$
for all $s \in S$ do
  $\pi^*(s) = \arg\min_{\mu \in A}[\sum_{s' \in S} P(s' \mid s, \mu)(C(s, \mu) + V(s'))]$
end for
```

A common reinforcement learning algorithm is for example the Monte Carlo Algorithm, which was implemented in an MDP-based spoken dialogue system [61]. In this thesis, the choice was made to implement a Monte Carlo algorithm, because it is known to be easy and efficient [104], and also due to the similarities between spoken dialogue systems and assistive systems (see Section 1.1.4).

2.3.1.3 Monte Carlo algorithm

The Monte Carlo (MC) Algorithm is a reinforcement learning algorithm tailored for learning tasks that are episodic [5]. A learning task is the process an agent goes through in order to find the optimal policy. Such a task is defined as episodic when it has a start state and a final one. To find the optimal policy during an episodic learning task, the agent goes through an episode (i.e., a trial) of interactions with its environment until a final state is reached. After reaching this final state, a new episode starts again, where the same problem as before needs to be solved (i.e., finding the optimal policy). During every new episode, the agent improves its behavior until the optimal one is reached. This process is explained in Algorithm 2.

Algorithm 2 Monte Carlo algorithm

```
Inputs:
$A$: set of machine's actions $\mu$
$S$: set of machine's states $s$
$k \leftarrow 0$, with $k$ number of iterations
for every $s \in S$ and $\mu \in A$ do
  $N(s, \mu) \leftarrow 0$, with $N(s, \mu)$ number of times $\mu$ is selected when in $s$
  $Q_k^*(s, \mu) \leftarrow$ random guess of expected cost for selecting $\mu$ when in $s$
  $\pi_k^*(s) \leftarrow \arg\min_\mu Q_k^*(s, \mu)$
end for
repeat
  $k \leftarrow k + 1$
  for every $s \in S$ and $\mu \in A$ do
    Generate an episode starting at $s$ with action $\mu$ and proceeding according
    to $\pi_{k-1}^*(s)$ until a final state is reached.
    for each pair $\langle s', \mu' \rangle_i$ recorded during the episode do
      Calculate $F(s', \mu')$, which is the sum of all the immediate costs incurred
      after $\langle s', \mu' \rangle_i$ until the final state.
      Update:
      $N(s', \mu') \leftarrow N(s', \mu') + 1$
      $Q_k^*(s', \mu') \leftarrow \frac{Q_{k-1}^*(s', \mu') * N(s', \mu') + F(s', \mu')}{N(s', \mu')}$
      $\pi_k^*(s) \leftarrow \arg\min_\mu Q_k^*(s, \mu)$
    end for
  end for
until converged
```

In this algorithm, the process starts by assigning random costs to each state-action pair $\langle s, \mu \rangle$, with $s \in S, \mu \in A$. This first guess allows the initialization of the policy, following Equation 2.10. The algorithm then proceeds as follows. Each time there is a new iteration, the agent goes through an episode by starting at a state s, takes action μ, then follows the current policy until a final state is reached. Suppose that it is the first iteration, with $k = 1$. The agent is in a state s, takes action μ, then follows policy π_0^* until a final state is reached. During each episode, the states the agent visits and actions it takes are recorded. So when the final state is reached, the expected cost is calculated for taking an action μ in a state s for each state-action pair $\langle s, \mu \rangle$. This information is then used to update the Q value and the policy after each episode.

Every new iteration, the agent will follow the policy that was updated during the previous iteration. This is repeated until convergence is reached. Here, we could define convergence as the point where the Q values remain stable, and the optimal action for each state remain the same after a large number of iterations.

2.3.1.4 MDP-based task management

In the two previous sections, we gave an overview of how an agent can learn how to act optimally in its environment. In the literature, this framework is generally defined to fit the idea that the agent is a robot. In the case of systems such as CogWatch, the agent would correspond to the Task Manager. Indeed, in order to guide users during a task, the Task Manager is the element of the system that needs to learn how to take decisions (i.e., decide which recommendations to send to the user) in an optimal way. As explained in Section 1.1.3.3, to provide such assistance, each time the user performs an action, the Task Manager updates its state, which is a simplified model of the user's state. After each update, the Task Manager plans an action/recommendation it "believes" the user should follow, and incurs a cost for doing so. Hence, the overall aim of the Task Manager could be defined as choosing the recommendations that lead to task completion and to the lowest long-term average cost [40] from the user point of view. For example, the goal of the CogWatch Task Manager is to provide recommendations that minimize the cost of completing the tea-making task.

The task can be modeled with an MDP, and the optimal policy can be found by applying a Monte Carlo Algorithm 2.3.1.2. The framework given in Section 2.3.1 can also be reapplied. However, more thoughts need to be given to the definition of the Task Manager's state. Indeed, the latter needs to take into account the fact that the user is part of the system's environment, and that this user goes through his or her own decision process. This section will then be dedicated to the definition of the user and Task Manager's state, the notion of user and Task Manager's action and probability distribution in the context of assistive/rehabilitation systems. In section 2.2, we discussed different cognitive decision models, such as Cooper and Shallice's Contention Scheduling model [22; 23]. Due to CogWatch's architecture constraints, we explained that it is not currently possible to implement such a model in the Task Manager. However, the Cooper and Shallice's method could be used to model a user's behavior during ADL. In this case, the main features composing the user's model could be defined as being composed of the user's resources, the objects' network and schemas network - see Section 2.2. In other words, the user state could be defined so that it depends on parameters such as:

- the objects the user has access to during the task,
- the user's knowledge of how these objects can be used,
- the user's cognitive and physical resources: his or her understanding of the task and how it can be performed, the actions he or she can take, and the history of actions that have already been taken during the task.

Ideally, the Task Manager (TM) would have access to the full user state, and would use the latter to decide what is the best next recommendation μ^* to provide (i.e., action it believes the user should follow). However, in CogWatch, the TM only receives observations about the actions a_u that the user has presumably performed. This information would then correspond to the features part of the schemas network, and no other network defined in Cooper and Shallice's method. In other words, the full user state is hidden, and the TM tries to infer a simplified user's state representation from the information that it receives from the Action Recognition System. Since the information received by the TM correspond to actions presumably performed by the user, it is assumed that its simplified user's state representation (i.e., the limited part of the full user state that the TM can recover) is the sequence s of actions that the user has presumably performed. Thus, this is this sequence s of actions that the TM relies on in order to provide assistance during a task. When an MDP is used to model such a task, this is the set of states s that corresponds to the Task Manager MDP state space S. Similarly to the schemas, each of these states can be interpreted as a sequence of actions the user could go through during a task. In such a case, the Task Manager state s can be defined as what it considers the user has achieved so far. Hence, a state s depends at least on the action a_u taken by the user, and the history of actions s_d the user has already taken.

Now that we have highlighted the difference between the user state and the Task Manager state, we need to explain the difference between the user actions and the Task Manager's actions, which are referred to as recommendations.

Each time the Task Manager receives an observation o as a consequence of the action a_u made by the user, it outputs a prompt ω. This prompt ω is at least composed of the recommendation μ that should guide the user during the task. Contrary to a robot which takes actions to move itself from a start state to reach a goal, the Task Manager's recommendations μ do not directly influence the environment during execution. Indeed, during execution, it is the user, through his or her own decision process, who modifies the environment with his or her own actions a_u. Thus, the output of the Task Manager only has a possibility to influence the user's decision process. Thus, at any step, the user may decide to proceed by following μ or taking any other action to modify the environment accordingly.

In such conditions, what makes the Task Manager "move" from one state to another essentially depends on the actions made by the user, which are observed by the Task Manager via the ARS outputs. This means that there is a need to modify the MDP probability transition $P(s' \mid s, \mu)$ for CogWatch. In the literature (see Section 2.3.1), $P(s' \mid s, \mu)$ corresponds to the probability for the agent to move from state s to s' after having taken μ. Previously we said that s depends on a_u and s_d, so the probability transition could be written:

$$\begin{aligned} P(s' \mid s, \mu) &= P(a'_u, s'_d \mid \mu, a_u, s_d) \\ &= P(a'_u \mid \mu, a_u, s_d, s'_d) P(s'_d \mid \mu, a_u, s_d, a'_u) \\ &\approx P(a'_u \mid \mu) P(s'_d \mid s_d, a'_u) \end{aligned} \tag{2.11}$$

where $P(a'_u \mid \mu)$ is the probability that the user takes action a'_u when the Task Manager outputs μ, and $P(s'_d \mid s_d, a'_u)$ is the probability for the user to move from one history of actions to another, when taking a new action a'_u. If we consider that the user always follow the Task Manager's recommendation then $P(a'_u \mid \mu) = 1$, and $P(s' \mid s, \mu) \approx P(s'_d \mid s_d, a'_u)$. Note that this definition of state s, where s may depend on a_u and s_d is inspired by the work of Williams et al. described in [114].

To understand what values $P(s' \mid s, \mu)$ can take, one needs to recall that each time the user takes an action a_u, the Task Manager receives an observation o related to a_u. In CogWatch, the state of the MDP based Task Manager is defined as a sequence of observations, where these observations are actions the Task Manager believes the user has just made. Indeed, MDPs model full observable domains, where the agent has a complete knowledge of the current world state [92]. So when a new observation o is received, the Task Manager moves from s to s', where s' corresponds to the previous state s incremented by the new ARS observation o'. For example, if $s = (a_1, a_2, a_3)$, $\mu = a_{10}$ and the new observation received is $o = a_{10}$, then $P(s' \mid s, \mu) = 1$ for $s' = (a_1, a_2, a_3, a_{10})$ and $P(s' \mid s, \mu) = 0$ for any other state. In other words, there is a probability 1 to move from $s = (a_1, a_2, a_3)$ to $s' = (a_1, a_2, a_3, a_{10})$ when the observation received is a_{10}.

As one can see, one of the intrinsic limitations of this framework is related to the fact that the MDP-based Task Manager tries to recover what the user has done during the task, "believing" the ARS observations without considering that they might not correspond to the real actions made by the user. In a realistic setting, the ARS may make mistakes due to inter- and intra-subject variability in the way that actions are performed, and also because of sensors noise. Since the MDP-based Task Manager assumes that the environment is fully observable, it cannot function well in noisy conditions. Indeed, the MDP-based Task Manager does not keep track of alternative states the user might be in during the task. For it to have a recovery mechanism that will handle uncertainty in its inputs, it will have to learn how to behave in an environment that is only partially observable. In such a case, the natural representation of the user's state from the Task Manager point of view is a probability distribution over all simplified user states, referred to as a belief state b. The latter denotes the fact that due to uncertainty, the system considers that the user may be in different states at the same time.

For the Task Manager to still be able to provide appropriate prompts to users during a task, and to do so under uncertainty, the task could be modeled as a POMDP. The latter can provide a framework to model the uncertainty in the Task Manager's input, allowing the latter to cope with noisy representation of its environment. Therefore, the POMDP offers the possibility to improve the robustness of an assistive/rehabilitation system focusing on task management when deployed in an environment where the states are not fully observable.

2.3.2 Partially Observable Markov Decision Process

Formally, a POMDP is a tuple $\{S, A, P, C, O, Z\}$ [52], where the tuple $\{S, A, P, C\}$ corresponds to an MDP as defined in Section 2.3.1. Two more parameters are introduced and are directly related to the POMDP:

- **Observation space O**
  An observation is an output from the ARS. Thus, the observation space is the set of observations o ($o \in O$) the ARS can make and that the Task Manager can receive. After each action a_u executed by the user, the ARS will output an observation o corresponding to its interpretation of the action that the user has just completed.

- **Observation probability Z**
  Observations are probabilistically related to states. Z is a set of observation probabilities, where $Z_{o',s',\mu} = P(o' \mid s', \mu)$, which denotes the probability that the observation o' is made, when the Task Manager is in an unobservable state s' and the recommendation μ is outputted.

When a task is modeled with a POMDP, it operates as follows. At each step, the user and the Task Manager are in an unobservable state. The Task Manager's unobservable state is s, where s is a simplified model of the user's state, and $s \in S$. As s is unobservable, the Task Manager maintains a probability distribution over states, called the belief state b, where $b(s)$ is the probability to be in state s. Taking into account the current belief state b, the Task Manager selects a recommendation $\mu \in A$. This is the action it believes the user should follow and which minimizes the expected cost.

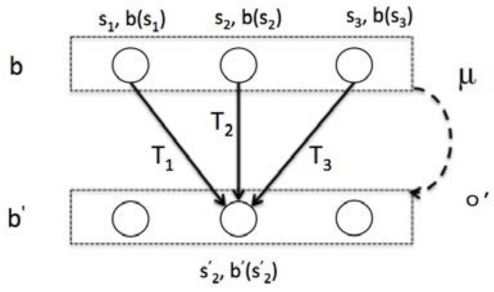


Figure 2.6: Partial backup diagram of the belief state update

Whichever action a_u the user executes causes sensor data to be passed to the ARS, which outputs a new observation $o' \in O$, which depends on the new unobserved state s' and μ [115]. As explained in [52], every time an observation is received, the Task Manager's belief state is updated by Bayes' rule:

$$b'(s') = k \cdot P(o' \mid s', \mu) \sum_{s \in S} P(s' \mid s, \mu) b(s) \ , \tag{2.12}$$

where

- $k = 1/P(o \mid \mu, b)$ is a normalization constant, which causes b' to sum to 1,
- $P(o' \mid s', \mu)$ is the observation probability,
- $\sum_{s \in S} P(s' \mid \mu, s) b(s)$ is the probability to move from state s to s' when selecting μ, considering the probability to be in s; summed over all the states $s \in S$.

For example, suppose like in Figure 2.6 that b is a probability distribution over 3 states: s_1, s_2, s_3, and $b(s_1), b(s_2), b(s_3)$ are the respective probabilities to be in these states. A recommendation μ is selected for the user, which ultimately leads to a new observation o' to be received. Then, for example, the new degree of belief in state s_2' is:

$$b'(s_2') = k \cdot P(o' \mid s_2', \mu)[T_1 b(s_1) + T_2 b(s_2) + T_3 b(s_3)] \tag{2.13}$$

with $T_1 = P(s_2' \mid s_1, \mu)$, $T_2 = P(s_2' \mid s_2, \mu)$ and $T_3 = P(s_2' \mid s_3, \mu)$.

Let b_t be the current belief state at step t, and $b_t(s)$ the probability of being in s at t. After each update, the Task Manager receives an immediate cost $C(b_t, \mu_t) = \sum_{s \in S} b_t(s) C(s, \mu_t)$, depending on its current belief state b_t and μ_t. The goal of the POMDP-based Task Manager is to select recommendations that minimize the long-term cost, for example - the cumulative, infinite horizon, discounted cost:

$$Cost = \sum_{t=0}^{\infty} \phi^t C(b_t, \mu_t) = \sum_{t=0}^{\infty} \phi^t \sum_{s \in S} b_t(s) C(s, \mu_t), \tag{2.14}$$

where ϕ is a geometric discount factor, and future costs received at step t are discounted by ϕ^t.

2.3.2.1 Exact POMDP optimization

Similarly to an MDP, the problem that a POMDP-based agent faces can be described as what optimal recommendation to select when in a belief state. To solve a POMDP means to find a policy π, which is a mapping between belief states and recommendations; $\pi : B \to A$. As belief states are probability distributions over states, the belief state space is infinite, which makes policy optimization in a POMDP model challenging. Similarly to the Bellman equation for MDPs 2.4, a policy π can be defined by a value function V^π; where V^π(b) is the expected cost when starting in belief state b and acting according to policy π thereafter [52]:

$$V^\pi(b) = \sum_{s \in S} b(s) V^\pi(s) \tag{2.15}$$

In Equation 2.15, $V^\pi(s)$ is the expected cost when starting in state s and acting according to policy π thereafter, as seen in Equation 2.3. For problems with a finite planning horizon, the value function is piecewise linear and convex [52; 102]. The planning horizon is how far the agent “looks into the future” when deciding what to do. A finite horizon planner is an agent that looks for a fixed finite number of steps ahead. To say that the value function is piecewise linear in such a case means that it can be represented by a set of alpha vectors Γ, where each alpha vector $\alpha_i \in \Gamma$ is of size $|S|$, and is associated to a recommendation $\mu(i) \in A$. Thus, Equation 2.15 can be written more compactly. If we let $\alpha_\pi = \langle V^\pi(s_1), V^\pi(s_2), ..., V^\pi(s_n) \rangle$ then $V^\pi(b) = b \cdot \alpha_\pi$.

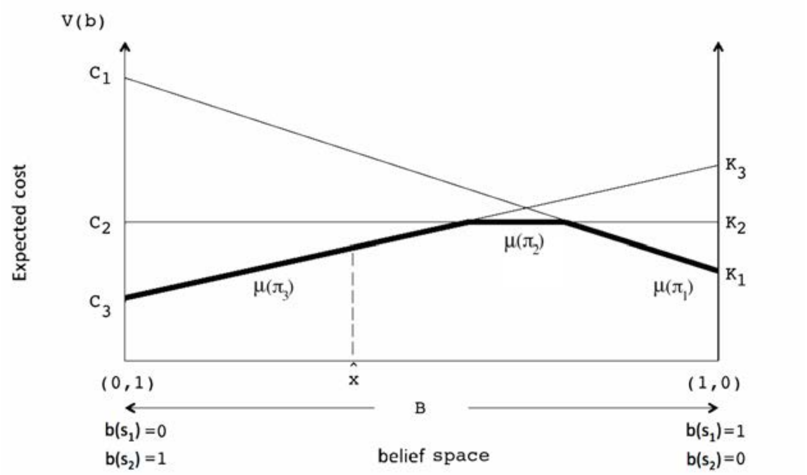


Figure 2.7: POMDP exact value function representation

Given a full set of alpha vectors, the optimal value function and corresponding optimal policy can be written:

$$V^*(b) = \min_{\pi^*} b \cdot \alpha_{\pi^*} \tag{2.16}$$

$$\pi^*(b) = \mu(\arg\min_{\pi^*} b \cdot \alpha_{\pi^*}) \tag{2.17}$$

where $(\cdot)$ is the dot product. An example of a value function for a two state (i.e., s_1 and s_2) POMDP is given Figure in 2.7. In this example, the belief state space B is depicted on the x-axis, the y-axis represents the expected cost for one step, and the value function itself is depicted by the lower line in bold. Suppose that when in belief state $b = (0, 1)$ the agent can act following three different policies (π_1, π_2, π_3). Thus, when starting at $b = (0, 1)$, then following each policy, the agent will incur three specific expected costs, each of them depicted in Figure 2.7, by C_1, C_2, C_3. Similarly, three expected costs K_1, K_2, K_3 will be incurred when starting in $b = (1, 0)$ and following π_1, π_2, π_3 afterward. Hence, the belief state space is divided into three regions, and the optimal recommendation the agent should take in each region is the one associated with the undermost vector in that region.

For example, if a belief state is defined such as it is inferior to belief state x, then the optimal recommendation the agent should take when in this belief state is $\mu(\pi_3)$. Similarly to an MDP, the exact optimal value function can be found by implementing a value iteration algorithm [102]. However, in practice, computing optimal planning solutions over all possible belief states in B for POMDPs is often intractable [65]. Indeed, one well known reason for the limited scalability of POMDP value iteration algorithms is the so-called curse of dimensionality [52]: in a problem with n states, POMDP planners must reason about belief states in an $(n-1)$-dimensional continuous space [83]. Hence, a lot of efforts have been made by researchers to develop and implement approximate solutions.

2.3.2.2 Approximate POMDP optimization

To cope with the computational complexity related to the exact value iteration, a family of approximate methods exists. These methods operate on a fixed set of belief states (i.e., belief subspace) rather than the full belief state space. Indeed, many belief states will never be encountered during execution by an agent, and may be unnecessary to consider during training.

Pineau's Point-based value iteration (PBVI) [83] is one of those approximation methods. This algorithm begins by generating a set $\tilde{B}$ of belief states that are likely to be reached. It initializes an alpha vector for each belief state, then repeatedly updates the value of that alpha vector. Similarly to the exact value iteration, PBVI produces a set of alpha vectors and their corresponding machine's actions. The algorithm continues by extending the set $\tilde{B}$ of belief states and finding a new solution for the extended set. The actions found for each belief state in the set are then guaranteed to be optimal for these belief states only. The hope is that they will also be optimal for nearby belief states. Another group of approximation methods consist of grid-based approximation algorithms [17]. When implementing such algorithms, the belief state space is discretized, and each belief point is associated with the value function at that point plus the corresponding optimal machine's action to take. Contrary to point-based algorithms, which maintain a full α-vector for each belief point, the grid-based approach only updates the value of each belief point until the optimal policy is found.

In [83], Pineau et al. compared the PBVI's performance with Brafman's grid-based algorithm and other methods, and results showed that PBVI achieved competitive performance. Indeed, in one example, the implementation of Brafman's grid-based algorithm and PBVI led to the same goal completion rate, which is one of the most important parameters to take into account in the context of assistive and rehabilitation systems. In this thesis, we implemented a grid-based approach, as proposed by Young et al. in [115]. With this approach inspired by Brafman's method [17], when the system needs to find the optimal machine's action for an arbitrary belief point, the nearest belief point is found and its action is used. Young et al. acknowledged the fact that other methods may be implemented in the case of large state space. However, their method has proved to work well in the HIS system, due to a reduction technique they implemented in order to map the full belief space into a summary space [115]. The HIS system, which is a spoken dialogue system, has a similar structure than CogWatch, as explained in Chapter 1. When implementing Young's method and applying a specific reduction state space technique taking into account CogWatch's specificities (see Chapter 5), we also succeeded to obtain satisfying results.
In the next paragraph, we explain how the optimal policy can be found when following a grid-based approximation approach. Similarly to the MDP 2.3.1.2, this can be done with reinforcement learning algorithms such as the Monte Carlo (MC) Algorithm [104; 115].

Monte Carlo Algorithm: When using a grid-based approach, the POMDP policy is represented as a grid where each belief point is related to a Q^{π} function, and $Q^{\pi}(b, \mu)$ is the expected cost for taking μ in b then following the policy π afterward. To find the optimal policy π^{*}, various episodes/trials need to be generated (see Section 2.3.1.3) in order to update the Q values for each $b \in B$ and $\mu \in A$. Since many trials are necessary to robustly estimate the Q values, it is common practice that a simulated user is used to generate episodes and interact with the system during policy optimization [27; 61; 87; 89; 96; 99; 105]. Hence, we will explain in detail the MC algorithm by considering that a simulated user (SimU) is used during training.

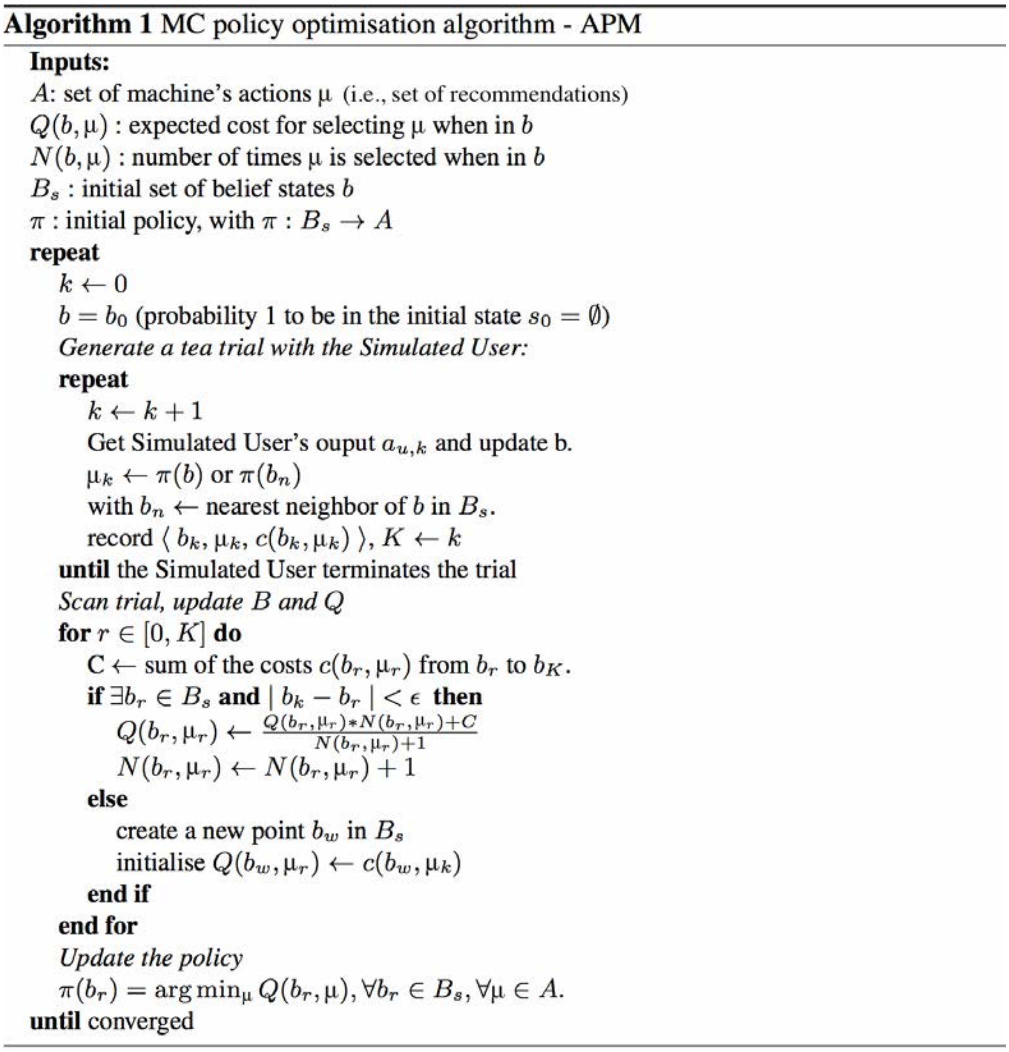
As one can see in Figure 2.8, at the beginning, the belief subspace B is initialized with a set of belief states [17], the expected costs $Q(b, \mu)$ for every $b \in B$ and $\mu \in A$ are guessed, and the initial policy π is deduced from these guesses. Before each episode (i.e., tea trial in the case of CogWatch) the *SimU* has no action contained in its history; its state s_0 is empty, and the Task Manager is at b_0, with $b_0 = b(s_0) = 1$. At each step of the process, the *SimU* makes an action a_u which generates an ARS observation o, leading the Task Manager to update its current belief state following Equation 2.12. At this point, the process follows a specific pattern:

- If the updated belief state is already contained in the belief subspace, the Task Manager selects its corresponding optimal recommendation μ^*.
- Conversely, if the newly updated belief state b is not part of the belief subspace, the distance between b and each belief state currently contained in the belief subspace is calculated:
- If the distance between b and its closest neighbor exceeds a chosen threshold (ϵ in Figures 2.8), then b is added to the belief subspace.
- If the distance between b and its nearest neighbor does not exceed ϵ, then b is not added to B and is replaced by its nearest neighbor in the training process. At the end of each episode, a sequence of belief points-recommendation pairs $\langle\ b, \mu\ \rangle$ is recorded, and used to update the estimate $Q(b, \mu)$.

At the end of the training, when the Q values are estimated, the optimal policy π^* is obtained with:

$$\pi^*(b) = \arg\min_{\mu} Q(b, \mu), \forall \mu \in A, \forall b \in B. \tag{2.18}$$

The inventory of belief points added to B grows over time until a predefined maximum number of belief points allowed is reached. Similarly to the MC algorithm used for MDPs (see Section 2.3.1.3), this process is repeated until convergence is reached. More information about user simulation will be given in Chapter 4.



Algorithm 1 MC policy optimisation algorithm - APM

```
Inputs:
$A$: set of machine's actions $\mu$ (i.e., set of recommendations)
$Q(b, \mu)$ : expected cost for selecting $\mu$ when in $b$
$N(b, \mu)$ : number of times $\mu$ is selected when in $b$
$B_s$ : initial set of belief states $b$
$\pi$ : initial policy, with $\pi : B_s \rightarrow A$
repeat
   $k \leftarrow 0$
   $b = b_0$ (probability 1 to be in the initial state $s_0 = \emptyset$)
   Generate a tea trial with the Simulated User:
   repeat
      $k \leftarrow k + 1$
      Get Simulated User's ouput $a_{u,k}$ and update b.
      $\mu_k \leftarrow \pi(b)$ or $\pi(b_n)$
      with $b_n \leftarrow$ nearest neighbor of $b$ in $B_s$.
      record $\langle\ b_k, \mu_k, c(b_k, \mu_k)\ \rangle$, $K \leftarrow k$
   until the Simulated User terminates the trial
   Scan trial, update $B$ and $Q$
   for $r \in [0, K]$ do
      C $\leftarrow$ sum of the costs $c(b_r, \mu_r)$ from $b_r$ to $b_K$.
      if $\exists b_r \in B_s$ and $\mid b_k - b_r \mid < \epsilon$ then
         $Q(b_r, \mu_r) \leftarrow \frac{Q(b_r, \mu_r) * N(b_r, \mu_r) + C}{N(b_r, \mu_r) + 1}$
         $N(b_r, \mu_r) \leftarrow N(b_r, \mu_r) + 1$
      else
         create a new point $b_w$ in $B_s$
         initialise $Q(b_w, \mu_r) \leftarrow c(b_w, \mu_k)$
      end if
   end for
   Update the policy
   $\pi(b_r) = \arg\min_\mu Q(b_r, \mu), \forall b_r \in B_s, \forall \mu \in A$.
until converged
```

Figure 2.8: MC policy optimisation algorithm

Once the optimal policy is found, it can be used as a *plan cache* by the Task Manager [17; 107], which stores each belief state seen during training and their corresponding optimal recommendations. Hence, during execution, if the Task Manager enters a belief state it has visited before during training, it will be able to re-use its precomputed optimal recommendations contained in its cache. In the case where the current belief state has never been seen during training, the cache plan will use a metric to find a neighbor of this current belief state. The distance between the current belief state and all the belief states contained in the Task Manager's cache plan will be computed. The closest neighbor will then be selected, and the Task Manager will select the optimal recommendation that has been found for it during training. Note that the metric or nearest neighbor search (NNS) technique used to retrieve the closest neighbor may have an impact on the quality of the neighbor selected.

2.3.2.3 POMDP-based task management

Compared to the MDP, one of the advantages of the Partially Observable Markov Decision Process framework is to handle the uncertainty that occurs in the task by considering that the user state is only partially observable.

In the field of dialogue systems, researchers have demonstrated the ability of a POMDP-based system to outperform its equivalent MDP [93; 116]. However, when the POMDP approach is applied to large domains where the number of states is high, two main challenges need to be faced. These are related to the belief state space dimensionality and the tractability of the policy optimization. One way to solve these issues is to use a factored POMDP as defined in [114].

Factored POMDP Factorization of the POMDP is a technique used in spoken dialogue systems [114; 115], where the user state s is defined such as $s = \{s_u, a_u, s_d\}$, with s_u being the user goal, a_u the user action and s_d the state of the dialogue. As explained in [114], the *factored representation reduces the number of parameters required for the transition function, and allows groups of parameters to be estimated separately.* To apply this technique to ADL management, the belief state's update equation is modified.

In the case of CogWatch, the user goal s_u is observable; indeed, contrary to a dialogue system, the aim of the CogWatch Task Manager is not to infer the request formulated by the user via his or her speech, but to allow the user to relearn how to perform a task that is specified at the beginning of each trial. Hence, as discussed in Section 2.3.1.4, we suppose that s can be factorized with $s = \left\{a_u,\ s_d\right\}$ where:

- The user's action a_u is the most recent action really made by the user.
- The history of the user state s_d is the sequence of actions representing what the user has achieved so far.

Hence, each part of Equation (2.12) can be expressed by Equations (2.19) and (2.20).

Observation Function

$$P(o' \mid s', \mu) = P(o' \mid a'_u, s'_d, \mu) \approx P(o' \mid a'_u) \tag{2.19}$$

The observation function, or **observation model**, is the probability that the ARS makes the observation o given that the Task Manager is in state s and previously selected the recommendation μ. It is simplified, taking into account the fact that, in CogWatch, the ARS observations only depend on the actions made by the user. In this form, the observation function corresponds to the ARS confusion matrix.

Transition Probability

$$P(s' \mid s, \mu) \approx P(a'_u \mid \mu) P(s'_d \mid s_d, a'_u) \tag{2.20}$$

As seen in Section 2.3.1.4, the transition probability is the probability for the Task Manager to be in state s', given that it was previously in s and selected recommendation μ. After decomposition of s, we can simplify the equation by making some assumptions:

- **The user action model:** The probability for the user to make a new action a'_u corresponds to his or her probability to follow the Task Manager recommendation μ : $P(a'_u \mid \mu)$; (i.e., compliance probability).

- **The task history model:** The probability for the user to be in state s'_d only depends on his or her current action a_u and state s_d he or she used to be in: $P(s'_d \mid s_d, a'_u)$.

When substituted in (2.12), they approximations give Equations (2.21):

$$b'(s'_d) = k \cdot P(o' \mid a'_u)P(a'_u \mid \mu) \sum_{s_d \in S} P(s'_d \mid s_d, a'_u)b(s_d) \ , \tag{2.21}$$

This factorization helps to simplify the belief state update process. Indeed, the latter is now only based on three update parameters: the observation model, the user action model and the task history model. This technique has proved its effectiveness in [114].

2.4 Summary

This chapter reviewed the research undertaken in the field of decision theory models, and how they could be applied to intelligent assistive/rehabilitation technology. An assistive system needs to be able to take decisions via its Task Manager, in order to provide recommendations to users during ADL. We explored the possibility of applying a cognitive decision model to the Task Manager, so it could potentially learn how to plan strategies like a human would. We explained the current constraints that do not allow the implementation of the cognitive models described, then focus on statistical decision models, such as the MDP and the POMDP. We then described how MDPs and POMDPs can be solved using reinforcement learning. This chapter also highlighted the MDP limitations and how a POMDP can overcome them by modeling uncertainties. The next chapter will introduce CogWatch, which will be the basis of the rest of this thesis.

Chapter 3

The CogWatch Project

3.1 Introduction

In this chapter, we will focus on the system CogWatch, give more insights about its main components, and explain the task for which it has been designed.

CogWatch is an assistive system designed to provide guidance to stroke survivors during activities of daily living (ADL). In this thesis, we will specifically focus on one type of activities: the tea-making task, and its variants. The overall goal of the system is to automatically generate instructional cues, so a user interacting with the system has a higher probability of successfully continuing or finishing a task. Its second goal is to detect when users make errors during the task and identify the cause of these errors when they occur.

In the following sections, we will explain the tea-making task, then describe the CogWatch's architecture.

3.2 Task definition

As discussed in Section 1.1, individuals with AADS have a high probability of making errors when performing activities of daily living. For example, in [100], the authors described the behavior of a patient with AADS when trying to make a cup of coffee. The authors reported that the patient made several errors, such as putting butter into the coffee, or adding coffee grinds into a bowl of oatmeal. Making a hot drink may appear as a “simple” task for people with no cognitive deficit or impairment. However, people with AADS are impaired in their cognitive ability to carry out ADL, which drastically decreases their independence on a daily basis. In order to help them regain independence, the choice has been made to design an assistive system which can target the issue related to the preparation of a hot drink such as tea. In CogWatch, 4 types of tea are considered: Black tea (BT), Black tea with sugar (BTS), White tea (WT) and White tea with sugar (WTS).

3.2.1 Actions tree

In [100], Schwartz et al. explained how an ADL can be divided into multiple levels of action steps. The most basic level of actions can be defined as “*the smallest component of a behavioral sequence that achieves a concrete, functional result or transformation, describable as the movement of an object from one place to another or as the change in the state of an object*” - [100]. These actions would correspond to the bottom level of the hierarchal tree described in Figure 2.1. These actions can be grouped together and form higher level actions, such as the actions at the top-level of the actions tree (Figure 2.1).
In CogWatch, the tea-making task is defined using a set A of top-level actions only. This set is composed of:

- “Fill kettle”
- “Boil water”
- “Pour kettle” (i.e., pour boiling water into the cup)
- “Add teabag”

- “Add sugar”
- “Add milk”
- “Remove teabag”
- “Stir”

These actions are part of the observations the Task Manager may receive from the ARS during execution. To describe a task in such a way enables the detection of different types of errors in the user’s behavior.

3.2.2 Error types

Schwartz et al. [100] distinguished between six error types: (1) place substitutions (moving objects to the wrong destination), (2) object substitution (misuse of objects, for example adding orange juice to the cup when making coffee), (3) anticipation (performing an action in the wrong sequence), (4) omissions (missing one step), (5) tool substitutions (using the wrong utensil), and (6) quality errors (the action was carried out but not in the appropriate way, for example, the packet of sugar was not completely opened). In the case of CogWatch and the tea-making task, the errors the Task Manager should be able to detect are defined as follows.

1. **Addition Error**: An addition error occurs when the user carries out an action that belongs to another task. For example, it is decided that the user should relearn how to make “Black tea”, but he or she adds sugar during the task.

2. **Perseveration Error**: A perseveration error occurs when the user repeats any action he or she has already performed during the task.

3. **Anticipation Error**: An anticipation error occurs when the user performs an action too early in time compared to his or her current history of actions. For example, stirring while the cup is empty.

4. **Perplexity Error**: A perplexity error occurs when the user does not perform any action during a specific amount of time T. A counter is reset after each observation received, and the Task Manager is configured to consider that the user needs assistance if no relevant action is received after T seconds.

5. **Omission Error**: An omission error occurs when the user considers that he or she has finished preparing his or her tea, but the Task Manager detects that the task is incomplete. For example, the user considers that the task is over, but he or she has not put teabag in the cup.

6. **Fatal Error**: A fatal error occurs when the user performs any action that can potentially cause injuries (for example toying with boiling water), or when one of the errors described above are repeated too many times.

In the next section, we will describe the structure of CogWatch and explain how it has been designed to provide instructional cues during the tea-making task.

3.3 CogWatch architecture

In Figure 3.1, one can see the main modules composing CogWatch. Following the general structure of assistive systems (see Section 1.1.3), the system is composed of a set of sensors (monitoring module), an action recognition system (ARS), a Task Manager, and a Cue Selector (cue generation module).

When interacting with a user, the system works as follows. First, the user chooses the type of tea (e.g., black tea with sugar) he or she would like to receive training for. When the task is chosen via a graphical interface, the user can begin the task. Immersed in an instrumented environment, the user is monitored by a Kinect^{TM} camera, while he or she moves sensorized objects at his or her disposal to perform an action (i.e., a_u in the figure) related to the task. When moved, these objects generate data that are passed to the ARS whose aim is to recognize the action the user has just performed. Processing the data in real-time, the ARS outputs an observation o to the Task Manager.

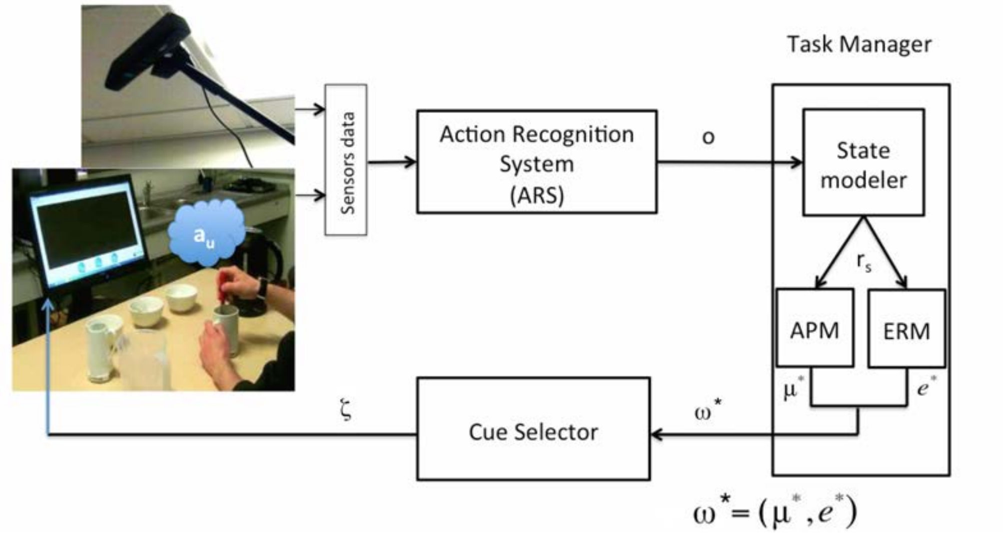


Figure 3.1: CogWatch architecture, APM: Action Policy Module, ERM: Error Recognition Module

The latter updates via the *State Modeler* its representation of the user state r_s, and uses it to select the optimal prompt ω^* to be output. This prompt is composed of:

1. The system's best next recommendation μ^*, which is found by the Action Policy Module (APM). This best next recommendation μ^* corresponds to the action the Task Manager considers the user should follow in order to successfully continue or finish the task,

2. The system's best understanding of whether the user has just made an error or not e^*; and if an error has been made, what was the type of this error. The output e^* is found by the Error Recognition Module (ERM).

Each Task Manager's prompt ω^* is passed to the Cue Selector, which displays, when necessary, the Task Manager's output in a way the user can easily understand: ζ (for example still images, video, recorded message). At this point, the user can make new actions and enter into new cycles with the system until the task is completed.

3.3.1 Sensors and sensorized objects

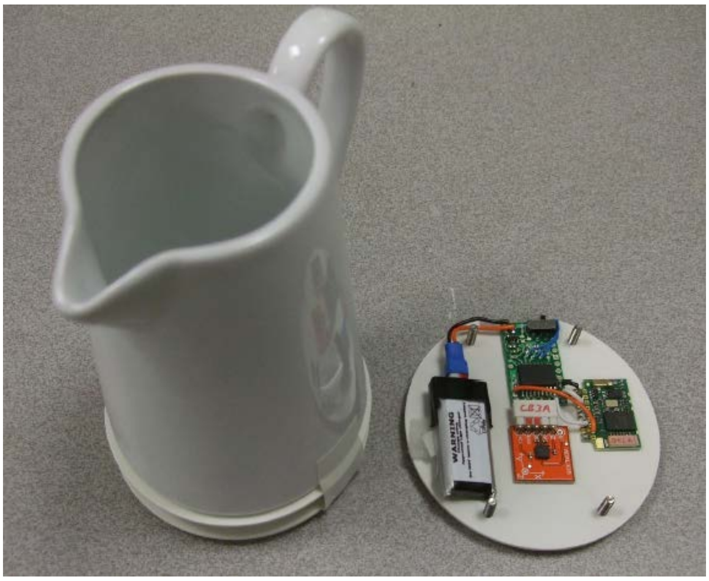


Figure 3.2: A jug fitted with a CogWatch Instrumented Coaster (CIC) and an 'open' CIC, showing the accelerometer, PIC microcontroller, Bluetooth module and battery [33]

The objects the user has access to during the tea-making task are a kettle, water jug, mug, milk jug, spoon and containers for the teabags, sugar and used tea-bags. Currently, only the kettle, mug and milk jug are instrumented. The chosen solution is to package the sensors and circuitry into an instrumented 'coaster' - the 'CogWatch Instrumented Coaster (CIC)', that is located under the objects as depicted in Figure 3.2. Each CIC contains a 3-axis accelerometers, 3 force sensitive resistors (FSRs), a PIC, a Bluetooth module and a battery.

The CICs function is to respond to changes in motion, tilting, and disturbances of the objects due to the addition of materials, stirring, collisions or (in the kettle) vibration during boiling. The FSRs can detect whether the object is standing on a surface or lifted, changes in weight due to the addition or removal of materials, and more subtle changes in weight distribution across the base of the object making it possible, for example, to detect stirring.

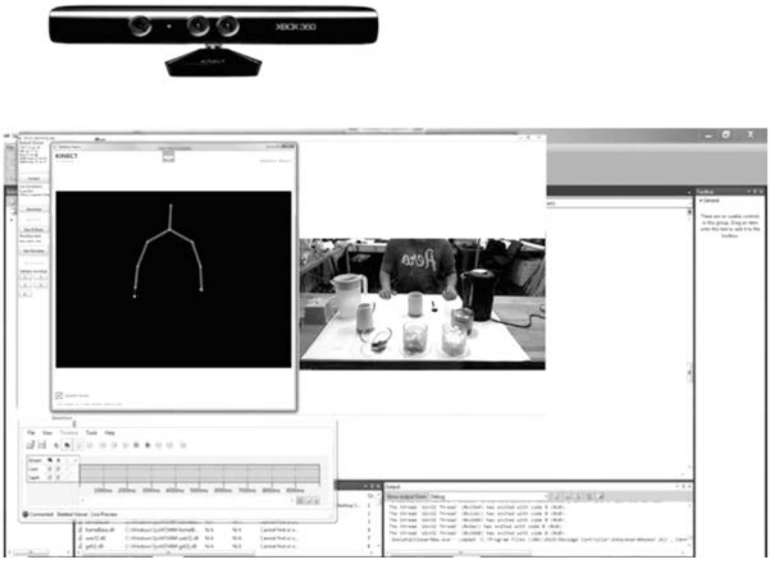


Figure 3.3: Kinect^{TM} user interface with skeleton view of arms and hands [33]

These coasters were designed and managed by Parekh et al. [33; 75], who were inspired by the MediaCup concept [26]. Indeed, in [26], the authors explained how to augment a coffee mug with sensing in order to track and capture human gestures when using the mug. It is interesting to note that, as such sensors are built almost invisibly into everyday objects, they allow the user to be monitored in an unobtrusive way. In addition to outputs from CICs, CogWatch's monitoring module collects hand-coordinate data captured via Kinect^{TM} [33]. Composed of a camera, a depth sensor, and an infrared projector, Kinect^{TM} enables the capturing and recording of 3D hand positions and color images from the user's environment. In Figure 3.3, one can see a screenshot of an interface, which shows how Kinect^{TM} raw data can be interpreted or visualized. The usability of the Kinect^{TM} was evaluated in both healthy and apraxic populations by Cogollor et al. [20], who were in charge of the implementation of Kinect^{TM} in CogWatch. In the study, results suggested that Kinect^{TM} is a reliable motion capture system in a cognitive rehabilitative context.

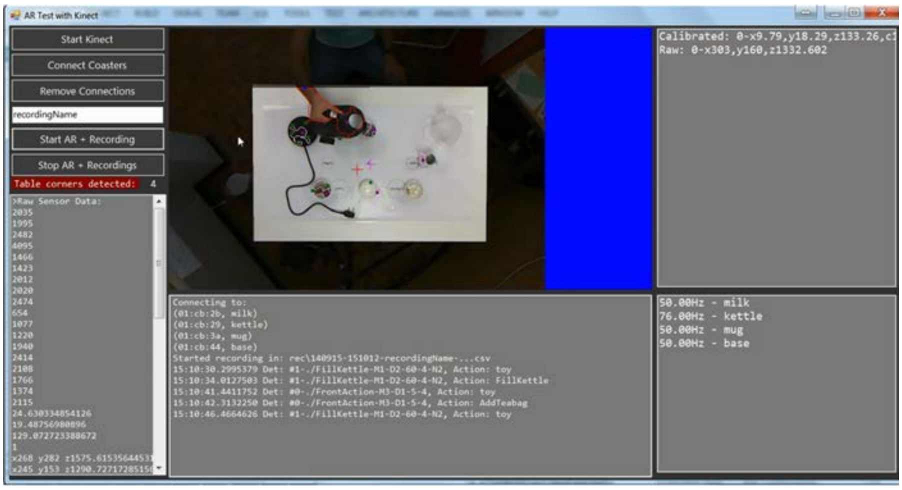
3.3.2 Action Recognition System

The sensorized objects and Kinect^{TM} continuously transmit data to the action recognition system (ARS) during the task. The aims of the ARS are to identify the actions made by the user and to send its observations to the Task Manager. The ARS can output ten different observations as far as the user's behavior is concerned. Eight out of these ten observations correspond to the actions defined in Section 3.2.1. The other observations correspond to erroneous actions that can be inferred from the way objects are moved or used: (1) "Pour cold water from jug to cup", (2) "Toying with boiling water" - users with AADS may have such hazardous behaviors that need to be detected early in the process in order to send them appropriate alerts.

In CogWatch, a Hidden Markov Model (HMM) based ARS is implemented [73]. HMMs are a generic framework for statistical sequential pattern processing [88]. Its utilization for action recognition has already been explored in previous research. For example, in [63], Liu et al. explained how a multi-HMM classification based on signals captured by Kinect^{TM} and a wearable sensor can enable hand-gesture recognition. The authors explained that each of their HMM detectors calculate a likelihood probability associated to a hand gesture, and that the gesture with the maximum average is considered to be the recognized gesture.

The framework used by the CogWatch ARS is similar to the one presented in [63]. During execution or evaluation, the sensors' data fed into several trained HMM based detectors, each responsible for detecting a specific action (see Section 3.2.1) or erroneous actions. The most probable explanation of the ARS input is then associated with a label indicating the action observed. This action label is then output and passed to the Task Manager.

Figure 3.4 shows a screen-shot from the real-time ARS. At the bottom of the screen-shot, one can see the various action labels printed on the screen as they are detected by the ARS.



```
Table corners detected:  4
```

```
>Raw Sensor Data:
2035
1995
2482
4095
1466
1423
2012
2020
2474
654
1077
1220
1940
2414
2108
1766
1374
2115
24.630334854126
19.48756980896
129.072723388672
1
x268 y282 z1575.6153564453
x245 y153 z1290.7271728515
```

```
Connecting to:
(01:cb:2b, milk)
(01:cb:29, kettle)
(01:cb:3a, mug)
(01:cb:44, base)
Started recording in: rec\140915-151012-recordingName-...csv
15:10:30.2995379 Det: #1-./FillKettle-M1-D2-60-4-N2, Action: toy
15:10:34.0127503 Det: #1-./FillKettle-M1-D2-60-4-N2, Action: FillKettle
15:10:41.4411752 Det: #0-./FrontAction-M3-D1-5-4, Action: toy
15:10:42.3132250 Det: #0-./FrontAction-M3-D1-5-4, Action: AddTeabag
15:10:46.4664626 Det: #1-./FillKettle-M1-D2-60-4-N2, Action: toy
```

```
Calibrated: 0-x9.79,y18.29,z133.26,c
Raw: 0-x303,y160,z1332.602
```

```
50.00Hz - milk
76.00Hz - kettle
50.00Hz - mug
50.00Hz - base
```

Figure 3.4: Screen shot showing the output of the real-time Action Recognition System [73]

3.3.3 Task Manager

As one can see in Figure 3.1, the Task Manager is composed of three main modules: the State Modeler, the APM and the ERM. When the Task Manager receives an observation o from the ARS, the State Modeler uses this information to re-create a representation r_s of the user's state. As discussed in Section 2.3.1.4, this representation of the user state is a simplified model of what the user has achieved so far during the task. This simplified state r_s is then passed to the APM and ERM, which decide what are the optimal prompts to be sent to the user during the task.

The APM is in charge of finding what is the optimal next recommendations μ^* the user should follow; and the ERM is in charge of tracking potential errors e^* in the user's behavior. Both μ^* and e^* compose a prompt ω^*, with $\omega^* = (\mu^*, e^*)$ that is sent to the Cue Selector. The process through which the APM and ERM select their outputs, and the way r_s is updated each time an observation is received, depend on the model the Task Manager is based on. In this thesis, the emphasis will be put on the MDP and POMDP based CogWatch Task Manager, which will respectively be fully described in Chapters 5 and 6.

The challenge of the Task Manager is to cope with the user's variability in task completion. Indeed, since users can have their own preferences or action plans as far as tea-making is concerned, this task can be completed in various ways. For example, when making a white tea with sugar, some ingredients can be added at any point in the task, such as adding milk or sugar. The Task Manager should not force the user to complete the task following one specific pattern, but adapt itself to what the user has achieved so far, and retrieve the appropriate information. In other words, when interacting with CogWatch, the user is free to go through the task as he or she usually does; and the Task Manager needs to be flexible enough to acknowledge the fact that different sequences of actions can lead to a correct accomplishment of the task. Only the errors defined in Section 3.2.2 draw a limit between all the potential correct ways to make a cup of tea, and what the Task Manager should label as erroneous user's behaviors.

3.3.4 Cue Selector

The information output by the Task Manager is processed by the Cue Selector. The latter decides when a cue should be sent to the user, and in which form this cue should be sent. Each cue is displayed via a graphical interface referred to as the "patient's interface" - see Figure 3.5. This interface is a Touch-Screen monitor that allows the user to receive information from CogWatch, and to trigger some of its functionalities when specific buttons are selected. Through this interface, the user has access to:

- A history of completed actions, provided by the Task Manager and displayed in the progress bar,
- A visual reminder of the type of tea currently selected (i.e., task selection),
- A "Finish button" that can be pressed if he or she considers that the task is finished,
- A "Help button" that can be pressed if he or she needs to receive the Task Manager prompt. The Cue Selector will then automatically generate a cue,

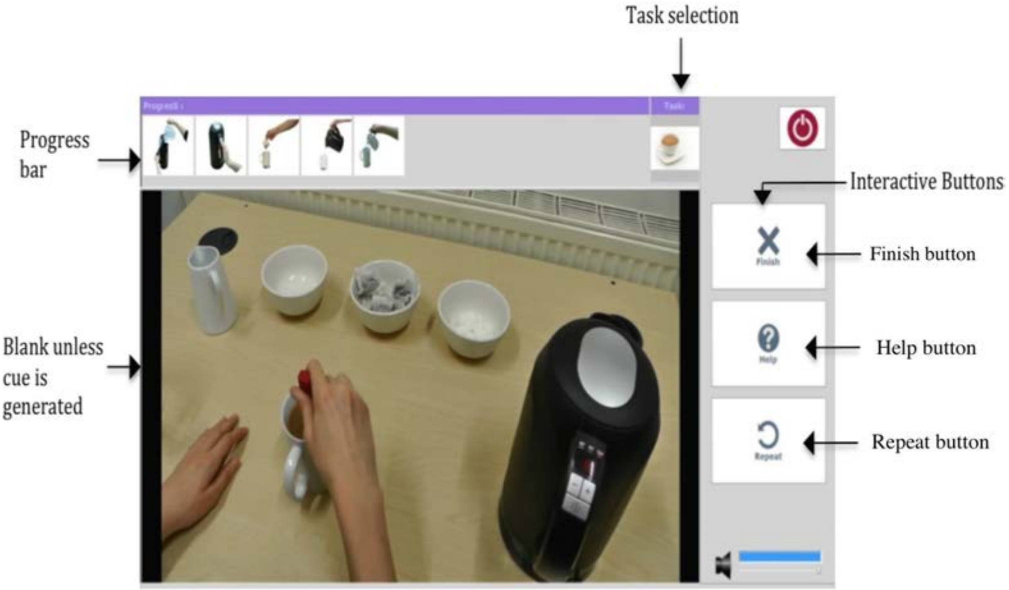


Figure 3.5: Screen shot showing the CogWatch patient's interface

- A "repeat button" that can be pressed if he or she needs to repeat the last prompt,
- A screen which displays a video or image of the prompt if necessary.

The Cue Selector is in charge of transforming the prompt ω^* output by the Task Manager in a user-friendly cue. Different types of cues can be sent to the user; ζ can be a video, a sound, a pictogram or a text conveying the information conveyed by ω^*. The type of cue can be chosen in advance by a clinician, depending on the user's preference or level of cognitive awareness.

The researchers in charge of this module have run several studies in order to explore the impact of different cues on users [12; 13; 42]. Moreover, depending on clinicians' choice, the Cue Selector can follow an errorless or errorfull method (Section 1.1). If it is decided that the system should provide cues only when errors are detected in the user's behavior, then each time the Cue selector receives a prompt $\omega^* = (\mu^*, e^*)$, it calculates the number of times F it has seen a specific error e^* since the beginning of the trial.

If at one point in the trial F is higher than a threshold, the Cue selector sends ζ to the user. Note that each type of errors (see Section 3.2) may have a different threshold based on the type of tea chosen. Each threshold can be chosen in advance by a clinician. If it is decided that the Cue selector should follow an errorless method, then each time the Cue selector receives ω^*, it will generate the corresponding cue ζ.

3.3.5 Interactive buttons

As discussed, the user can press a "Finish button" or "Help button" during trials via the patient's interface. When these events occur, the Task Manager is triggered and is configured to automatically output another prompt ω^*. It works as follows.

- **Finish button**: When the finish button is selected by the user, the ERM automatically verifies the validity of the current user state. If no fatal error is detected, the ERM outputs a specific label *C01* which means that the task was a success.

- **Help button**: When the help button is selected by the user, both the APM and the ERM retrieve the current user state from the State Modeler for the Task Manager to output its previous prompt ω^*. In this specific case, instead of evaluating whether to give a cue or not, the Cue Selector will pass the information contained in ω^* to the user as requested.

3.4 Implementation and definition of errors in the tea-making task

In this section, we focus on how errors are specifically defined and implemented by default during the tea-making task. In Table 3.1 one can see the specific types of errors the ERM can automatically detect during a trial. This table contains the rules the ERM uses to detect whether an error has been made by a user, the "ID" corresponding to these errors, and for which tasks these errors are relevant. Table 3.2 shows how many times a specific type of error can be repeated during a trial, before it is considered that the user has failed the task. If the errors defined in Table 3.1 are repeated x times, and that x is higher than a threshold (see Table 3.2), the system considers that the user has failed his or her task. As explained in Section 3.3.4, each time a specific error is detected by the Task Manager, the information is sent to the Cue Selector which increments the number of time F this error occurs during the trial. If at one point in the trial F is superior to a threshold, as shown in Table 3.3, then the Cue selector sends to the user the best next recommendation of the Task Manager.

All the information given in these tables were defined by the CogWatch psychologists' team, and based on their observations of the most common stroke survivors' behaviors. The main rational for the definition of these errors was to enable the system to assess users with AADS and ensure that the task could be achieved. For example, the goal of the ERM is to be able to respond to health and safety concerns, in the case where users "Toy with boiling water" (see error *E03* in Table 3.1), and detect when :

- users' actions violate meaningful sequence (for example pouring water to cup before boiling water - *E12*, or removing teabag before adding boiled water to cup - *E28*),
- users' action clearly contrast with the tasks goal (for example adding sugar to black tea with no sugar - *E21*; not adding milk when a tea with milk should be made *E26*),
- users hesitate for too long or request for help (i.e., *E01*, *E02*).

These errors were also defined to take into account the current limitations of the system. Indeed, the current ARS can only output a finite set of observations, with which the ERM cannot infer all types of errors a human can make. Thus, the errors described in Table 3.1 focus on the most observable errors made by stroke survivors during the tea-making task, and what the system is currently able to observe and detect.

3.5 Summary

In this section, we gave an overview of the main modules composing CogWatch, and explained the task for which the system has been designed. We enumerated the actions the system is able to detect during a task, and described the user's errors it should automatically recognize. References to similar work related to each module were given.

Except for the Task Manager, all the other modules (i.e., sensors, ARS, Cue Selector) were designed or managed by other researchers and colleagues who have been cited accordingly. The focus of this thesis is the Task Manager and its impact on the ability of CogWatch to guide users during tea-making. All the other modules' intrinsic processes are unobservable from its point of view.

Before going through the details related to the MDP and POMDP based Task Manager, the next Chapter will focus on how the Task Manager can be trained via simulation to select appropriate prompts, and how it can be evaluated within a virtualization of the whole system.

Table 3.1: Types of user's behaviors that the ERM can detect. BT = Black tea, BTS = Black tea with sugar, WT = White tea, WTS = White tea with sugar, "all" = [BT, BTS, WT, WTS]. AD = Addition error, OM = Omission error, FE = Fatal error, PE = Perplexity error, PsE = Perseveration error, AN = Anticipation error, QT = Quantity error, BTr = button trigger, NE = Not an error.

ID	Error type	**Task**	**Description of user interaction with CogWatch**
E01	BTr	all	Presses "Help Button"
E02	PE	all	Makes long pause during task
E03	FE	all	Toys
E04	BTr	all	Presses "Finish Button" when not required
E05	OM	all	Omits to press "Finish Button" when required
E06	PsE	all	Adds water to kettle multiple times
E07	OM	all	Omits to add water to kettle
E08	OM	all	Omits to boil water
E09	PsE	all	Boils water multiple times
E10	OM	all	Omits to add teabag to cup
E11	PsE	all	Adds teabag multiple times
E12	AN	all	Pours water to cup before boiling water
E13	PsE	all	Pours water to cup multiple times after boiling water
E14	OM	all	Omits to add boiled water to cup
E15	OM	all	Omits to remove teabag from cup when required
E16	AN	all	Stirs while no water is in the cup
E17	OM	BTS, WTS	Omits to stir
E18	PsE	BT	Stirs multiple times
E19	PsE	WT, BTS	Stirs multiple times
E20	PsE	WTS	Stirs multiple times
E21	AD	BT, WT	Adds sugar when not required (based on type of task)
E22	QT	BTS, WTS	Adds too much sugar
E23	OM	BTS, WTS	Omits to add sugar
E24	AD	BT, BTS	Adds milk when not required (based on type of task)
E25	QT	WT, WTS	Adds too much milk
E26	OM	WT, WTS	Omits to add milk
E27	AN	all	Boils water before adding water to kettle
E28	AN	all	Removes teabag before adding boiled water to cup
C01	NE	all	Task successfully completed

Table 3.2: Number of times an error can be repeated during a trial until the system considers that the user has failed the task.

ID	Fatal when repeated x time(s)
E01	3
E02	3
E03	1
E04	3
E05	2
E06	4
E07	3
E08	3
E09	4
E10	3
E11	1
E12	1
E13	5
E14	3
E15	3
E16	3
E17	3
E18	5
E19	6
E20	7
E21	1
E22	1
E23	3
E24	1
E25	4
E26	3
E27	1
E28	1

Table 3.3: Maximum number of times an error can be repeated during a trial before the Cue Selector begins to send cues to alert the user about his or her behavior.

ID	When error occurs F times
E01	1
E02	1
E03	1
E04	1
E05	1
E06	2
E07	1
E08	1
E09	2
E10	1
E11	1
E12	1
E13	2
E14	1
E15	1
E16	1
E17	1
E18	2
E19	3
E20	4
E21	1
E22	1
E23	1
E24	1
E25	2
E26	1
E27	1
E28	1

Chapter 4

Simulation

4.1 Introduction

In this Chapter, we will describe the Simulated Users ($SimUs$) which have been implemented for training and/or evaluation of the CogWatch Task Manager. We will explain why a simulated approach was taken, and the advantages and disadvantages of this approach. Since the aim of a Simulated User is to solely train or evaluate the Task Manager, the whole CogWatch system can be virtualized and simplified from the Task Manager's point of view. An overview of the virtualization of the system will be given, and we will explain how it can be used with a Simulated User.

4.2 Simulated User ($SimU\alpha$)

As discussed, CogWatch is composed of different modules. Each module goes through its own process based on the type of inputs it receives and outputs it needs to deliver. The inputs of the Task Manager are discrete and correspond to action labels output by the ARS, as defined in Section 3.2.1. Contrary to the ARS's challenge, which is to cope with the variability in the way that actions are performed when the user moves sensorized objects, the Task Manager's challenge is to cope with the variability in task completion and the fact that the outputs of the ARS may be erroneous.

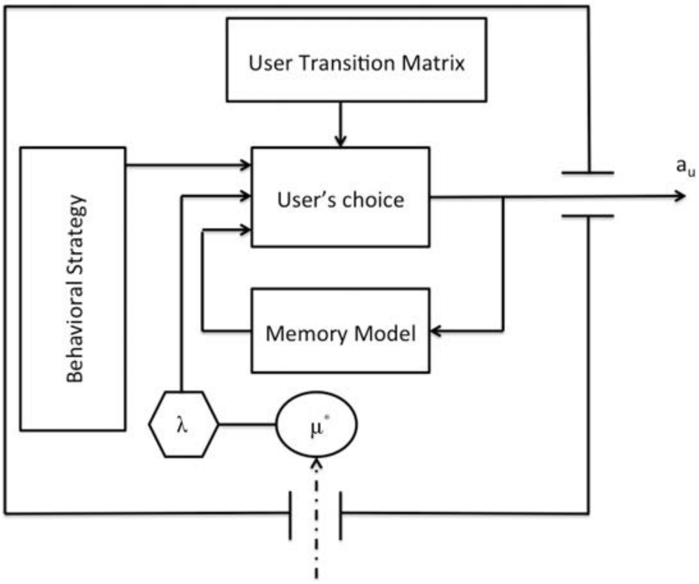


Figure 4.1: The architecture of the Simulated User ($SimU\alpha$).

Hence, to train or evaluate the Task Manager via simulation, we need a component able to output user's actions of the same type that the Task Manager's inputs, and display the same variability in task completion as real users.
In this thesis, the implemented simulated user is based on bigram probabilities estimated from stroke survivors' data. Its architecture and functioning are explained in the next section.

4.2.1 Simulated user's architecture

The Simulated User ($SimU\alpha$) that has been designed to enable training or evaluation of the CogWatch Task Manager is composed of 5 main modules, as seen in Figure 4.1. The core module is the *User's choice*, which takes as inputs parameters from the *User transition matrix*, the *Memory Model*, the current APM best next recommendation μ^* (see Section 3.3.3), and the *Behavioral Strategy* module. When $SimU\alpha$ generates an action a_u, the Task Manager responds with a prompt ω^*.

If the Cue Selector does not let the Task Manager's output reach $SimU\alpha$, the latter chooses what action to take by itself (i.e., the simulated user follows its own process). If the Cue Selector lets the Task Manager's output reach the simulated user (see Section 3.3.4), the simulated user takes into account the optimal recommendation μ^* contained in the prompt with a compliance λ ($0 \leq \lambda \leq 1$), then selects its next action a'_u. In other words, if $SimU\alpha$ compliance is $\lambda = 0$, then $SimU\alpha$ always ignores μ^* and selects its next actions taking into account its own process. In this specific case, the Task Manager has no impact on the simulated user's performance. Conversely, if the compliance is $\lambda = 1$, $SimU\alpha$ will always follow the recommendation μ^* provided by the Task Manager, and $a'_u = \mu^*$. For example, if $\lambda = 0.7$, then $SimU\alpha$ has 70% chance of selecting the Task Manager's output as its new action a'_u.

$SimU\alpha$ uses a *User's transition matrix* during its action selection process. This *User's transition matrix* is based on action bigram probabilities from data generated by 63 control and cognitively impaired participants, aged between 21 and 82, who completed different types of tea. The database is composed of 156 different sequences of actions that real participants performed when going through the tasks. This data provides the simulated user with limited information about the order in which human subjects execute the various actions during different tasks. This information is processed to emulate the variability with which a task can be achieved. However, as only bigram probabilities are taken into account, this knowledge is incomplete. Indeed, bigrams only provide conditional probability for a specific action given the previous one. They do not take into account the whole history of actions performed, which limits the simulated user's behavior. For example, suppose that the action that has just been performed is "Fill kettle": bigrams will allow to have access to the probability for the user to perform the action "Boil water" (or any other action listed in Section 4.2) after "Fill kettle". However, these probabilities will not take into account any of the actions that have been performed before "Fill kettle".

In order to compensate for this incomplete knowledge, $SimU\alpha$ has access to three behaviors through the *Behavioural strategies* module:

1. $SimU\alpha$ can select the “Finish button” or the “Help Button” (see Section 3.3). This will force the system to deliver the Task Manager prompt to $SimU\alpha$ which will comply with it (i.e., $\lambda = 1$ for one step).

2. $SimU\alpha$ can decide to do nothing, which will trigger the Task Manager after a specific amount of time, forcing the latter to deliver a prompt with which $SimU\alpha$ will comply (see Section 3.3).

3. $SimU\alpha$ can perform a random but meaningful action. A meaningful action is an action that is not impossible for a human to perform. For example removing a teabag from a cup while the latter was not put in beforehand, is not a meaningful action.

$SimU\alpha$ also has access to 5 different types of *memory* through the *Memory model* module. Each type of memory has a specific impact on how $SimU\alpha$ remembers the history of actions it has already performed:

1. $SimU\alpha$ can remember the last action it performed only.

2. $SimU\alpha$ can remember all the actions it performed and can repeat some of them. Suppose that $SimU\alpha$ history of action is $h = (a_1, a_2, a_4)$ and the new action it decides to take is $a'_u = a_2$. In this case, if a_2 is a meaningful action that can be performed given the context provided by h, then $SimU\alpha$ will be allowed to select a_2 as its next action, even if this action is part of the actions it has already selected earlier in the task.

3. $SimU\alpha$ can remember all the actions it performed and cannot repeat any.

4. $SimU\alpha$ can have a probability δ to forget the actions performed in the past and cannot repeat any actions it still remembers.

5. $SimU\alpha$ can have a probability δ to forget the actions performed early in the task and can repeat some of those that it still remembers.

Points (4) and (5) can be explained as follows. Suppose that at step t, $SimU\alpha$ history of action is $h_t = (a_1, a_2, a_4)$ and h_t is in $SimU\alpha$ memory. To say that $SimU\alpha$ has a probability δ to forget its previous actions means that, at $t+1$, the probability for $SimU\alpha$'s earliest action a_1 to disappear from its history is δ. Thus, if this probability leads $SimU\alpha$ to effectively "forget" a_1, then it will select its new action a'_u considering that $h_{t+1} = (a_2, a_4)$.

4.2.2 Simulated User's process (example)

Now that the architecture of $SimU\alpha$ is given, we summarize the way it works with an example. Suppose that it is decided to have a $SimU\alpha$ which "remembers all the actions it performed and cannot repeat any of them", and "performs a random but meaningful action" when necessary. These options will be referred to as rules R_1 and R_2.

Suppose that we are at step t and $SimU\alpha$'s actions history is $h_t = (a_1, a_2, a_4)$. The Task Manager sends the best next recommendation $\mu^* = a_{10}$ to the Cue Selector. The Cue Selector lets this information reach $SimU\alpha$, which receives $\mu^* = a_{10}$. The *SimU* complies with this input with a probability λ. If $SimU\alpha$ complies, then its output is automatically $a'_u = a_{10}$. If it does not comply with its input, then the simulated user goes through its own process to select its next action. In the case where $SimU\alpha$ goes through its own action selection process, it focuses on the action it selected at $t-1$, which is a_4 in our example. It looks into the *User transition matrix* containing the probabilities for a real user to perform any other action after a_4. Taking into account these bigram probabilities, the simulated user pre-selects its next action a'_u. Following R_1, if the pre-selected next action is a meaningful action and is not already contained in h_t, then $SimU\alpha$ will be allowed to output it. Conversely, if the pre-selected next action does not respect R_1, this pre-selected next action will not be output and will need to be changed.

In this case, the simulated user will follow rule R_2 provided by the "Behavioral strategy" module and randomly select another action that will be meaningful and not part of h_t.

In this thesis, *SimU*α was implemented following this configuration during training and evaluation. However, during training this *SimU*α had the possibility to choose its next actions by itself or to follow the system's recommendations when available; while during evaluation, it had a fixed level of compliance toward the system's recommendations (for example $\lambda = 1$, see Chapter 6.)

4.3 Simple actions generator (*SimU* β)

In order to analyze the potential impact of a different simulated user on the Task Manager, another simulated user is proposed. In this thesis, it will be referred to as *SimU*β. Contrary to *SimU*α, *SimU*β is not based on bigrams calculated from real participants' data, and uses a simpler algorithm than *SimU*α. It works as follows.

At the beginning of each trial, when a specific task is chosen, *SimU*β is given access to a specific database composed of examples of trials performed by real participants. This database is different from the one used by *SimU*α to calculate its bigrams. The database used by *SimU*β is composed of 626 different sequences of actions, all generated by 27 stroke survivors, aged between 52 and 82. *SimU*β is configured to randomly select a sequence of actions from the database, and to follow the actions composing the sequence up to 50% of the sequence's length. When *SimU*β does not follow the actions contained in the selected sequence, it complies to the Task Manager's prompt at 100%. Similarly to *SimU*α, *SimU*β interacts with the virtualization of the CogWatch system that will be described in the next section 4.4.

The aim of this other simulated user is to analyze the ability of the Task Manager to fulfill its goal when interacting with simulated users which do not only display different behaviors but which are also based on different databases. Note that in this thesis, contrary to *SimU*α, *SimU*β was implemented during evaluation only.

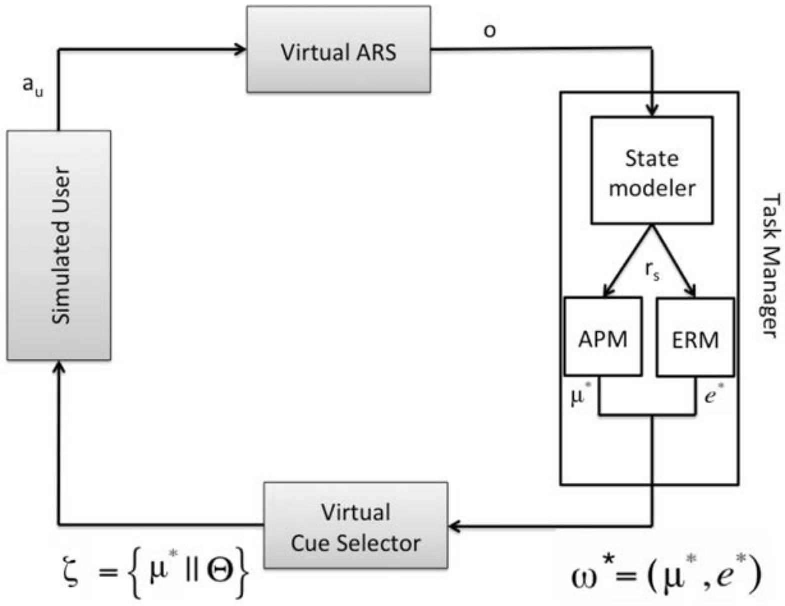


Figure 4.2: Architecture of the Virtualization of CogWatch.

4.4 Virtualization of CogWatch

To interact with the Task Manager, a simulated user can be implemented into a virtualization of CogWatch, which is similar to the real system described in Figure 4.2. As one can see, the virtual model of CogWatch is composed of a simulated user, a virtual ARS and a virtual Cue Selector. It works as follows.

When the task is chosen, the simulated user starts with an empty actions history, then takes an action a_u that is sent to a virtual ARS. The virtual ARS is implemented as a $N \times N$ confusion matrix C (N is the number of actions) whose i, j^{th} entry is the probability that the ARS outputs observation j when the user executes action i. Thus, if a_u is the i^{th} action, then the output of the virtual ARS is determined randomly according to the i^{th} row of C. The ARS observation is then passed to the Task Manager (see Section 3.3).

After receiving this observation, the Task Manager sends its optimal prompt $\omega^* = (\mu^*, e^*)$ to a virtual Cue Selector. The latter is a filter, which sends $\zeta = \left\{\mu^* || \Theta\right\}$ to the simulated user, where μ^* is the next best action the Task Manager considers the user should perform, and Θ a signal forcing the simulated user to chose the next action by itself. Note that the symbol "||" denotes the logical OR operator. In other words, each time the Cue Selector passes its output to the simulated user, this output only contains μ^* **or** Θ. For example, Θ is sent instead of μ^* if the Task Manager has not detected any error in the simulated user's behavior. Conversely, if an error is detected, then the Task Manager's output μ^* will be part of ζ.

Note that this virtual model of CogWatch has been designed with the sole purpose of training or evaluating the Task Manager. Thus "virtual sensors" do not appear in this architecture. Indeed, from the Task Manager point of view, all the complexity related to action detection via sensors output, and the impact of such outputs on the ARS, can be synthesized by a confusion matrix whether the user is real or virtual. Similarly to the real Cue Selector, the virtual Cue Selector chooses when to send the information provided by the Task Manager to the user by processing e^*. However, it does not emulate the ability of the real Cue Selector to choose the type of cue (for example, audio, video).

4.5 Training via user simulation

A simulated user can be implemented to train the Task Manager. The notion of training corresponds to the policy optimization process the Task Manager goes through in order to "learn" how to select appropriate prompts during a task (see Chapter 2). Two approaches can be applied during policy optimization: model-based or simulation-based (also known as model-free) approaches [90]. When applying a model-based approach, an annotated corpus is used to estimate the state transition probabilities for the MDP, and both transitions and observation probabilities for the POMDP. However, this approach has numerous deficiencies:

- A corpus needs to be available and large enough to robustly estimate the probabilities.
- The corpus needs to be annotated, so the true identity of the task needs to be known.
- Learning with a fixed corpus means that the Task Manager will only be able to explore the states recorded at the time the corpus was created.

At the current time, no such annotated corpus is available for CogWatch. Theoretically, it is possible to optimize a policy by letting real users interact with the system. However, the policy's performance at the initial phase of learning is generally too low to be acceptable, especially when the target users are cognitively impaired. Moreover, many trials are necessary to train the system, which is not feasible with real users. Hence it is common practice that a simulated user is implemented to interact with the system during policy optimization. For example, this approach has been applied in [27; 61; 87; 89; 96]. Many techniques, such as graph-based and agenda-based techniques can be applied to design a simulated user [95; 96; 98]. Statistical methods, such as n-gram [28], cluster-based user simulation [89], Bayesian networks [82] and hidden-agenda [97] have also proved their ability to make the process of task modeling automatic [61].

With this simulated-based approach, a simulated user can generate any number of training episodes and a variety of scenarios, so the Task Manager can explore the state space in a more exhaustive way. A risk with this approach is that the simulated user implemented does not properly capture real users behaviors.

4.6 Evaluation via user simulation

User simulation is a common way to evaluate POMDP techniques [105; 113]. It is known to provide a useful basis for comparing systems [115], if the behavior of the simulated user accurately reflects the behaviors of real users.
Intuitively, one could consider that the best approach to evaluation is to let users interact with the system. However, when participants are not available or when it is planned to run exhaustive trials, an alternative is to let a simulated user be in charge of the evaluation. This will enable a wider coverage of user space, and the possibility to analyze the main elements in the Task Manager structure (i.e., APM and ERM - see Section 3.3.3) that have an impact of its ability to fulfill its goals.

The evaluation approach taken in this thesis is based on the $SimUs$ described in Section 4.2.2 and 4.3, when interacting with the virtualization of CogWatch described in Section 4.4. The configuration of the $SimUs$ and virtualization of CogWatch were defined differently for training and evaluation, so the inputs received by the Task Manager during training would not necessarily correspond to those received during evaluation. Indeed, only $SimU\alpha$ was used during training, at random level of compliance toward the Task Manager's recommendations. During evaluation, $SimU\beta$ and $SimU\alpha$ were implemented, and $SimU\alpha$ was configured to have a fixed level of compliance in this context.

By applying a simulation-based approach during evaluation, we allowed various comparisons between the POMDP and MDP-based Task Manager, at varying Action Recognition System's error rates, and Nearest Neighbor Search techniques. The aim was to measure the impact of the Task Manager's intrinsic parameters on its ability to deliver correct prompts during a task. In other words, the aim of the evaluations that will be described in this thesis is to measure the performance of the Task Manager rather than the whole system.

4.7 Summary

In this chapter we described the architecture of two Simulated Users and the virtualization of CogWatch with which they interact, in order to train or evaluate the Task Manager. We gave an overview of the advantages of such a simulation-based approach, and highlighted other research where it has also been applied. The Simulated Users used in CogWatch are based on stroke survivors' data, composed of different sequences of actions they performed during different tasks.
The Simulated User referred to as $SimU\alpha$ is flexible and can be configured to display different behaviors during training and evaluation. In this thesis, this $SimU\alpha$ was used during evaluation of the MDP-based Task Manager. The details related to this evaluation will be given in Section 5.4.2. The simulated user was also used during training and evaluation of the POMDP-based Task Manager, see Chapters 6 and 7, in respectively Sections 6.3 and 7.3.
A different Simulated User ($SimU\beta$) was also introduced. The latter was used during evaluation of the MDP and POMDP-based Task Manager to verify whether the evaluation comprises any user-dependency.

In the next Chapter, we will explain how an MDP-based Task Manager can be defined and implemented in an assistive system such as CogWatch. We will then describe the evaluations that have been run, and analyze the results obtained via simulation and with real participants.

Chapter 5

MDP-based Task Manager

5.1 Introduction

This Chapter gives an in-depth description of the MDP-based Task Manager implemented in the CogWatch system, where Section 5.3 explains how training and evaluation are performed with real participants and via user simulation.

5.2 CogWatch MDP-based Task Manager

5.2.1 Task formalism

In Section 3.3, we described the general architecture of the CogWatch system and the Task Manager. In the case of the MDP-based Task Manager, the State Modeler depicted in Figure 3.1 is replaced by an Actions History module as shown in Figure 5.1. The system works as explained in Section 3.3. Nevertheless, it is important to focus on the observations o output by the ARS and how they are processed by the MDP-based Task Manager.

Each time the user or Simulated User performs an action a_u, the ARS outputs an observation o based on its interpretation of the action that has just been performed. This observation may be incorrect. However, the CogWatch MDP-based Task Manager assumes that the user's environment is fully observable. Thus, it believes that the output of the ARS is an accurate description of the actions performed by the user.

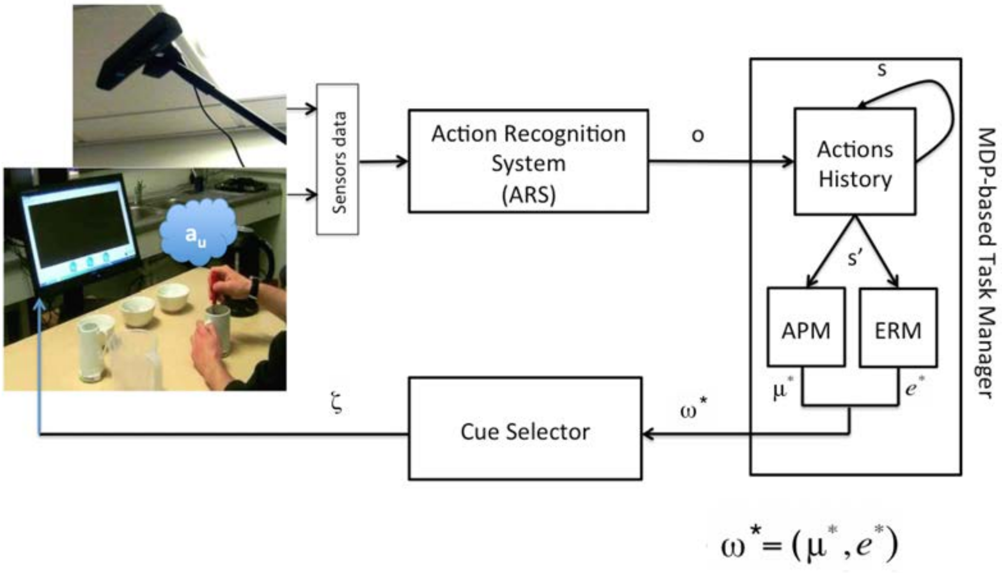


Figure 5.1: Structure of CogWatch with the MDP-based Task Manager.

In this case, the Task Manager keeps track of the user's history of actions via the Actions History module. The previous state s is always kept in its memory and re-used when a new observation is received, in order to update a new state s'. This new state s' is then processed by the APM and ERM in order to infer what action μ^* the user should do next, whether he or she has made an error during the task and, if an error has been made, what type of error has been made e^*.

5.2.2 State representation

One of the contributions of the work presented in this thesis is to define the MDP states as sequences of actions supposedly performed by the user during trials. In this subsection only, suppose that at step t the MDP-based Task Manager state is s_t, and for simplification purpose, suppose that the user's history of actions $s_{d,t}$ is the user's state.

At step $t = 0$, which corresponds to the beginning of each trial, the assumption is made that the user has made no action so far and the MDP-based Task Manager state is empty. At each step, the user performs an action $a_{u,t}$ which is added to $s_{d,t}$. Thus, $s_{d,t}$ grows with the number ς_t of actions made at step t, such that $|s_{d,t}| = \varsigma_t$. Here, $|s_{d,t}|$ denotes the size of the sequence $s_{d,t}$. For example, if $s_{d,t} = (a_1, a_1)$ then $|s_{d,t}| = 2$.

5.2.2.1 APM reduced state representation

Recall that the aim of the MDP-based APM is to plan what should be done next by the user. Its state s_t should recover the user state $s_{d,t}$ at step t. For this to be possible, one solution is to have $s_t = s_{d,t}$ at all steps. However, due to the high variance in task completion, the size of the user state approaches a very large number during trials. Indeed, since the user is free to perform a task as he or she wishes, any action can potentially be performed an infinite number of times. This would lead the MDP state space to become intractable (if $s_t = s_{d,t}$ at all steps). In order to keep a finite number of states in the MDP state space, another of the contributions presented in this work is a technique that systematically restricts the growth of the state taken into account by the MPD-based APM during trials. In this section only, we will note $\tilde{s}$ the state processed by the MDP-based APM, which is obtained when the growth of state s is restricted by the rule described as follows.

At the beginning ($t = 0$), we suppose that the start state is empty, with $\tilde{s}_0 = s_0$. At each step t, when the user makes an actions $a_{u,t}$, the latter is added to the MDP-based APM state $\tilde{s}_t$ only if the newly generated MDP-based Task Manager's state s_{t+1} is "valid" after this addition. A state is considered to be "valid" if it does not contain any of the errors defined in Section 3.2.2, or summarized in Table 3.1. This means that the MDP-based APM states $\tilde{s}$ copy states s only if the latter are "valid": all actions that could lead $\tilde{s}$ to contain any error defined previously are ignored.

Thus, any action added to the MDP-based APM state must be non erroneous and a non-redundant continuation of the current state s. This assumption is based on the fact that, even with such a restriction, the Task Manager APM state $\tilde{s}$ still contains enough information for the APM to fulfill its goal.

In Figure 5.2, one can see an example showing how, from the MDP-based APM's perspective, the user state evolves based on the actions made by the user, and how the restriction technique allows the MDP-based APM to maintain a reduced state. Figure 5.2 can be explained as follows.

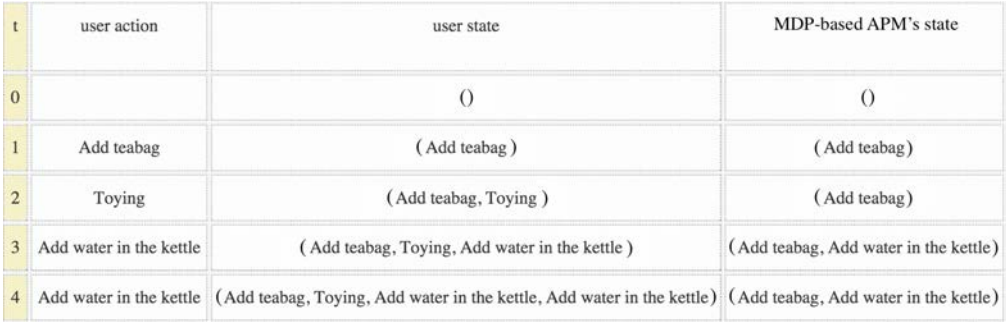
- At $t = 0$, as discussed, both MDP-based APM and user state are empty : no action has been made by the user yet.

- At $t = 1$, the user makes action $a_{u,1}$ =“Add teabag”. As one can begin making a cup of tea by doing so, it is valid and the MDP-based APM copies it, thus $\tilde{s}_1$=$s_{d,1}$ =(“Add teabag”).

- At $t = 2$, the user toys with the kettle $a_{u,2}$ =“Toying”. The user state automatically takes this actions into account, with $s_{d,2}$ =(“Add teabag”, “Toying”). However, this action is not valid as it can cause injuries. Moreover, “Toying” is defined as an error in Table 3.1 (see E03). Thus, the APM ignores this actions and $\tilde{s}_2$ remains the same, such that $\tilde{s}_2$ =(“Add teabag”).

- At $t = 3$, $a_{u,3}$ =“Add water to the kettle”, which is valid, so both $\tilde{s}_3$ and $s_{d,3}$ are updated.

- At $t = 4$, the user repeats the same action $a_{u,3}$ =(“Add water in the kettle”). The user state takes it into account during its update, but not the APM's state, as this action has already been performed. Indeed, redundancy is not allowed in the MDP-based APM state (see E06 in Table 3.1). Thus $\tilde{s}_4$ remains the same as $\tilde{s}_3$.



t	user action	user state	MDP-based APM's state
0		()	()
1	Add teabag	(Add teabag)	(Add teabag)
2	Toying	(Add teabag, Toying)	(Add teabag)
3	Add water in the kettle	(Add teabag, Toying, Add water in the kettle)	(Add teabag, Add water in the kettle)
4	Add water in the kettle	(Add teabag, Toying, Add water in the kettle, Add water in the kettle)	(Add teabag, Add water in the kettle)

Figure 5.2: Example of state representation in the MDP-based Task Manager

5.2.2.2 ERM state representation

In the MDP-based Task Manager, the APM models the task as an MDP and uses reinforcement learning to find the optimal policy for each state contained in the MDP state space. Therefore, there is a need to restrict the size of the state space. However, the ERM simply applies a set of verification rules to each state that it receives; no learning process is required. Thus, the state processed by the ERM in the MDP-based Task Manager is not subjected to any restriction. To detect errors in the user's state, the ERM uses the full user state, which corresponds to the second column in Figure 5.2. In other words, when the APM takes into account a state where no error is contained, the state of the ERM directly corresponds to the user state: it takes into account the full Task Manager's state s_t, in the case where $s_t = s_{d,t}$ for all steps t.

In the rest of this thesis, for readability purpose, we will always refer to the Task Manager state as s. When the focus is on the MDP-based APM, s will correspond to the state after application of the restriction rules described above. When the focus is on the ERM, s will correspond to the Task Manager state with no restriction applied.

5.2.3 Policy representation

The APM state restriction technique explained in the previous section gives the possibility to the APM of performing policy optimization on a *summary space*. From the APM perspective, the full cycle of recommendation selection is described in Figure 5.3. In this figure, the *Master Space* corresponds to the set of all sequences of actions the user can go through during a task, when no restriction rule is applied. The *Summary Space* corresponds to the set of sequences of actions the MDP-based APM can update, taking into account the restriction rules described previously. In other words, the MDP-based APM state space contains all the combinations of all possible restricted user states (i.e., all valid user states), which means that any user state from the *Master Space* has its corresponding *summary state*. In Figure 5.3, we can see that after each user action, the APM maps the new user state from the *Master space* into a *Summary space* for which it selects a recommendation μ^*. This recommendation then fills its corresponding slot in the prompt ω^* to be sent to the next module in CogWatch.

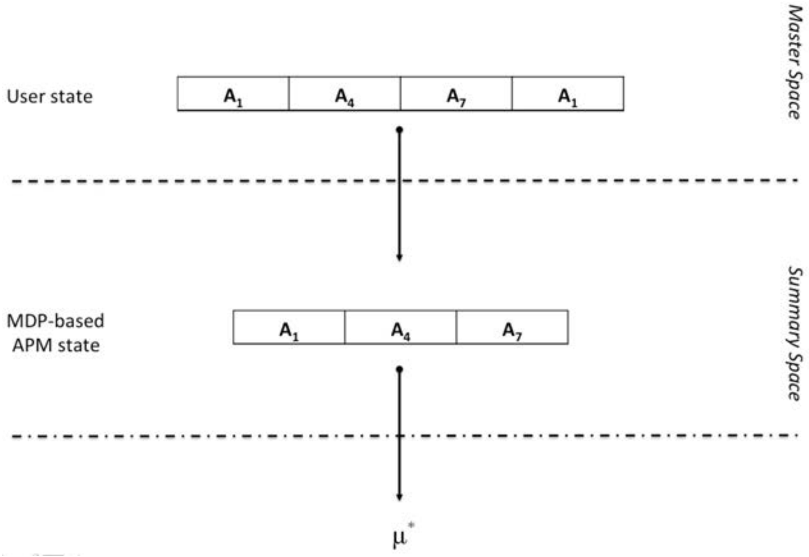


Figure 5.3: MDP-based APM state mapping, with $A_{1,4,7}$ some of the actions the user can perform, and μ^* the recommendation selected by the APM.

5.2.4 Error recognition

Contrary to the MDP-based APM, the ERM does not base its functionality on any reinforcement learning process. Because the MDP-based APM state space only contains states that clinicians define as valid, then any user state that is not part of the *Summary space* is considered to be erroneous by the ERM. Once a state is detected as erroneous, the second goal of the ERM is to find what type of error it is. To do so, it takes as input the full state s (i.e., which is a complete copy of the user state), then automatically compares it with different rules that have been encoded based on clinicians' definitions of specific types of error (see Table 3.1). When a type of error is detected, the ERM looks for the cause of this error (i.e., the action, order of actions or combination of actions, which led to this error). Once the cause is found, the ERM can associate to this error a specific error ID corresponding to its type (see Section 3.2.2). Thus, in the MDP-based system, when the ERM outputs e^*, the latter is a tuple containing two parameters: *error bool* and *error ID*. In this tuple, *error bool* is a Boolean which is $True$ if an error is detected and $False$ otherwise; *error ID* is the type of error detected if an error has been made. When outputted, e^* fills its corresponding slot in the prompt ω^*, before being sent to the next module in CogWatch: the Cue Selector. In the rest of this thesis (unless if the contrary is specified) e^* will only refer to *error bool*. As discussed in Section 3.2.2, the ERM implemented in the MDP-based Task Manager can detect six types of errors.

5.3 Training

We saw that contrary to the MDP-based APM, the ERM does not go through any learning process. A rule-based approach similar to the one applied by the ERM could have been applied to the APM. However, in order to make the system less dependent on clinicians and reduce the development complexity, the MDP-based APM functionality is based on a reinforcement learning technique in order to find the optimal policy. In CogWatch, the MDP-based APM learns how to find the optimal policy by being trained with the Monte Carlo Algorithm (Chapter 2).

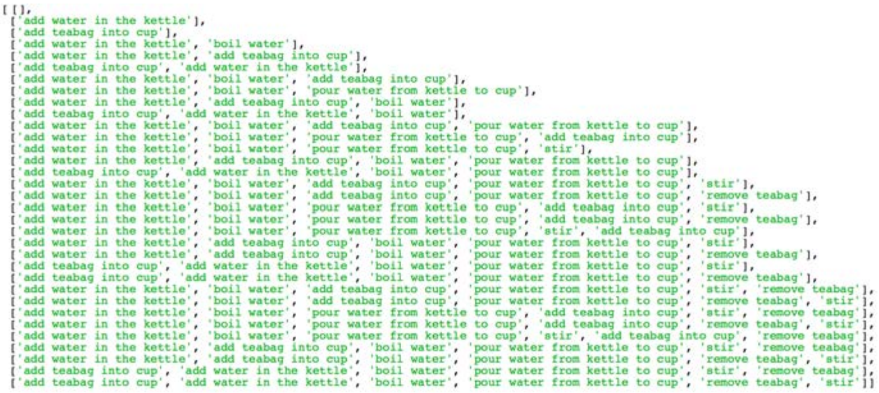
5.3.1 MDP framework adaptation to CogWatch

As discussed in Section 2.3.1, the Markov Decision Process framework is defined by the tuple $\{$A, S, P, C$\}$. Taking into account CogWatch specifications, these parameters are adapted as follows:

- **Action Space** A: The action space is a finite set of recommendations μ that the APM can output. These recommendations directly correspond to the user's actions the ARS can detect. Thus $A = \{$"Fill kettle", "Boil water", "Pour kettle", "Add teabag", "Add sugar", "Add milk", "Stir", "Remove teabag", "Pour cold water from jug to cup", "Toying with boiling water"$\}$, and $\mu \in A$.

- **State Space** S: As discussed in Section 5.2.2, the MDP state space is a finite set of all the states that are considered as "valid". In other words, each state s, with $s \in S$ is a "valid" sequence of actions, and S correspond to the *Summary space* (see Figure 5.3).
  With the state restriction technique, the size of the S is 33 for "Black tea", 205 for "Black tea with sugar" and 1539 for "White tea with sugar". The rapid growth of the state space can be explained by the variability in task completion that still exists even with the state restriction technique. For example, in the case of "White tea with sugar" 8 actions are considered to be mandatory for the task to be completed (i.e., "Fill kettle", "Boil water", "Pour kettle", "Add teabag", "Add sugar", "Add milk", "Stir", "Remove teabag"). Even when taking into account the state restriction technique, these actions can be arranged through many valid combinations. Indeed, an action like "Add sugar" can be performed at any point in a sequence, and S will take into account all possible "valid" combinations.
  For a better understanding, Figure 5.4 shows the 33 states composing the state space for "Black tea". In the case of this specific task, it is for each of these states that the MDP-based APM needs to find an optimal policy.

- **Transition Probability** P: The transition probability $P(s' \mid s, \mu)$ corresponds to the probability for the Task Manager to move from state s to s' when the recommendation μ is selected, see Section 2.3.1.4.



```
[[],
 ['add water in the kettle'],
 ['add teabag into cup'],
 ['add water in the kettle', 'boil water'],
 ['add water in the kettle', 'add teabag into cup'],
 ['add teabag into cup', 'add water in the kettle'],
 ['add water in the kettle', 'boil water', 'add teabag into cup'],
 ['add water in the kettle', 'boil water', 'pour water from kettle to cup'],
 ['add water in the kettle', 'add teabag into cup', 'boil water'],
 ['add teabag into cup', 'add water in the kettle', 'boil water'],
 ['add water in the kettle', 'boil water', 'add teabag into cup', 'pour water from kettle to cup'],
 ['add water in the kettle', 'boil water', 'pour water from kettle to cup', 'add teabag into cup'],
 ['add water in the kettle', 'boil water', 'pour water from kettle to cup', 'stir'],
 ['add water in the kettle', 'add teabag into cup', 'boil water', 'pour water from kettle to cup'],
 ['add teabag into cup', 'add water in the kettle', 'boil water', 'pour water from kettle to cup'],
 ['add water in the kettle', 'boil water', 'add teabag into cup', 'pour water from kettle to cup', 'stir'],
 ['add water in the kettle', 'boil water', 'add teabag into cup', 'pour water from kettle to cup', 'remove teabag'],
 ['add water in the kettle', 'boil water', 'pour water from kettle to cup', 'add teabag into cup', 'stir'],
 ['add water in the kettle', 'boil water', 'pour water from kettle to cup', 'add teabag into cup', 'remove teabag'],
 ['add water in the kettle', 'boil water', 'pour water from kettle to cup', 'stir', 'add teabag into cup'],
 ['add water in the kettle', 'add teabag into cup', 'boil water', 'pour water from kettle to cup', 'stir'],
 ['add water in the kettle', 'add teabag into cup', 'boil water', 'pour water from kettle to cup', 'remove teabag'],
 ['add teabag into cup', 'add water in the kettle', 'boil water', 'pour water from kettle to cup', 'stir'],
 ['add teabag into cup', 'add water in the kettle', 'boil water', 'pour water from kettle to cup', 'remove teabag'],
 ['add water in the kettle', 'boil water', 'add teabag into cup', 'pour water from kettle to cup', 'stir', 'remove teabag'],
 ['add water in the kettle', 'boil water', 'add teabag into cup', 'pour water from kettle to cup', 'remove teabag', 'stir'],
 ['add water in the kettle', 'boil water', 'pour water from kettle to cup', 'add teabag into cup', 'stir', 'remove teabag'],
 ['add water in the kettle', 'boil water', 'pour water from kettle to cup', 'add teabag into cup', 'remove teabag', 'stir'],
 ['add water in the kettle', 'boil water', 'pour water from kettle to cup', 'stir', 'add teabag into cup', 'remove teabag'],
 ['add water in the kettle', 'add teabag into cup', 'boil water', 'pour water from kettle to cup', 'stir', 'remove teabag'],
 ['add water in the kettle', 'add teabag into cup', 'boil water', 'pour water from kettle to cup', 'remove teabag', 'stir'],
 ['add teabag into cup', 'add water in the kettle', 'boil water', 'pour water from kettle to cup', 'stir', 'remove teabag'],
 ['add teabag into cup', 'add water in the kettle', 'boil water', 'pour water from kettle to cup', 'remove teabag', 'stir']]
```

Figure 5.4: Representation of the MDP state space for Black tea.

- **Cost Function** C: The cost $C(s, \mu)$ corresponds to the cost incurred when taking recommendation μ in state s. The cost function is defined as follows. If the new state obtained after adding μ to state s is:
  - "valid", then the immediate cost incurred is 1.
  - not "valid", then the immediate cost incurred is 1000.
  - complete and "valid", which means that the task has been correctly completed, then the cost immediately incurred is -100.

Once the MDP framework is adapted to CogWatch, it needs to be solved so the MDP-based APM can find the optimal recommendations to send to the user for each state in S.

5.3.2 APM Policy optimization

Policy optimization is performed using a Monte Carlo Algorithm defined in Section 2.3.1.2. During training, the algorithm automatically generates many trials through which the MDP-based APM learns how to act in an optimal way based on the state it is in. First, a specific task for which the MDP-based APM needs to be trained is chosen. Once this choice is made, the corresponding MDP state space is selected by the MDP-based APM and used during learning. At the beginning of each trial, a Q matrix is initialized with the first guess of the expected cost for taking each recommendation μ from the action space A in each state s from the selected state space S. At the end of each trial the total cost is calculated for each pair $\langle s, \mu \rangle$ visited. The corresponding Q values are then updated taking into account the accumulated costs. The whole process is then repeated until convergence is reached. When convergence is reached for one specific task, another task (i.e., another type of tea) is selected and the MDP-based APM goes through another process of learning. At the end of training, the MDP-based APM saves for each type of tea a mapping between each state s from the corresponding state space and an optimal recommendation μ^*.

5.4 Evaluation

Once the optimal policy is obtained, it is important to evaluate the performance of the system. This section will explain how the MDP-based Task Manager was evaluated with real participants interacting with the system, and with simulated users implemented in a virtual environment (see Chapter 4).

5.4.1 Trials with stroke survivors

When real users interacted with the system, the assumption of full observability from the MDP-based Task Manager was based on the fact that a clinician was allowed to correct the ARS outputs at all times. In other words, if during a trial the ARS output an observation that did not correspond to the action made by the user, the observation could be corrected. In such a case, the Task Manager always had access to the true user's behavior.

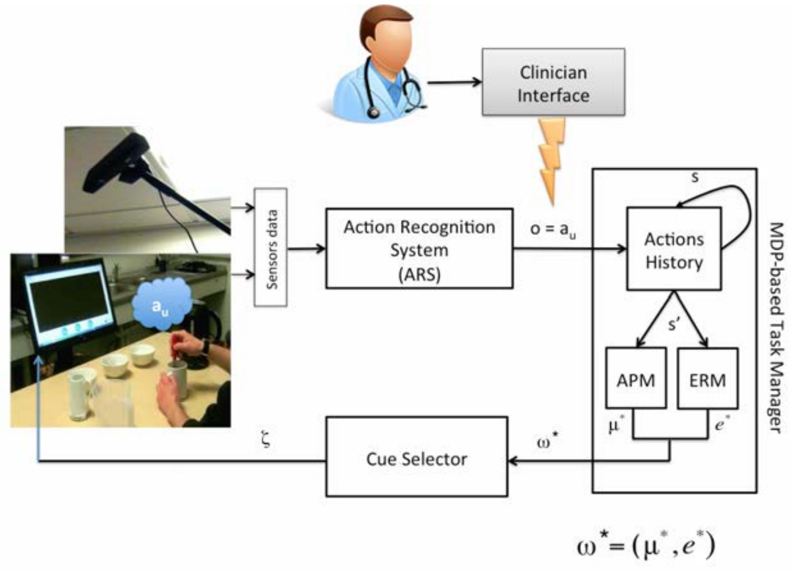


Figure 5.5: Structure of CogWatch with the MDP-based Task Manager during evaluation with stroke survivors.

Taking into account this fact, the architecture of CogWatch with the MDP-based Task Manager can be updated as in Figure 5.5. The fundamental structure of CogWatch remains the same as the one described in Figure 5.1. However, we can see that the observation o of the ARS is verified and can be corrected by a clinician via an interface. Hence, the ARS observation always corresponded to the action a_u performed by the user.

To measure the performance of CogWatch with stroke survivors, two experiments of 96 trials were performed by 12 stroke survivors (i.e., 6 women, 6 men, aged between 53 and 82). They went through 24 trials of each type of tea: "Black tea with sugar", "White tea with sugar", "White tea", "Black tea" in random order. During one experiment, the users did not receive any guidance at all, while in the other experiment the same users could receive assistance from the system. In both experiments, they simply received the instruction to make a specific type of tea. Note that during all trials where users could interact with CogWatch, they were free to follow or not the system's recommendations.

Both experiments used a randomized cross over design. Users were randomly allocated to the order in which they had access to the Cogwatch system or when they had to go through the tasks by themselves. All users were given the time to be introduced to the system (i.e., 5 to 10 minutes), and received explanations about how to interact with the patient's screen (see Chapter 3). During this introduction phase, examples of cues were shown to them, and they had the possibility to choose which ones they preferred the most. They also had the possibility to try the system themselves before starting the experiment with CogWatch. Note that a clinician or an experimenter was always in the room in case the user had questions.

To measure the performance of the MDP-based Task Manager implemented in CogWatch, the number of Non Fatal Errors (NFE) and Fatal Errors (FE) made by the users when assisted by CogWatch (CW) were compared with its equivalent when the users were performing the tasks without having access to the system's recommendations. Note that CogWatch clinicians defined NFE as recoverable errors; in other words, when such errors occur the task can be continued and potentially correctly completed by the user. By contrast, FE are non-recoverable errors and force the task to be aborted; for example, they can occur when the user's safety is at risk.

Recall that the Cue Selector acts like a filter. At each step, it receives a prompt (from the Task Manager), takes into account the number of errors detected by the Task Manager, and decides by itself when to transform each prompt into a cue that will be shared with the user. Hence, an Errorfull technique was applied (see Section 1.1) during the evaluation with stroke survivors, and the number of errors detected by the Task Manager did not necessarily correspond to the number of cues received by the user.

5.4.2 Trials with simulated users

The evaluation performed via user simulation was done by implementing $SimU\alpha$ and $SimU\beta$ respectively described in Section 4.2.1 and Section 4.3. These simulated users were configured to interact with a Virtualization of the whole CogWatch system depicted in Figure 5.6. The structure of this virtualization was described in Section 4.4. In Figure 5.6, we see that the MDP-based Task Manager was specifically implemented in the virtualization of CogWatch in order to be evaluated.
During evaluation, the $SimUs$ were used to analyze the impact of the Task Manager's best next recommendation μ^* on the $SimUs$' success rate without the effect of the Cue Selector. In other words, we virtually run trials following an Errorless technique. (Note that during evaluations using user simulation, contrary to the evaluations run with real participants, no clinician could intervene to adjust the ARS observations.)

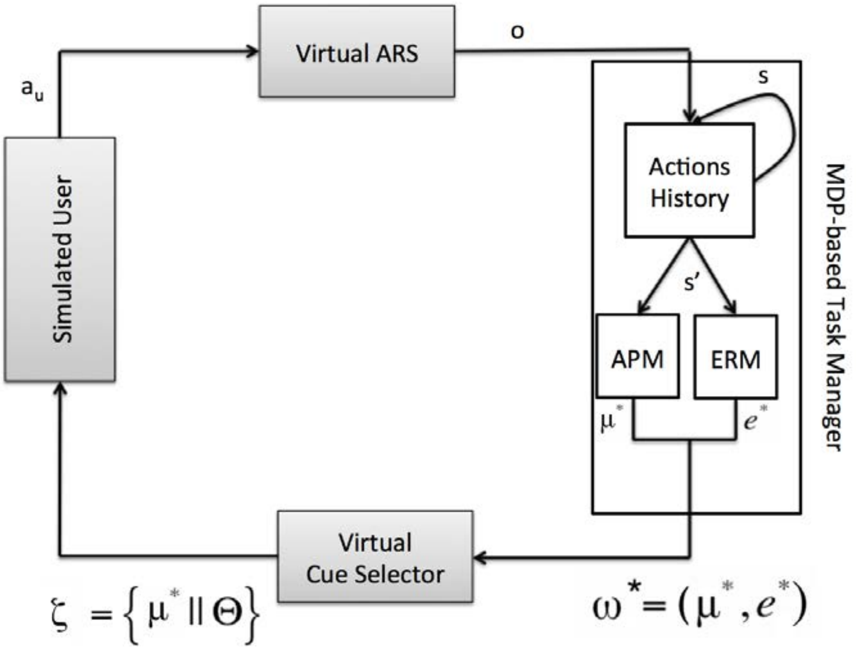


Figure 5.6: Diagram of the system used to evaluate the MDP-based Task Manager

Two scenarios were run during evaluation. In the first scenario,

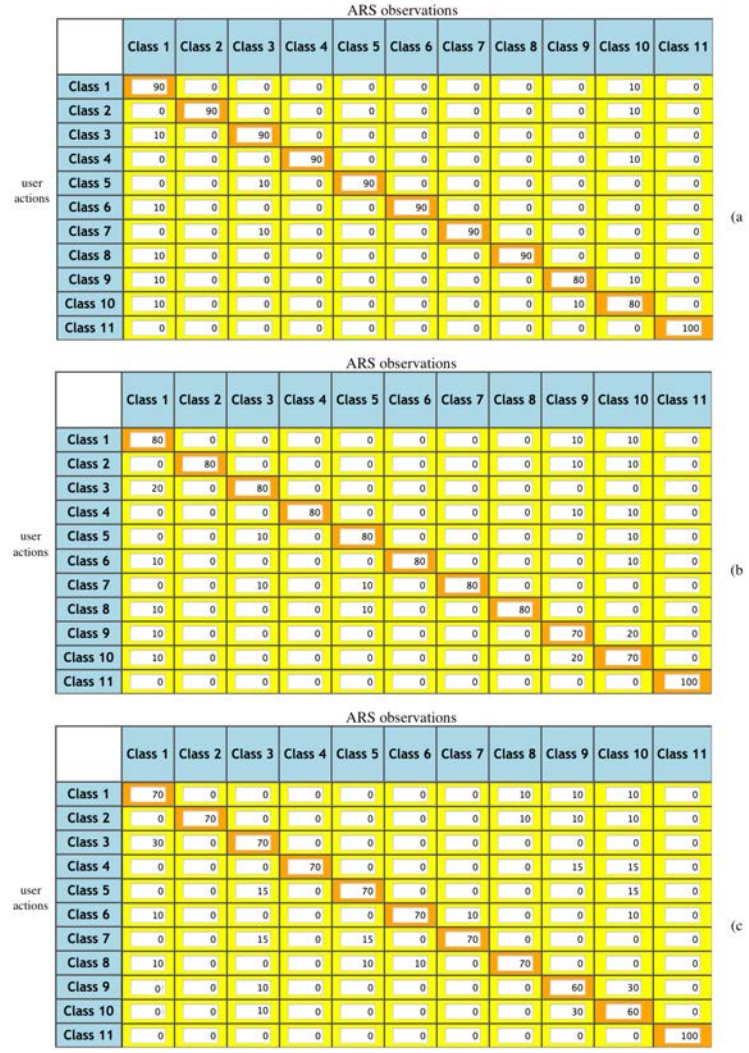
- The virtual ARS error rate was configured to be 0%. In other words, the ARS was perfect,
- The virtual Cue Selector was configured to always let the Task Manager's prompts reach *SimU*α,
- *SimU*α was used to performed 300 trials of "Black tea", "Black tea with sugar" and "White tea with sugar" at varying level of compliance; respectively 100%, 50%, 20% and 0% compliance.

The notion of *SimU*'s compliance toward the Task Manager's prompts was explained in Chapter 4 Section 4.2.1

In the second scenario,

- The virtual Cue Selector was set up to always let the Task Manager's prompts reach *SimU*α and *SimU*β,
- The level of compliance of *SimU*α was 100%,
- *SimU*β was configured to display the behavior explained in Chapter 4 Section 4.3,
- The *SimU*α and *SimU*β were used to performed 300 trials of "Black tea", "Black tea with sugar" and "White tea with sugar" at varying ARS error rate, respectively 0%, 10%, 20% and 30%.

As discussed in Chapter 4, the virtual ARS is modelled as a confusion matrix. The confusion matrices used in the second scenario can be seen in Figure 5.7. Note that the error rate and confusion matrix of the real CogWatch ARS are unknown. Thus, the confusion matrices of the virtual ARS are randomly chosen.



ARS observations

user actions

(a

	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Class 10	Class 11
Class 1	90	0	0	0	0	0	0	0	0	10	0
Class 2	0	90	0	0	0	0	0	0	0	10	0
Class 3	10	0	90	0	0	0	0	0	0	0	0
Class 4	0	0	0	90	0	0	0	0	0	10	0
Class 5	0	0	10	0	90	0	0	0	0	0	0
Class 6	10	0	0	0	0	90	0	0	0	0	0
Class 7	0	0	10	0	0	0	90	0	0	0	0
Class 8	10	0	0	0	0	0	0	90	0	0	0
Class 9	10	0	0	0	0	0	0	0	80	10	0
Class 10	10	0	0	0	0	0	0	0	10	80	0
Class 11	0	0	0	0	0	0	0	0	0	0	100

ARS observations

user actions

(b

	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Class 10	Class 11
Class 1	80	0	0	0	0	0	0	0	10	10	0
Class 2	0	80	0	0	0	0	0	0	10	10	0
Class 3	20	0	80	0	0	0	0	0	0	0	0
Class 4	0	0	0	80	0	0	0	0	10	10	0
Class 5	0	0	10	0	80	0	0	0	0	10	0
Class 6	10	0	0	0	0	80	0	0	0	10	0
Class 7	0	0	10	0	10	0	80	0	0	0	0
Class 8	10	0	0	0	10	0	0	80	0	0	0
Class 9	10	0	0	0	0	0	0	0	70	20	0
Class 10	10	0	0	0	0	0	0	0	20	70	0
Class 11	0	0	0	0	0	0	0	0	0	0	100

ARS observations

user actions

(c

	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Class 10	Class 11
Class 1	70	0	0	0	0	0	0	10	10	10	0
Class 2	0	70	0	0	0	0	0	10	10	10	0
Class 3	30	0	70	0	0	0	0	0	0	0	0
Class 4	0	0	0	70	0	0	0	0	15	15	0
Class 5	0	0	15	0	70	0	0	0	0	15	0
Class 6	10	0	0	0	0	70	10	0	0	10	0
Class 7	0	0	15	0	15	0	70	0	0	0	0
Class 8	10	0	0	0	10	10	0	70	0	0	0
Class 9	0	0	10	0	0	0	0	0	60	30	0
Class 10	0	0	10	0	0	0	0	0	30	60	0
Class 11	0	0	0	0	0	0	0	0	0	0	100

Figure 5.7: Virtual ARS confusion matrices. (a - ARS error rate 10%, (b - ARS error rate 20%, (c - ARS error rate 30%. Class 1: "Fill kettle", Class 2: "Boil water", Class 3: "Add teabag", Class 4: "Pour kettle", Class 5:"Add sugar", Class 6:"Add milk", Class 7: "Stir", Class 8:"Remove teabag", Class 9:"Pour water from jug to cup", Class 10:"Toying with boiling water", Class 11: ∅ (no action performed or detected).

5.5 Results

This section reports the results obtained when comparing real users' performance with and without the assistance of the MDP-based Task Manager implemented in CogWatch. It also shows the impact of the ARS error rate and level of compliance of the simulated user on the ability of the latter to succeed its tasks.

5.5.1 Results obtained with stroke survivors

In Table 5.5.1, one can see that when interacting with CogWatch, stroke survivors made fewer Fatal and Non Fatal Errors, and succeeded to complete the tasks more often than when they did not have access to the system's prompts. Indeed, stroke survivors' success rate with CogWatch was 95.8%, and only 67.7% without. In Figure 5.8, one can see that the use of CogWatch significantly reduced the occurrence of some types of errors, such as omission, sequence and quantity errors. In other words, when stroke survivors were guided by the system, they forgot mandatory actions less often (for example adding the teabag in the cup); they made fewer sequential errors (for example trying to switch the kettle on when the latter is empty); and they did not misjudge the amount of water to put in the kettle or in the cup (i.e., quantity error). Figure 5.9 shows the total number of errors (i.e., $NFE + FE$) made by each stroke survivor over the 96 trials. One can see that the impact of CogWatch was very beneficial to those who had difficulties performing the tasks alone (i.e., Pt19, Pt11, Pt07, Pt10, Pt16, Pt14, Pt15). By contrast, stroke survivors who had no difficulty performing the task by themselves made more errors when interacting with the system (i.e., Pt06, Pt13, Pt09, Pt08). This could mean that stroke survivors who have a higher probability of succeeding the task by themselves, and rarely need the system's help, might be disturbed by some of the system's recommendations.

	NFE	FE	Successes
With CW	73	4	92
Without CW	90	31	65

Table 5.1: Users' number of Non Fatal Errors (NFE), Fatal Errors (FE) and successes over 96 trials.

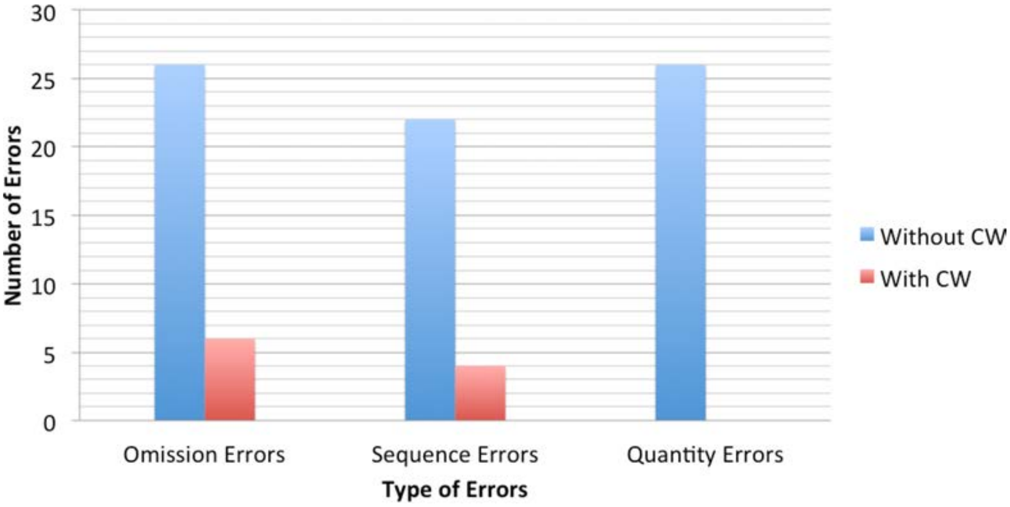


Figure 5.8: Number of specific types of errors made by users over 96 trials.

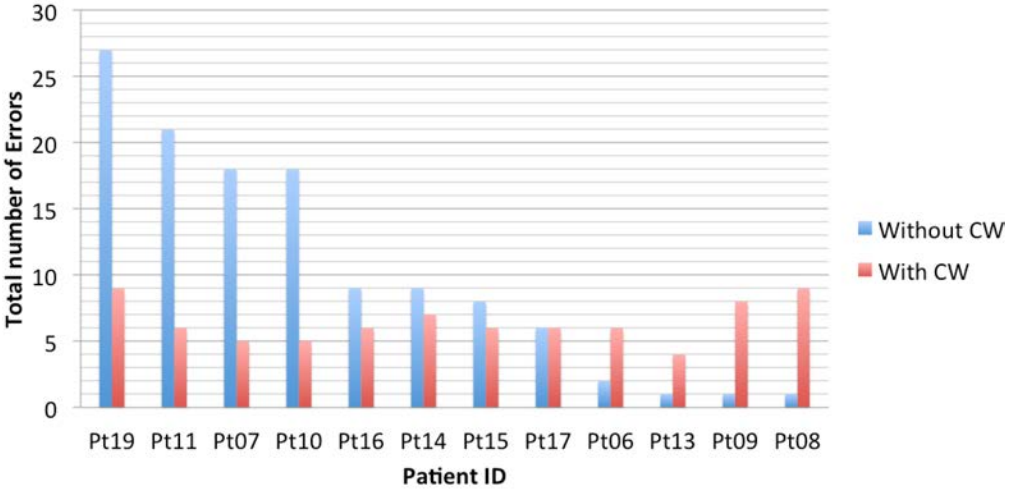


Figure 5.9: Total number of errors made by users over 96 trials.

Another reason that could explain why more errors are detected in this specific case could be related to the type of the errors made: perseveration errors related to the action of stirring (see E18 - E20 in Table 3.1). The action of "stirring" is difficult to define precisely (i.e., difficulty to define when it begins, when it stops, when it is repeated based on the movements made). Without CogWatch, the number of errors made by stroke survivors was calculated by clinicians, who may tend to be more flexible than the system as far as the action of stirring is concerned. For example, if a user continuously stirs, making circular movements without stopping, this could be interpreted by clinicians as one "stirring" action, while the system may consider that several "stirring" actions occurred because of the number of circular motions performed (i.e., even without interruption). In this case, it would mean that the system tend to count the number of "stirring" actions differently than clinicians, and potentially detect more perseveration errors than they do.

5.5.2 Results obtained via simulation

Following the first scenario described in Section 5.4.2, we obtained the results depicted in Figure 5.10. In this figure, one can see that $SimU\alpha$ success rate was higher when it was configured to follow the recommendations μ^* provided by the Task Manager at each step. During “Black tea” and “Black tea with sugar”, when the simulated user followed the system’s recommendations 100% of the time, its success rate was 100%.

During “White tea with sugar”, even if the recommendations provided by the Task Manager were always correct, $SimU\alpha$ success rate was 97%. This is due to the fact that the CogWatch system is an after-effect system, where an action has to be made by the user, then observed by the ARS, for the Task Manager to react to it and retrieve the appropriate prompt. If the first action of the user is seen as a Fatal Error, the system cannot help the user and the trial will be labeled as a failure from start. More specifically, during this particular task, the simulated user began few trials by selecting the action “Boil water”, which is considered as a Fatal Error. Indeed, in Table 3.1, one can see that this behavior is defined as

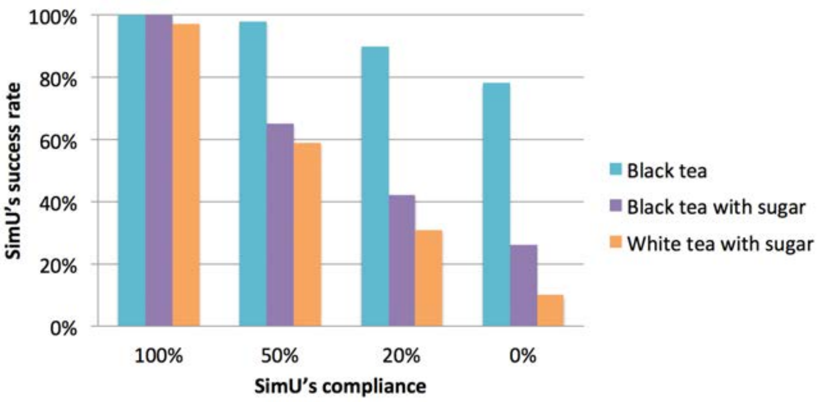


Figure 5.10: $SimU\alpha$ success rate at varying levels of compliance during “Black tea”, “Black tea with sugar” and “White tea with sugar”. Virtual ARS error rate: 0%.

erroneous (i.e., Error ID: E27). This erroneous behavior was part of the different behaviors seen in real participants data that $SimU\alpha$ emulates.

Figure 5.10 also shows that when $SimU\alpha$ level of compliance decreased, its ability to succeed the tasks also decreased. The case where $SimU\alpha$ compliance was 0% corresponds to the case where it did not receive any guidance from the system and went through each trial by itself. One can see that $SimU\alpha$ had a high probability to fail the task in this case. For example, during "Black tea with sugar", $SimU\alpha$ succeeded the task only 21% of the time when performing the task by itself. Similarly to real participants who had difficulties in completing the task by themselves (see previous section), $SimU\alpha$ had a higher probability of succeeding all tasks when it received guidance from the system and complied to this guidance.

It is also possible to note that $SimU\alpha$ success rate decreased when the complexity of the task increased. When performing the task by itself (0% compliance), $SimU\alpha$ succeeded "Black tea" 78% of the time. During this task, it needed to find a correct combination of at least 6 mandatory actions. By contrast, in "White tea with sugar" where a correct combination of at least 8 actions needed to be found, $SimU\alpha$ succeeded the task only 10% of the time.

Following the second scenario explained in Section 5.4.2, we measured the ability of the MDP-based Task manager to correctly guide a $SimU\alpha$ complying 100% of the time with the system's recommendations, and $SimU\beta$ at varying ARS error rates.

Figures 5.11, 5.12 and 5.13 respectively show $SimU\alpha$ and $SimU\beta$'s success rate at varying ARS error rates, when performing "Black tea", "Black tea with sugar" and "White tea with sugar". One can see the impact of the increase of ARS error rate on the ability of the MDP-based APM to guide the simulated users during the tasks: The more the ARS error rate increases, the more it is difficult for the MDP-based APM to properly guide the simulated users.
Indeed, any divergence between the MDP-based APM representation of the user state and the real user state will decrease the ability of the system to output optimal prompts. Moreover, because the simulated users in this scenario have a high level of compliance (note that $SimU\alpha$ has a higher level of compliance than $SimU\beta$), they are configured to respect the recommendations output by the MDP-based APM. In other words, at high ARS error rate, the MDP-based APM has more chances to make the simulated users fail the task.

On each figure, the simulated users' success rate when constantly ignoring the prompts of the system was also represented (i.e., 0% compliance). It shows what the simulated users were capable of by themselves - how they performed the tasks when they did not receive any assistance from the system. As such, it is not dependent on the ARS error rate, and is a good indication of a threshold to measure the usefulness of the system. For example, in Figure 5.11, one can see that around 12% ARS error rate, $SimU\alpha$'s success rate when assisted by the system was lower than what it was capable of by itself. In other words, at this level of uncertainty, the MDP-based APM was not useful to $SimU\alpha$ as the latter was able to succeed the task more often without it.

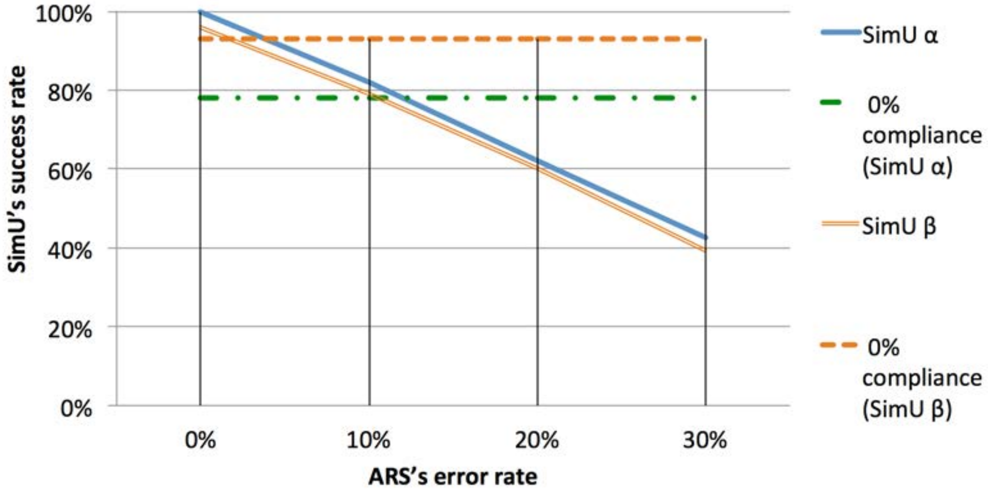


Figure 5.11: $SimU\alpha$ and $SimU\beta$'s success rate at varying ARS error rate - "Black tea".

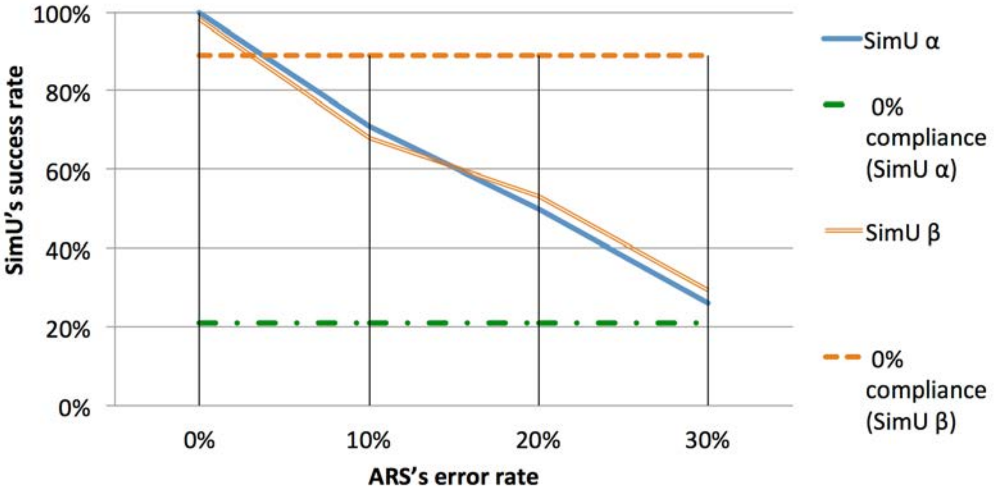


Figure 5.12: $SimU\alpha$ and $SimU\beta$'s success rate at varying ARS error rate - "Black tea with sugar".

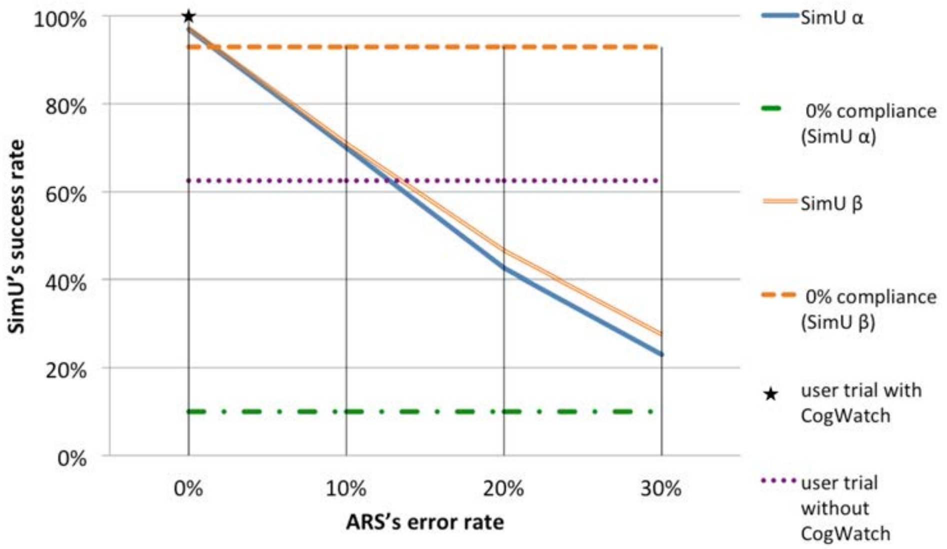


Figure 5.13: *SimUα* and *SimUβ*'s success rate at varying ARS error rate - "White tea with sugar".

As discussed in Chapter 4, *SimUα* and *SimUβ* display different behaviors, and are based on different user's database. On each figure, one can see that *SimUβ* had a higher success rate at 0% compliance than *SimUα*. This shows that the database used by *SimUβ* is composed of many correct user's examples of tasks completion, while *SimUα* behaves as a virtually limited user (due to its intrinsic limitations, see Section 4.2.1). In the case of "Black tea with sugar" (i.e., Figure 5.12), one can see that around 3% ARS error rate, the MDP-based Task Manager began to output recommendations that made *SimUβ* fail the task more often than what it was capable of by itself. By contrast, this phenomenon did not occur with *SimUα*, even at 30% ARS error rate. When comparing each figure with each other, it is also possible to see that the complexity of the task has an impact on the simulated users' success rate, when interacting with the system. For example, the Figures show that at high ARS error rate (for example, 20%, 30% ARS error rate) the MDP-based APM has a higher probability of misleading the simulated users during "White tea with sugar" than during "Black tea".

Indeed, at 30% ARS error rate, the simulated users' success rate is approximately 41% when performing "Black tea", and only 25.5% when performing "White tea with sugar". Globally, we can note that the more the ARS makes mistakes, the more the MDP-based Task Manager has an incorrect representation of the user's state, which leads to more incorrect prompts to be output. In other words, when the ARS error rate is high, the MDP-based Task Manager misguides users and make those who follows its recommendations fail more often.

Compared to Figures 5.11 and 5.12, Figure 5.13 also shows the data points related to the user trial (i.e., "White tea with sugar" task) described in Section 5.5.1. One can see that, in the specific case of "White tea with sugar", when stroke survivors had access to the system's prompts, they succeeded the task 100% of the time. As explained in Section 5.4.1, the system was assumed to be perfect in this context (i.e., ARS's error rate: 0%). In the other scenario, when stroke survivors went through the tasks without the system's recommendations, their success rate was 62.5% during "White tea with sugar". These results give a preliminary suggestion that similarly to $SimU\alpha$ and $SimU\beta$, at ARS's error rate 0%, stroke survivors had a higher probability to succeed the task when having access to the system's prompts, than when they went through the task by themselves.

However, in the experiments run with simulated users, the latter were used to go through 300 trials of "White tea with sugar" at each ARS error rate, following the APM's outputs 100% and 0% of the time. Stroke survivors only went through 24 trials, and when they had access to the system, they were free to follow or not its recommendations at any point during the trials. Moreover, a EF technique was applied when stroke survivors interacted with the system, while a EL technique was applied with simulated users. Thus, for a robust comparison between stroke survivors and simulated users' performance, more trials should be run with stroke survivors, in the same conditions simulated users were used. To be able to do so in the future, will give us the possibility to analyze the validity of the methodology used to design the simulated users implemented.

5.6 Summary

This section described in detail the MDP-based APM and the ERM, and how these modules allow the MDP-based Task Manager to provide prompts to users. The MDP framework of the APM and the rule-based technique used to design the ERM were fully described. Implementing a Monte Carlo Algorithm, the MDP-based APM was evaluated with stroke survivors and via user simulation. An attempt to compare stroke survivor and simulated users' performance was made: it was highlighted that in order to do so in a robust way more experiments needed to be run in the same conditions.

The results of the current experiments showed the ability of the system to be an efficient assistive system with stroke survivors when the MDP-based Task Manager has access to the user state. This was possible with a clinician correcting the inputs of the Task Manager if necessary. However, in a full automatic setting, when no clinician can intervene, the results highlighted the difficulties of the MDP-based Task Manager to cope with uncertainty. Each time the ARS makes a wrong observation, the MDP-based Task Manager will update a wrong representation of the user state, which at high ARS error rate will make the system tend to fail to fulfill its goals. This can be explained by the fact that the MDP cannot robustly model uncertainty. Hence, in the next section, a Partially Observable Markov Decision Process will be implemented in order to analyze how the POMDP is able to compensate for the MDP limitations.

Chapter 6

POMDP-based Task Manager

6.1 Introduction

This section gives an in-depth description of the POMDP-based Task Manager that was implemented in the CogWatch system. Section 6.2.2 shows how the MDP state space (i.e., *summary space*) can be used to represent the POMDP states called belief states. Section 6.3 then explains how training and evaluation were performed via user simulation at varying ARS error rate.

6.2 CogWatch POMDP-based Task Manager

Contrary to the MDP, in this Chapter we take into account the fact that the user's environment is only partially observable from the system's point of view. In such a case, the CogWatch POMDP-based Task Manager maintains a probability distribution over the user states, and infers what he or she should do next. Similarly to the MDP-based Task Manager, the POMDP-based Task Manager should also be able to detect whether the user makes errors or not during the tasks.

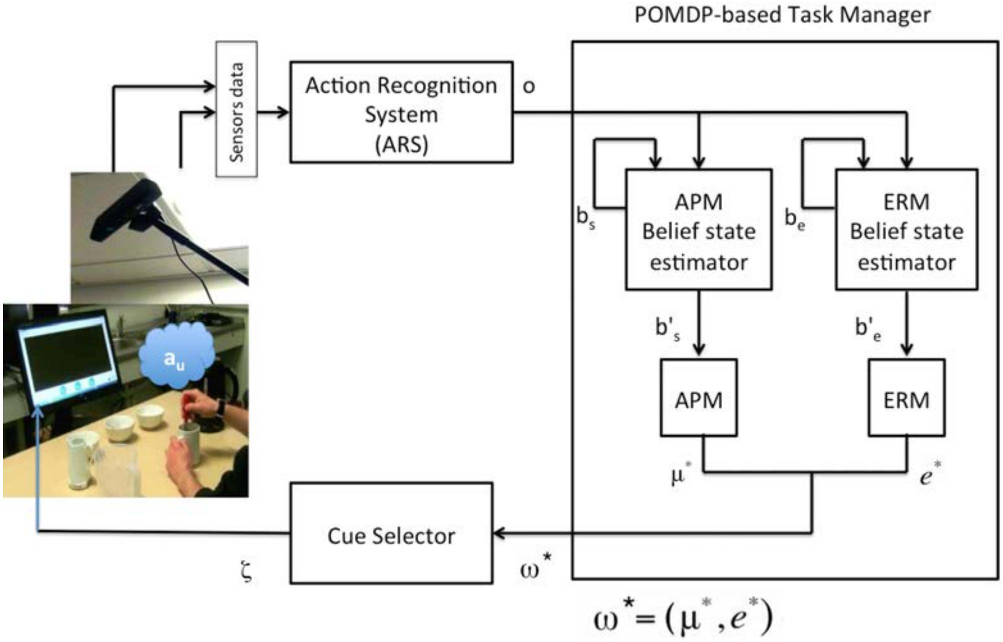


Figure 6.1: Structure of CogWatch with the POMDP-based Task Manager.

6.2.1 Task formalism

In Section 3.3, we described the general architecture of the CogWatch system and the Task Manager. In the case of the POMDP-based Task Manager, the State Modeler is replaced by a Belief State Estimator (i.e, one for the APM and another for the ERM) as shown in Figure 6.1. The system works as explained in Section 3.3. Nevertheless, once again, it is important to focus on the observation o output by the ARS and how it is processed by the POMDP-based Task Manager. Each time the user or Simulated User performs an action a_u, the ARS outputs an observation o based on its interpretation of what action has just been performed. This observation may be incorrect, and contrary to the MDP-based Task Manager, the POMDP-based Task Manager acknowledges this possibility.

Each observation o is used to update a belief state b_s that will be used by the APM, and a belief state b_e that will be used by the ERM (see Equation 2.21). Via the APM, the POMDP-based Task Manager uses b'_s in order to select the best next recommendation μ^* that the system considers the user should follow to successfully finish or continue a task. Via the ERM, the POMDP-based Task Manager uses b'_e to output its best understanding of whether the user has just made an error or not e^*. The outputs μ^* and e^* are then passed in the form of a prompt ω^* that is sent to the Cue Selector. The latter then decides when it is necessary to share the cue ζ with the user or Simulated User.

Note that with the MDP-based system, the ERM could also find what was the type of errors made when an error occurred. With the ERM used in the POMDP-based Task Manager, we will only focus on the ability of the latter to detect whether an error has been made or not.

6.2.2 Belief state representation

In [37], a POMDP approach was taken for action planning during ADL, and the POMDP belief state was defined as a probability distribution over variables such as users attitude (dementia level, responsiveness, awareness), and user's behavior. Here, we show how the belief state can directly be defined as a probability distribution over the underlying MDP state space.

6.2.2.1 APM perspective

The MDP state space S for "Black tea" contains 33 valid user states (see Section 5.3). Hence, the belief state b_s that the APM uses to plan the next best action the user should perform is a probability distribution over just 33 states. Following the same rule, the belief state b_s used by the APM is a probability distribution over 205 states for "Black tea with sugar" and 1539 states for "White tea with sugar". Each belief state b_s sums up to 1.

6.2.2.2 ERM perspective

In the case of the POMDP-based Task Manager, for the ERM to detect if the user makes a mistake during the task under uncertainty, we add an error state s_e to S, where s_e is an encapsulation of all user states that are not "valid" (see definition of state validity in Chapter 5). Let S_e be this new state space. Hence, the belief state b_e used by the ERM is a probability distribution over 34 states for "Black tea", 206 states for "Black tea with sugar" and 1540 states for "White tea with sugar". Each belief state b_e sums up to 1.

6.2.3 Belief update

In this implementation a factored POMDP was used, as discussed in Section 2.3.2.3. In such a case, the belief state's update equation is defined as in Equation 2.21.

6.3 Training and evaluation

To solve a POMDP, a policy ($\pi^* : B \rightarrow A$) needs to be found between a belief subspace B and the action space A, such that $\pi^*(b)$ minimizes the expected cost of task completion given belief state b.

For the POMDP-based APM, the states of the MDP are sequences of actions that can lead to successful task completion, where B_s is the belief subspace and A is the action space. For the ERM used in the POMDP-based Task Manager, an additional state s_e is added to the MDP state space, where B_e is the belief subspace and E is the error space, with $E = \{\text{True, False}\}$. In other words, when a belief state b_e is updated, the ERM can output $e^* = True$ if it considers that b_e captures an erroneous user's behavior, or $e^* = False$ it if considers that b_e captures a non erroneous user's behavior.

Both POMDP-based APM and ERM implemented in the POMDP-based Task Manager are trained offline with a simulated user (see Section 4.2.2). As far as the APM is concerned, a Monte Carlo (MC) Algorithm [104] is used for policy optimization (see Section 2.3.2.2). The ERM uses the algorithm given in Algorithm 3, to learn which label e^* (erroneous or not erroneous) to associate to each belief state b_e in B_e. Note that contrary to μ^*, e^* are not recommendations that a user can follow in order to change his or her state. They are indications of whether the system considers that an error was made by the user or not.

This section explains the framework on which is based the POMDP-based APM and ERM's training; gives a general description of the optimization algorithm used, an overview of the POMDP-based Task Manager operations, and the details related to the evaluations run to verify the performance of the POMDP-based system.

6.3.1 POMDP-based APM

The POMDP-based Task Manager implements a POMDP-based APM. Similarly to the MDP-based APM, its aim is to learn how to select which optimal actions the user should follow to successfully continue or complete a task. In this subsection only, the belief state b_s updated by the POMDP-based APM, and the belief state space B_s will respectively be referred to as b and B for simplification purpose.

6.3.1.1 POMDP framework adaptation to CogWatch

Formally, a POMDP is a tuple $\{S, A, P, C, O, Z\}$ [52], where the tuple $\{S, A, P\}$ corresponds to the MDP as defined in Section 5.3, and Z is the observation probability explained in Section 2.3.2. The other parameters are adapted to CogWatch as follows.

The Cost Function C: $C(b, \mu)$ is the cost for selecting recommendation μ when in belief state b. As we consider that b is a probability distribution over the MDP states, we define $C(b, \mu)$ such as:

$$C(b, \mu) = \sum_{s \in S} C(s, \mu) b(s), \tag{6.1}$$

where $C(s, \mu)$ is the cost for selecting recommendation μ when in state s, as explained in Section 5.3.

The Observation Space O: In CogWatch, the Observation Space is defined such as $O = A$. As discussed in Chapter 3 Section 3.3.2, the set of observations the Task Manager can receive from the ARS is such that $O = \{$"Fill kettle", "Boil water", "Pour kettle", "Add teabag", "Add sugar", "Add milk", "Stir", "Remove teabag", "Pour cold water from jug to cup", "Toying with boiling water"$\}$.

6.3.1.2 APM Policy optimization

During training, a belief-state-action value function $Q(b, \mu)$ records the immediate cost for selecting the recommendation (i.e., action the user should follow) μ in belief state b, for all belief states contained in the belief subspace B. Q is defined as the expected cost of a trial starting in belief state b, the POMDP-based APM selecting recommendation μ, and thereafter proceeding according to the current policy π until the trial ends and a final belief state is reached. The process is repeated until convergence is reached.

At the beginning, the initial policy is guessed. The belief subspace B is initialized and contains a chosen number of belief states b. In this work, when the task is chosen (i.e., when the type of tea is chosen), B is initialized with the belief states related to the MDP states contained in the corresponding MDP state space. This initialization is inspired by the method implemented by Brafman in [17], where the initial grid is composed of all perfect information belief states, which correspond to the underlying MDP.
For example, if the task chosen is "Black tea", the corresponding MDP state space size is 33, as explained in Section 5.3 and B is initially populated with 33 belief states. Each belief state has a probability one to be in a specific MDP state s and probability zero for all other MDP states. In other words, at the initialization, the APM belief subspace contains belief states such as:

- **1**: [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
- **2**: [0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
- ...
- **33**: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]

Each of those initial belief states is then associated to their respective optimal policy found when solving the MDP (see Chapter 5). In CogWatch, the POMDP-based APM finds a mapping from belief states $b \in B$ to recommendations $\mu \in A$ by implementing the Monte Carlo algorithm explained in Chapter 2, Figure 2.8.

6.3.2 ERM for POMDP-based Task Manager

In the POMDP-based Task Manager, the goal of the ERM is to find an optimal policy ε^* ($\varepsilon^* : B_e \rightarrow E$), which is a mapping from each belief state $b_e \in B_e$ to labels $e \in E$. In [47] an attempt to use a modified Monte Carlo Algorithm to find such a policy was implemented but was not successful. Indeed, in this previous work, the ERM used in the POMDP-based system did not succeed to outperform the one used in the MDP-based system under uncertainty.
In order to explore another technique which could allow the ERM to find policy ε^*, another algorithm is proposed in this thesis (Algorithm 3). Contrary to the MC algorithm used by the APM (see Figure 2.8), in the Algorithm given in 3, the knowledge of the true user's state is used to calculate the cost $C(b_e, e)$ for considering that a belief state b_e is erroneous or not. In this thesis, the cost $C(b_e, e)$ is defined such that:

$$C(b_e, e) = 1000 \; , \tag{6.2}$$

for all $b_e \in B_e$ and $e \in E$. Note that for simplification purposes, in this subsection only, b_e and B_e will respectively denoted as b and B.

Similarly to the MC algorithm, a *SimU* is implemented to interact with a virtualization of the system (Chapter 4) during training. The Algorithm works as follows. First, the assumption is made that the *SimU*'s initial state s_0 is empty (i.e., it has not performed any action yet), and that the initial belief state is b_0, with $b_0 = b(s_0) = 1$. During each virtual trial, the *SimU* selects different actions that are passed to the virtual ARS. Each time an action is received by the ARS, the latter outputs an observation, which allows the belief state b to be updated.

Each time the belief state is updated, the algorithm records the current true state s of the $SimU$ and makes it observable in the rest of the process. Following the current policy, the belief state b that has just been updated is labeled with e, which can be $True$ or $False$. The algorithm then verifies whether the user state is truly erroneous or not. If the ERM suggested the correct label for the current belief state, it receives a negative cost. If its suggestion was not correct compared to the true user's state, it receives a positive cost. After each suggestion, the corresponding $Q(b, e)$ is updated. In this case, $Q(b, e)$ is defined as the cost of a trial starting in belief state b, the ERM selecting output e, and thereafter proceeding according to the current policy ε until the trial ends and a final belief state is reached. At the end of each trial, the current policy is updated, considering that the label incurring the lowest cost for each belief state is the optimal one. The whole process is repeated until convergence. Here, we can consider that convergence occurs when the optimal labels allocated to each belief state remain unchanged during a large number of iterations.

6.3.3 Training at varying ARS error rates

The training of a POMDP-based system can be executed at varying ARS error rates [115]. In this work, the APM is trained around 10% and 20% ARS error rate (see Figure 7.10). The ERM is trained at 10%, 20% and 30% ARS error rate (see Figure 5.7). During training of the APM, $SimU\alpha$ was implemented to interact with the virtualization of CogWatch (see Section 4.2.1). It was configured to select its actions by itself, complying to the APM's outputs only when the virtual Cue Selector randomly accepts to let these outputs pass.

The ERM was trained twice. During its first training, $SimU\alpha$ was used and had the possibility to randomly decide to follow pre-defined sequences of actions. During its second training, $SimU\alpha$ was configured to comply with the system's recommendations 0% of the time and had to follow its intrinsic behavior in order to select actions. The results of both trainings were recorded separately.

Algorithm 3 Algorithm for ERM

```
Inputs:
$E$: set of ERM's labels $e$
$S$: set of (valid) user's states $s$
$Q(b, e)$ : cost for selecting $e$ when in $b$
$B$ : initial set of belief states $b$
$\varepsilon$ : initial policy, with $\varepsilon : B \to E$
repeat
  $j \leftarrow 0$
  $b \leftarrow b_0$
  Generate a tea trial with the Simulated User:
  for Each observation output by the ARS do
    $j \leftarrow j + 1$
    Update the belief state $b_j$.
    Record the current state $s_j$ of the Simulated User.
    $e_j \leftarrow \varepsilon(b_j)$ or $\varepsilon(b_n)$
    with $b_n \leftarrow$ nearest neighbor of $b_j$ in $B$.
    if $e_j \leftarrow False$ then

      if $s$ is in $S$ then
        $Q(b_j, e_j) \leftarrow Q(b_j, e_j)$ - $C(b_j, e_j)$
      else
        $Q(b_j, e_j) \leftarrow Q(b_j, e_j) + C(b_j, e_j)$
      end if
    else

      if $s$ is in $S$ then
        $Q(b_j, e_j) \leftarrow Q(b_j, e_j) + C(b_j, e_j)$
      else
        $Q(b_j, e_j) \leftarrow Q(b_j, e_j)$ - $C(b_j, e_j)$
      end if
    end if
    $J \leftarrow j$
  end for
  for $r \in [0, J]$ do
    Update the policy
    $\varepsilon(b_r) \leftarrow \arg\min_e Q(b_r, e), \forall b_r \in B, \forall e \in E$.
  end for
until converged
```

6.3.4 Evaluation

Once the policies $\pi^* : B_s \rightarrow A$ and $\varepsilon^* : B_e \rightarrow E$ are obtained, the performance of the POMDP-based Task Manager can be evaluated.
One approach to evaluation is to let real users interact with the system, as it has been done with the MDP-based Task Manager (see Chapter 5). In the case of the POMDP-based system, the APM and ERM were evaluated via simulation (see Chapter 4). This enables a wider coverage of user space, and the possibility to exhaustively analyze the main parameters that have an impact on the POMDP-based Task Manager's ability to fulfill its goals.

In this section, we will explain how the APM and ERM were evaluated separately, and how they were evaluated jointly.

6.3.4.1 Trials with simulated users

The evaluation performed via user simulation was done by implementing $SimU\alpha$ and $SimU\beta$ respectively described in Section 4.2.2 and Section 4.3; the latter being implemented to interact with a Virtualization of the whole CogWatch system (see Figure 6.2). The structure of this virtualization was described in Section 4.4. This structure is similar to the one where the MDP-based Task Manager was implemented (Figure 5.6). In Figure 6.2 we see that the POMDP-based Task Manager is implemented instead of the MDP-based Task Manager. Note that similarly to the evaluations run via simulation with the MDP-based system (Section 5.4.2), no clinician could intervene to adjust the ARS observations.

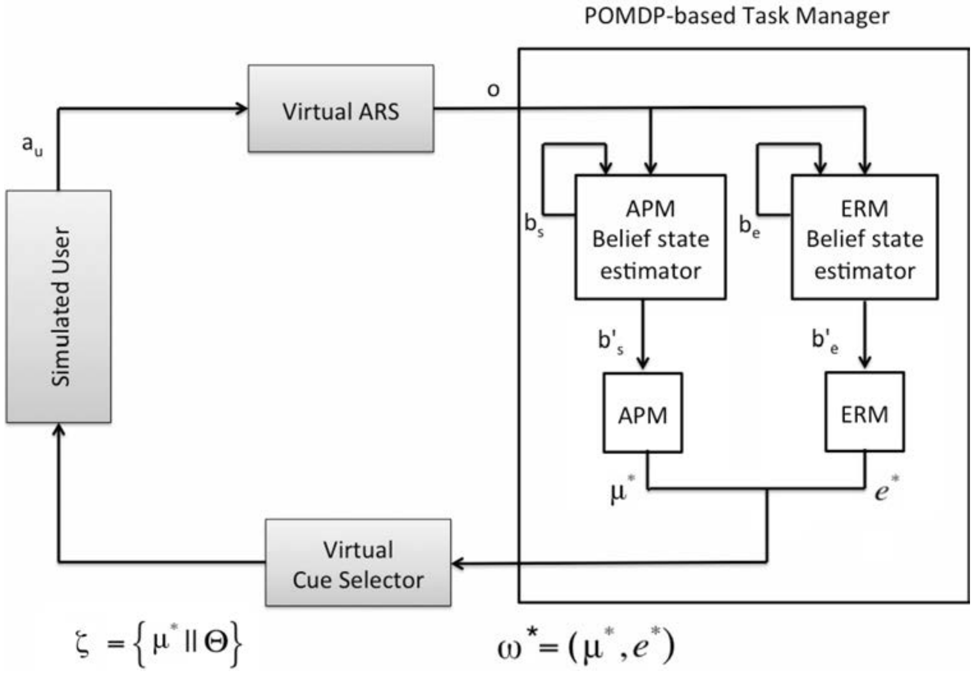


Figure 6.2: Diagram of the system used to evaluate the POMDP-based Task Manager

Evaluation of the APM In order to evaluate the APM only, we analyzed the impact of its best next recommendations μ^* on the $SimUs$' success rate, at varying ARS error rates, and without the effect of the Cue Selector (i.e., Errorless technique). Both $SimU\alpha$ and $SimU\beta$ were implemented to run the experiments. Here, $SimU\alpha$ was configured to display another behavior than the simulated user implemented for training: During evaluation, $SimU\alpha$ was configured to follow the APM's outputs 100% of the time.
Two scenarios were run.

- In the first scenario, the assumption was made that the virtual ARS confusion matrices were perfectly observable from the POMDP-based APM point of view. The confusion matrices used in this scenario can be seen in Figure 5.7.

- In the second scenario, we considered that the POMDP-based APM had access to an approximation of the virtual ARS confusion matrix. Since the latter is taken into account into the belief update equation (Equation 2.21), the aim was to investigate the potential impact of the imperfect perception of different ARS confusion matrices, on the performance of the POMDP-based APM.

Evaluation of the ERM In order to evaluate the ERM only, the $SimU\alpha$ implemented for the APM's evaluation was reused. We analyzed the ability of the ERM to correctly detect when the simulated user made errors or not at 30% ARS error rate during "Black tea" (Figure 5.7). We also investigated the potential impact of the simulated user's behavior implemented during training on the ERM's performance during evaluation. The ERM trained following Algorithm 3 and implemented in the POMDP-based system was also compared with the performance of the ERM implemented in the MDP-based system.

Joint evaluation In the joint evaluation, the $SimU\alpha$ implemented for the APM's evaluation is reused. The virtualization of the system was configured to only send a prompt to the simulated user when the ERM detected an error. When a prompt was passed to the simulated user, the latter complied with the recommendation suggested by the APM. When no prompt was received, the simulated user automatically chose its next action based on its intrinsic settings. In this scenario, the POMDP-based Task Manager performance was calculated with:

$$P_{hits} = P_{CR} + P(\mu_{SC}|e_u, e) \ , \tag{6.3}$$

where P_{CR} is the probability that the ERM correctly detects that the user did not make an error, and $P(\mu_{SC}|e_u, e)$ is the probability that the APM retrieves a semantically correct recommendation μ_{SC} ($\mu_{SC} \in A$) given that the user made an error e_u and that the ERM detects that an error has been made e ($e \in E$). This scenario tests the ability of the Task Manager to select semantically correct recommendations (see Section 1.1.2).

6.4 Results

This section focuses on the performance of the APM and ERM implemented in the POMDP-based Task Manager. The evaluations were run via simulation and the results obtained were compared with the MDP-based Task Manager described in Chapter 5. Note that in this Section, the POMDP-based APM and ERM used the Euclidean metric to find the closest neighbor of the current belief state they are in, if the latter was not already contained in their respective belief subspace. By contrast, recall that the MDP-based Task Manager does not use any metric during the evaluation.

6.4.1 APM performance only

6.4.1.1 First scenario

Here, the assumption is made that the POMDP-based Task Manager knows the ARS confusion matrix, and each graph depicts the *SimUs*' success rate at varying ARS error rate. Figures 6.3, 6.4 and 6.5 show each *SimU*'s performance when interacting with the MDP-based APM, the POMDP-based APM and when following the APM outputs 0% of the time, for respectively "Black tea", "Black tea with sugar", "White tea with sugar".

As expected, one can see that the POMDP-based APM is more efficient than the MDP one at increasing ARS error rate for all types of tea. For example, at 30% ARS error rate, for "Black tea", "Black tea with sugar" and "White tea with sugar", the *SimU*α success rate was respectively 96%, 55%, 39.6% when complying to the POMDP-based APM outputs, compared to 42.6%, 26%, 23% when complying to the MDP-based APM outputs.

As discussed in Chapter 5 Section 5.5.2, one can note that when the complexity of the task increases, the *SimUs*' success rate decreases. Indeed, when interacting with the POMDP-based APM, and at increasing ARS error rate, the *SimUs* have a higher probability to succeed a simple task such as "Black tea" rather than "White tea with sugar". For example, as noted earlier, the *SimU*α success rate was 96% at 30% ARS error rate for "Black tea" and 39.6% during "White tea with sugar".

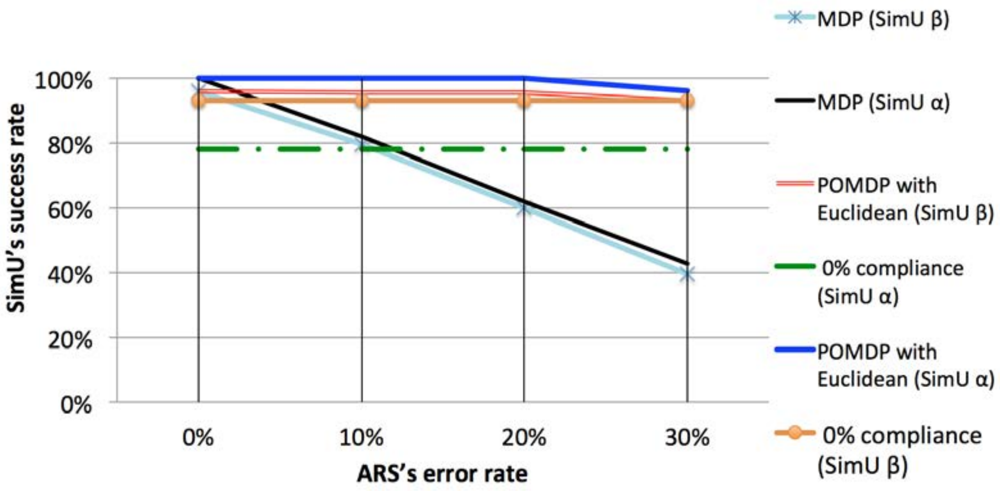


Figure 6.3: SimU success rate at varying ARS error rate during “Black tea”. POMDP-based APM using euclidean distance. ARS confusion matrix observable.

This can be explained by the fact that during “White tea with sugar”, each belief state is a probability distribution over 1539 states (i.e., each time the APM receives an observation, it may believe that the user is potentially in 1539 different states at the same time). By contrast, during “Black tea”, each belief state is a probability distribution over 33 states only. It is known that one of the difficulties in solving POMDPs can be attributed to the curse of dimensionality [83]. The curse of dimensionality refers to the increase in complexity because of the number of hidden states.

Comparing the $SimUs$ success rate when following the APM outputs 0% of the time and when interacting with the POMDP-based APM allows to measure the usefulness of the system. One can see that for all types of tea, it is more advantageous for $SimU\alpha$ to comply with the POMDP-based APM outputs rather than trying to perform the tasks by itself. As highlighted in Section 5.5.2, $SimU\beta$ has a higher success rate than $SimU\alpha$ when performing the tasks by itself. During “Black tea” (Figure 6.3) it is advantageous for $SimU\beta$ to interact with the POMDP-based APM up to 30% ARS error rate.

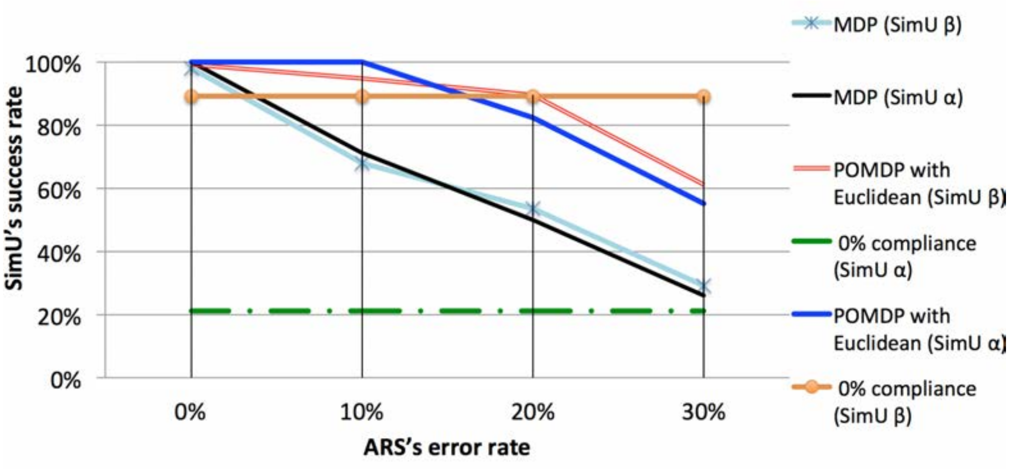


Figure 6.4: SimU success rate at varying ARS error rate during "Black tea with sugar". POMDP-based APM using Euclidean distance. ARS confusion matrix observable.

At 30% ARS error rate, *SimUβ* success rate is the same whether it interacts with the system or performs the task by itself. During "Black tea with sugar" (Figure 6.4), the POMDP-based APM outputs are advantageous for *SimUβ* up to 20% ARS error rate. At a higher ARS error rate, the POMDP-based APM outputs begin to have a downward impact on *SimUβ* success rate. For example, at 30% ARS error rate, *SimUβ* succeeds the task more often by itself (89% success rate) than when it interacts with the POMDP-based APM (61% success rate). At the same level of ARS error rate, *SimUα* succeeds the task 55% of the time. In this specific case, some of the errors it made and which ultimately led it to fail some trials were omission and perseveration errors. For example, in some trials, *SimUα* forgot to add sugar, which is a mandatory ingredient during "Black tea with sugar" (see E23 in Table 3.1). It also sometimes added too many teabags in the cup or boiled water multiple times (see E09 and E11 in Table 3.1). Another important difference between *SimUα* and *SimUβ* is their level of compliance toward the POMDP-based APM outputs. *SimUα* complies all the time with the POMDP-based APM outputs, which is not the case with *SimUβ*.

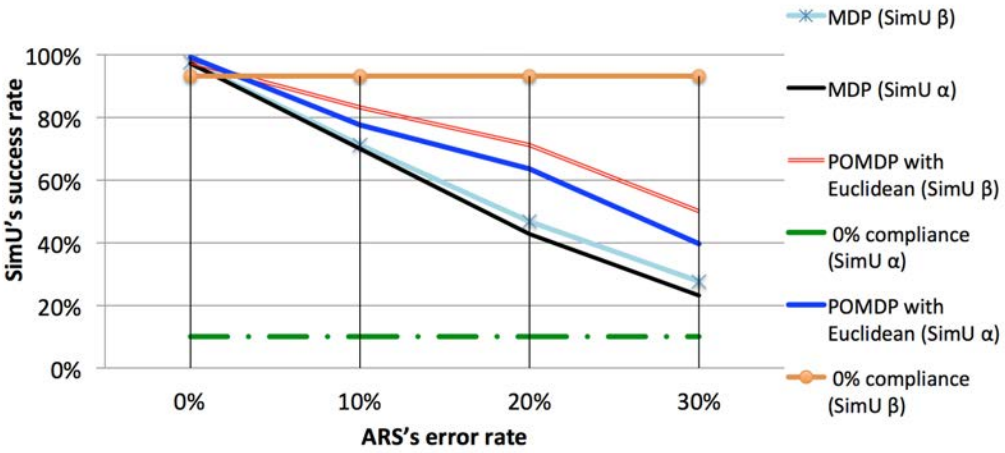


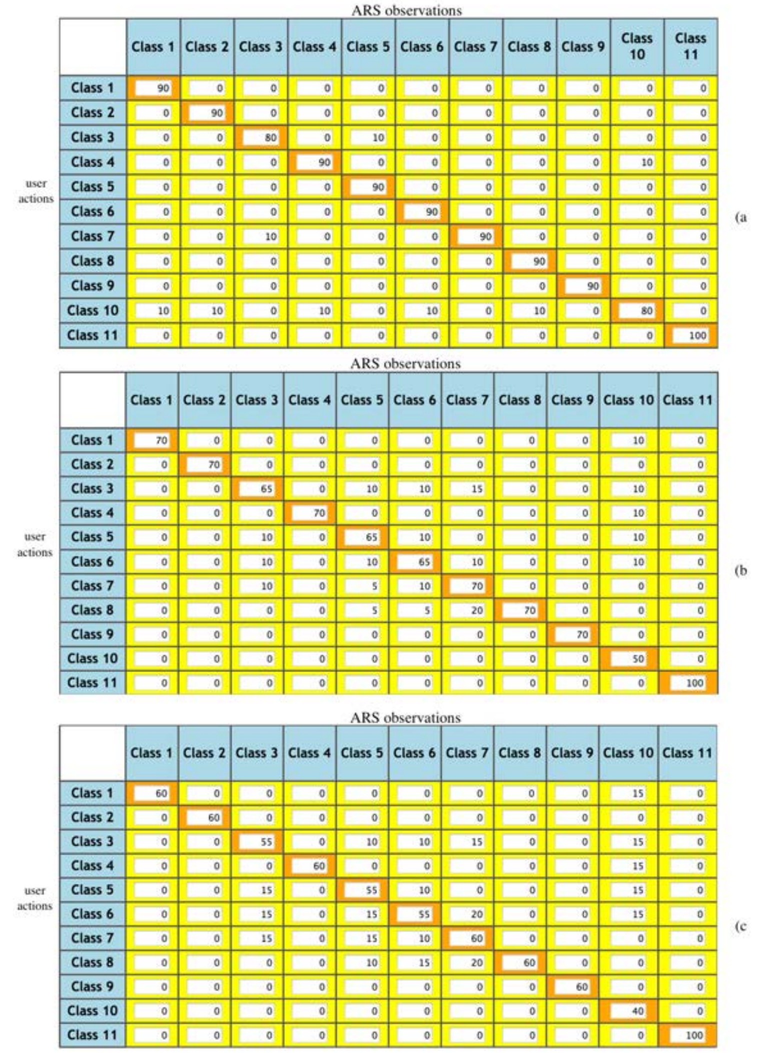
Figure 6.5: SimU success rate at varying ARS error rate during "White tea with sugar". POMDP-based APM using Euclidean distance. ARS confusion matrix observable.

The latter follows a predefined sequence of actions up to 50%, then complies with the POMDP-based APM outputs (see Section 4.3). Hence, $SimU\beta$ tends to have a lower compliance to the system's recommendations compared to $SimU\alpha$. For example, during "Black tea", the $SimU\alpha$ success rate was higher than $SimU\beta$ success rate. Hence, it was advantageous to always comply to the POMDP-based APM outputs during this task. However, when the task complexity increased (with "Black tea with sugar" and "White tea with sugar"), to not always comply to the system's outputs allowed $SimU\beta$ to have a higher success rate than $SimU\alpha$ at increasing ARS error rate. For example, during "Black tea with sugar" (Figure 6.4), at around 14% ARS error rate, one can see that $SimU\beta$ began to outperform $SimU\alpha$; and at 20% ARS error rate, $SimU\beta$ had 89% success rate compared to 82% success rate for $SimU\alpha$. In other words, during complex tasks, when the ARS error rate increases, as the POMDP-based APM outputs have a higher probability to be suboptimal or incorrect, the fact that $SimU\beta$ does not comply all the time with the POMDP-based APM outputs tend to be an advantage.

6.4.1.2 Second scenario

In Equation 2.21, the observation model takes into account the corruption induced by the ARS - its confusion matrix. In this scenario, the assumption is made that the POMDP-based APM does not correctly acknowledge the corruption induced by the ARS. In other words, the confusion matrix it takes into account in equation 2.21 is different from the real confusion matrix of the virtual ARS. This phenomenon may occur if the ARS confusion matrix is not perfectly known, and the Task Manager only has an approximate understanding of the type of errors made by the ARS during execution.

In Figures 6.6 and 6.7, one can see the different confusion matrices related to the virtual ARS and the confusion matrices as seen by the POMDP-based APM. They are similar, however, the distribution of errors among actions is not the same. We can also highlight that this distribution of errors is more important in this scenario than in the previous one (see the confusion matrices used in scenario 1 - Figure 5.7.) Indeed, in the previous scenario, the confusion matrices implemented were sparser at the level of actions which are mandatory and cannot be repeated (see Table 3.2); for example “Add teabag”, “Add sugar”, “Remove teabag”.



ARS observations

user actions

	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Class 10	Class 11
Class 1	90	0	0	0	0	0	0	0	0	0	0
Class 2	0	90	0	0	0	0	0	0	0	0	0
Class 3	0	0	80	0	10	0	0	0	0	0	0
Class 4	0	0	0	90	0	0	0	0	0	10	0
Class 5	0	0	0	0	90	0	0	0	0	0	0
Class 6	0	0	0	0	0	90	0	0	0	0	0
Class 7	0	0	10	0	0	0	90	0	0	0	0
Class 8	0	0	0	0	0	0	0	90	0	0	0
Class 9	0	0	0	0	0	0	0	0	90	0	0
Class 10	10	10	0	10	0	10	0	10	0	80	0
Class 11	0	0	0	0	0	0	0	0	0	0	100

(a

ARS observations

user actions

	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Class 10	Class 11
Class 1	70	0	0	0	0	0	0	0	0	10	0
Class 2	0	70	0	0	0	0	0	0	0	0	0
Class 3	0	0	65	0	10	10	15	0	0	10	0
Class 4	0	0	0	70	0	0	0	0	0	10	0
Class 5	0	0	10	0	65	10	0	0	0	10	0
Class 6	0	0	10	0	10	65	10	0	0	10	0
Class 7	0	0	10	0	5	10	70	0	0	0	0
Class 8	0	0	0	0	5	5	20	70	0	0	0
Class 9	0	0	0	0	0	0	0	0	70	0	0
Class 10	0	0	0	0	0	0	0	0	0	50	0
Class 11	0	0	0	0	0	0	0	0	0	0	100

(b

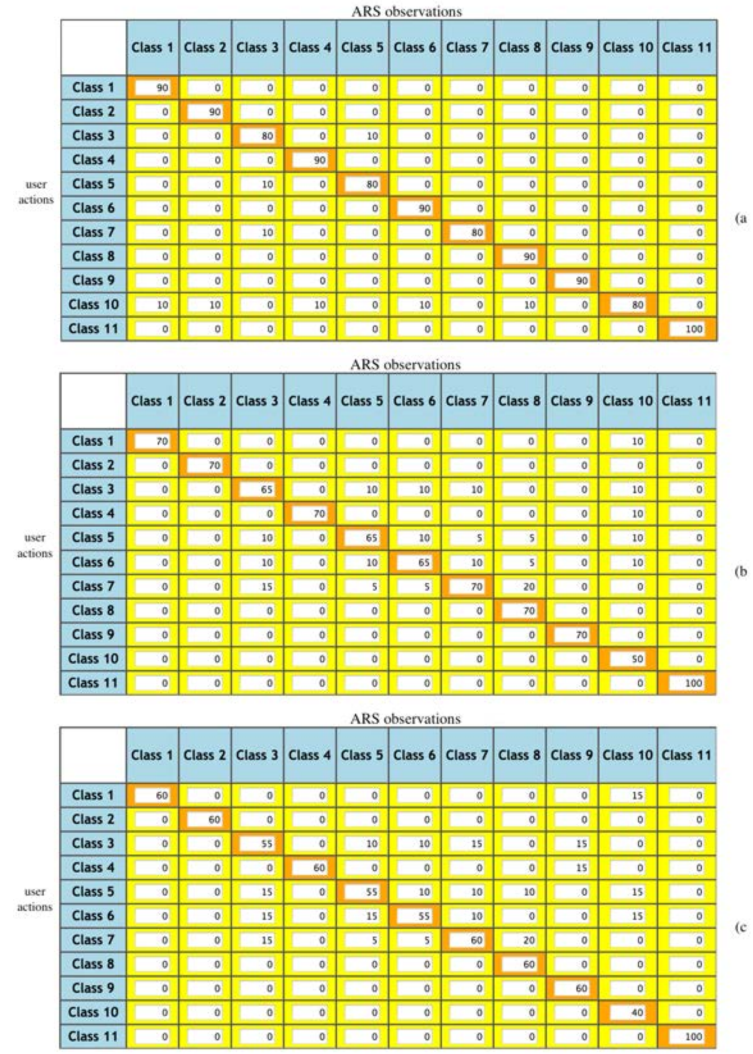
ARS observations

user actions

	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Class 10	Class 11
Class 1	60	0	0	0	0	0	0	0	0	15	0
Class 2	0	60	0	0	0	0	0	0	0	0	0
Class 3	0	0	55	0	10	10	15	0	0	15	0
Class 4	0	0	0	60	0	0	0	0	0	15	0
Class 5	0	0	15	0	55	10	0	0	0	15	0
Class 6	0	0	15	0	15	55	20	0	0	15	0
Class 7	0	0	15	0	15	10	60	0	0	0	0
Class 8	0	0	0	0	10	15	20	60	0	0	0
Class 9	0	0	0	0	0	0	0	0	60	0	0
Class 10	0	0	0	0	0	0	0	0	0	40	0
Class 11	0	0	0	0	0	0	0	0	0	0	100

(c

Figure 6.6: Confusion matrices as implemented in the virtual ARS. (a - ARS error rate 7.5% (approximation used: 8%), (b - ARS error rate 20%, (c - ARS error rate 29%. Class 1: "Fill kettle", Class 2: "Boil water", Class 3: "Add teabag", Class 4: "Pour kettle", Class 5: "Add sugar", Class 6: "Add milk", Class 7: "Stir", Class 8: "Remove teabag", Class 9: "Pour water from jug to cup", Class 10: "Toying with boiling water", Class 11: ∅ (no action performed or detected).



ARS observations

user actions

(a

	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Class 10	Class 11
Class 1	90	0	0	0	0	0	0	0	0	0	0
Class 2	0	90	0	0	0	0	0	0	0	0	0
Class 3	0	0	80	0	10	0	0	0	0	0	0
Class 4	0	0	0	90	0	0	0	0	0	0	0
Class 5	0	0	10	0	80	0	0	0	0	0	0
Class 6	0	0	0	0	0	90	0	0	0	0	0
Class 7	0	0	10	0	0	0	80	0	0	0	0
Class 8	0	0	0	0	0	0	0	90	0	0	0
Class 9	0	0	0	0	0	0	0	0	90	0	0
Class 10	10	10	0	10	0	10	0	10	0	80	0
Class 11	0	0	0	0	0	0	0	0	0	0	100

ARS observations

user actions

(b

	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Class 10	Class 11
Class 1	70	0	0	0	0	0	0	0	0	10	0
Class 2	0	70	0	0	0	0	0	0	0	0	0
Class 3	0	0	65	0	10	10	10	0	0	10	0
Class 4	0	0	0	70	0	0	0	0	0	10	0
Class 5	0	0	10	0	65	10	5	5	0	10	0
Class 6	0	0	10	0	10	65	10	5	0	10	0
Class 7	0	0	15	0	5	5	70	20	0	0	0
Class 8	0	0	0	0	0	0	0	70	0	0	0
Class 9	0	0	0	0	0	0	0	0	70	0	0
Class 10	0	0	0	0	0	0	0	0	0	50	0
Class 11	0	0	0	0	0	0	0	0	0	0	100

ARS observations

user actions

(c

	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Class 10	Class 11
Class 1	60	0	0	0	0	0	0	0	0	15	0
Class 2	0	60	0	0	0	0	0	0	0	0	0
Class 3	0	0	55	0	10	10	15	0	15	0	0
Class 4	0	0	0	60	0	0	0	0	15	0	0
Class 5	0	0	15	0	55	10	10	10	0	15	0
Class 6	0	0	15	0	15	55	10	0	0	15	0
Class 7	0	0	15	0	5	5	60	20	0	0	0
Class 8	0	0	0	0	0	0	0	60	0	0	0
Class 9	0	0	0	0	0	0	0	0	60	0	0
Class 10	0	0	0	0	0	0	0	0	0	40	0
Class 11	0	0	0	0	0	0	0	0	0	0	100

Figure 6.7: Confusion matrices as seen by the POMDP-based APM. (a - ARS error rate 7.7%, (b - ARS error rate 20%, (c - ARS error rate 26.5%. Class 1: “Fill kettle”, Class 2: “Boil water”, Class 3: “Add teabag”, Class 4: “Pour kettle”, Class 5:“Add sugar”, Class 6:“Add milk”, Class 7: “Stir”, Class 8:“Remove teabag”, Class 9:“Pour water from jug to cup”, Class 10:“Toying with boiling water”, Class 11: ∅ (no action performed or detected).

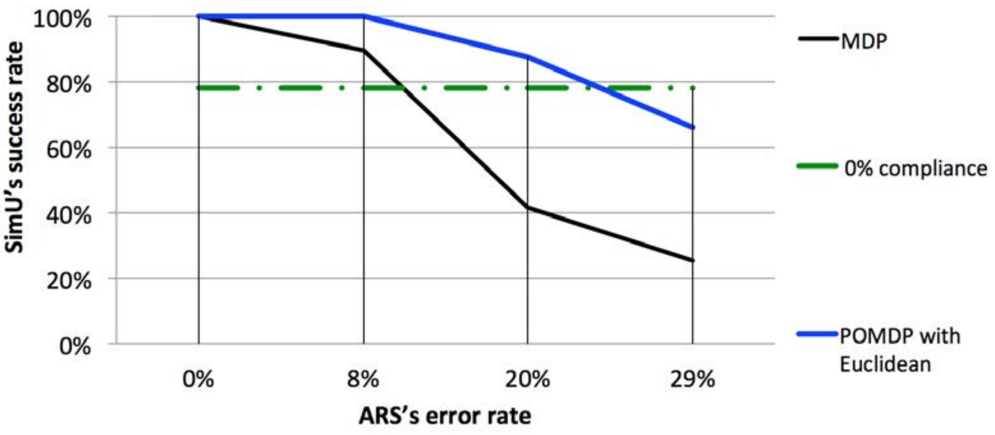


Figure 6.8: *SimUα* success rate at varying ARS error rate during "Black tea". POMDP-based APM using Euclidean distance. POMDP-based TM has an approximation of ARS confusion matrix.

Figures 6.8, 6.9 and 6.10 show *SimUα* success rate at varying ARS error rate, for respectively "Black tea", "Black tea with sugar" and "White tea with sugar", with the POMDP-based APM using the Euclidean metric for its Nearest Neighbor Search (NNS) technique.
One can see that for "Black tea" and "Black tea with sugar", the POMDP-based APM is more efficient than the MDP one at increasing ARS error rate. However, for "Black tea", at 29% ARS error rate, the *SimU* succeeded the task more often when ignoring the POMDP-based APM prompts. Indeed, at 29% ARS error rate, the *SimU* succeeded the task 66% of the time with the POMDP-based APM, compared to 78% success rate when it ignored the system. This is a sign of lost of usefulness as far as the system is concerned. When performing "White tea with sugar" in Figure 6.10, both MDP and POMDP-based APM have a similar impact on the *SimU* success rate, except at 29% ARS error rate, where one can see that the MDP outperforms the POMDP. Indeed, both MDP and POMDP are affected by the complexity of the task, as seen in the first scenario.

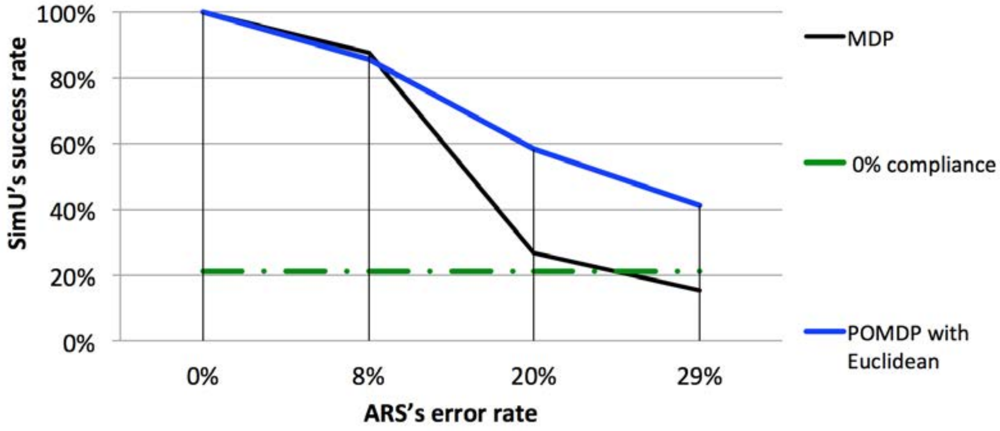


Figure 6.9: *SimUα* success rate at varying ARS error rate during "Black tea with sugar". POMDP-based APM using Euclidean distance. POMDP-based TM has an approximation of ARS confusion matrix.

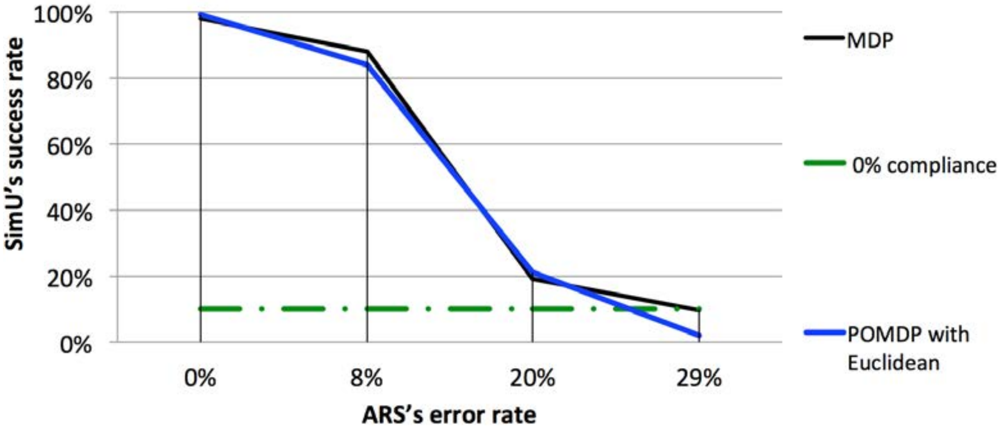


Figure 6.10: *SimUα* success rate at varying ARS error rate during "White tea with sugar". POMDP-based APM using Euclidean distance. POMDP-based TM has an approximation of ARS confusion matrix.

Note that during "Black tea with sugar", when interacting with the POMDP-based system at 29% ARS error rate, the simulated user made more fatal errors than in the case where the ARS confusion matrix was observable by the Task Manager (see Figure 6.4 at 30% ARS error rate). Indeed, here, the simulated user tends to make more perseveration errors (for example, repeating too many times actions such as "Add sugar", "Add teabag", "Stir" and "Pour water to cup"). The number of time such actions can be repeated during a trial before the system considers that a fatal error has been made was given in Table 3.2. These perseveration errors tend to occur when the ARS sends a wrong observation to the Task Manager. For example, the simulated user performs the action "Add sugar", but the ARS outputs that it has just stirred. In some of these cases, the Task Manager would then believe that some mandatory actions are missing from the simulated user's state, and take the decision to send the recommendation to do an action that the simulated user has already performed. Following the previous example, it means that the Task Manager would recommend the simulated user to add sugar to the cup, even thought the simulated user has already performed this action. Due to its level of compliance, the simulated user would then follow the system's recommendation, repeat "Add sugar" and fail the task.

6.4.2 ERM performance only

In Figure 6.11, one can see the percentage of correct detection, false positive and false negative made by the ERM implemented in the MDP-based Task Manager and in the POMDP-based Task Manager. Before being implemented in the POMDP-based Task Manager, the ERM was trained following Algorithm 3, with $SimU\alpha$ displaying two different behaviors (i.e., training γ and training η). During training γ, $SimU\alpha$ could randomly choose to follow a predefined sequence of actions, or select actions following its intrinsic configuration and comply to the POMDP-based APM outputs when the latter were passed to it. During training η, $SimU\alpha$ was configured to follow the system's outputs 0% of the time; so it performs the tasks by itself. During evaluation, $SimU\alpha$ was used and configured to follow the APM's outputs 100% of the time when the latter were sent to it.

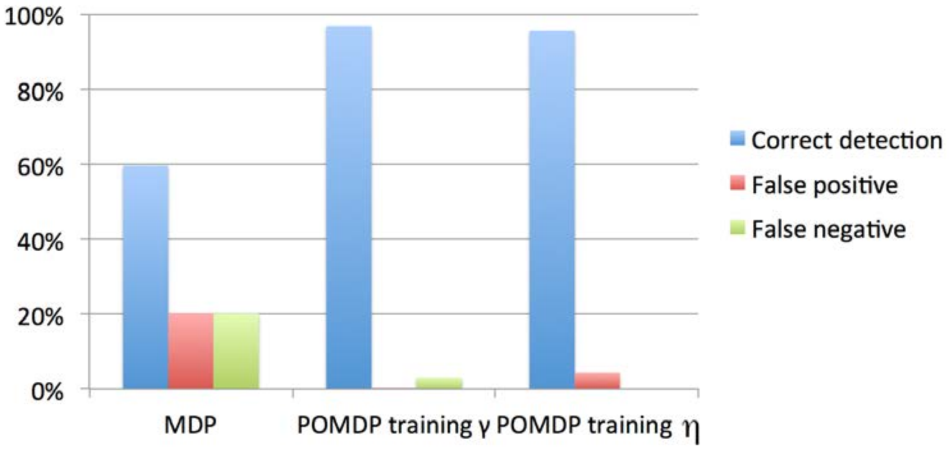


Figure 6.11: ERM evaluation: percentage of correct detection, false positive and false negative. ARS error rate: 30%. Task: "Black tea".

Contrary to the ERM implemented in the POMDP-based Task Manager, the ERM implemented in the MDP-based Task Manager is not trained; it follows specific rules explained in Section 5.2.4. As one can see in Figure 6.11, the performance of the ERM implemented in the POMDP-based Task Manager outperforms the one implemented in the MDP-based Task Manager. Indeed, at 30% ARS error rate during "Black tea", the ERM implemented in the MDP-based system correctly detected when $SimU\alpha$ made errors or not (True positive and True negative) 59% of the time, while the ERM implemented in the POMDP-based Task Manager was able to correctly detect True positive and True negative around 97% of the time when trained following training γ and 96% when trained following training η.

We can note that the behavior of the simulated user implemented to train the ERM (POMDP-based system) did not have an important impact of its ability to detect True positives and True negatives.

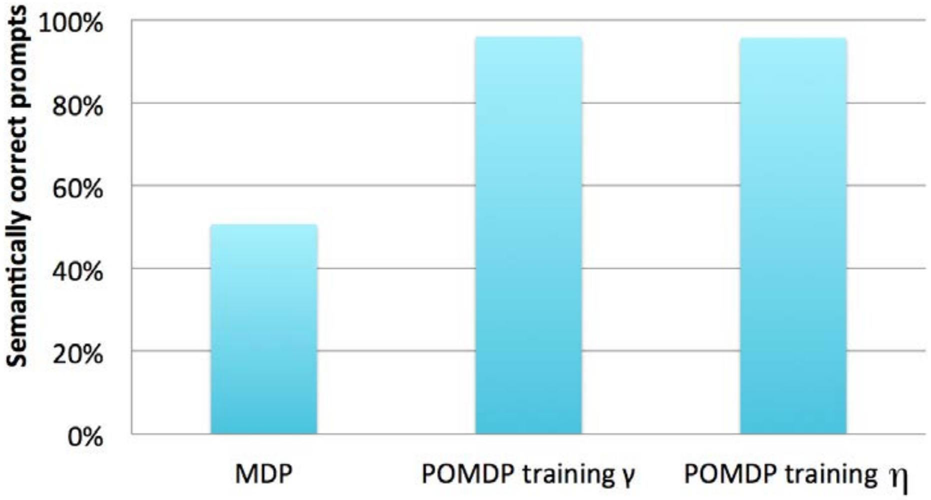


Figure 6.12: ERM evaluation: percentage of semantically correct prompts delivered to a simulated user. ARS error rate: 30%. Task: "Black tea".

6.4.3 Joint performance

Before running the joint performance, the ERM implemented in the POMDP-based Task Manager was trained as explained in the previous subsection 6.4.2. The APM was trained as explained in section 6.3.1. During evaluation, $SimU\alpha$ was configured to follow the APM's outputs 100% of the time, when the latter were sent to it.

Figure 6.12 shows the percentage of semantically correct prompts delivered by the POMDP-based Task Manager during "Black tea", at 30% ARS error rate. The notion of semantically correctness was explained in Section 6.3.4.1. It takes into account the number of times the ERM correctly detects when the simulated user makes an error or not, and whether the recommendation suggested by the APM is semantically correct according to what the user has achieved so far. One can also see that the joint utilization of the ERM and APM implemented in the POMDP-based Task Manager outperformed the one implemented in the MDP-based Task Manager.

This is due to the fact that in the POMDP-based Task Manager, the POMDP-based APM has a higher probability to output correct prompts than in the MDP-based Task Manager under uncertainty. The other reason is related to the fact that the ERM implemented in the POMDP-based Task Manager has a better ability to detect True positives and True negatives than the one implemented in the MDP-based Task Manager under uncertainty. For example, one can see that the MDP-based Task Manager provides semantically correct prompts around 50% of the time only, while the POMDP-based Task Manager succeeds to do so 96% of the time.

6.5 Summary

This section described in detail the POMDP-based APM and ERM within the Task Manager, and how these modules plan under uncertainty. We explained how the MDP once solved can be reused within the POMDP. The POMDP framework and the implementation of its factorization within CogWatch were fully described. Both APM and ERM implemented in the POMDP-based Task Manager were evaluated via user simulation, and compared with the MDP-based system. The results showed that the ability of the POMDP to model the uncertainty in its environment makes it more robust against noise compared to the MDP. However, in some cases, we showed that the POMDP does not succeed to outperform the MDP. The evaluation was run while using the Euclidean metric for the POMDP-based APM and ERM.

In the next section, we will focus on the POMDP-based APM and run the same evaluation with different metrics and classification techniques. The aim will be to see how the Nearest Neighbor Search technique or classifiers such as SVM can affect the performance of the system, at varying ARS error rates. Indeed, in the case where the POMDP-based APM updates a belief state that is not part of its belief subspace obtained after training, its ability to retrieve an appropriate neighbor for this belief state in order to copy its strategy, may depend on the NNS technique implemented.

Chapter 7

Nearest Neighbor Search in the POMDP-based APM

7.1 Introduction

During evaluation, the POMDP-based APM may update a belief state that it has never seen during training, and for which it has no optimal recommendation associated. In such a case, the nearest neighbor to this belief state which already has an optimal recommendation μ^* can be found, and μ^* will be output. This technique consists of finding the neighbor of a given belief state and imitating its behavior [17]. Hence, the problem of selecting the optimal recommendation for a belief state can be seen as a classification problem. For example, in Figure 7.1, the updated belief state b'_e generated by the ERM belief state estimator is not contained in M_e, so the neighbor $\widetilde{b}$ is selected instead. The ERM then associates the recommendation of this neighbor to the current belief state.

In the previous chapter, the results focused on the performance of the POMDP-based system when implementing the Euclidean metric to select neighbors when necessary. However, many metric distance functions and approaches can be used to solve the Nearest Neighbor Search (NNS) problem [1; 10]. Moreover, as highlighted in [2; 34; 111], the accuracy of the k-nearest neighbor (KNN) classification will depend on the metric applied, and on the dimensionality of the points taken into account.

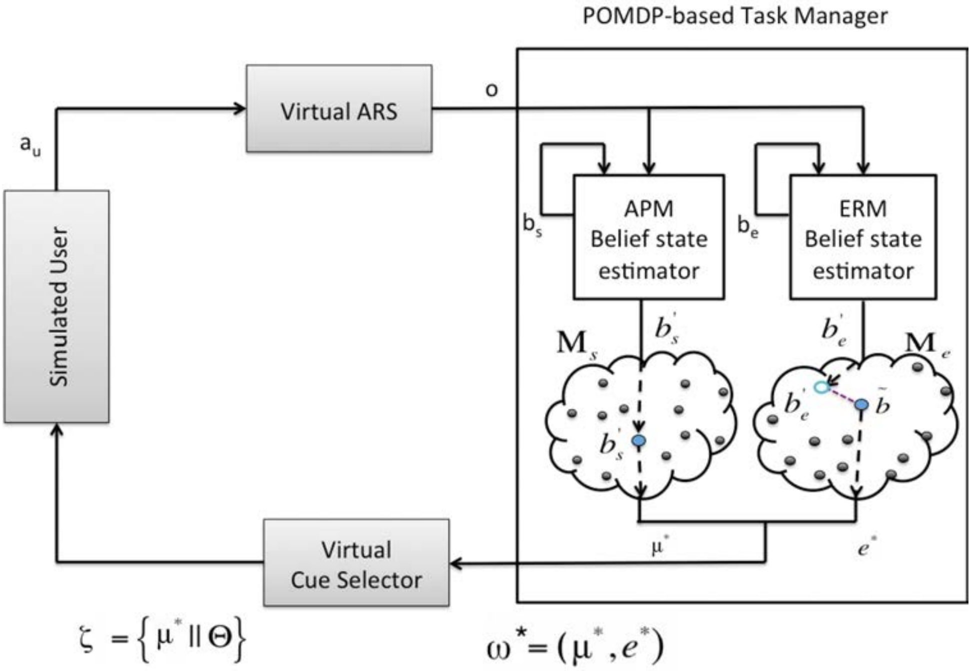


Figure 7.1: Detailed diagram of the system used to evaluate the POMDP-based Task Manager

The aim of this chapter is to investigate the impact of the metrics used to select the nearest neighbors on the quality of the CogWatch POMDP-based APM's outputs. We will show that the POMDP-based APM's performance does not only depends on the ARS error rate, but also on the metric or method applied to select neighbors during evaluation. We will compare the impact of techniques such as Support Vector Machine (SVM), Euclidean distance, Manhattan distance, Correlation distance, k-d tree and another metric-based algorithm that we developed, and that we will referred to as SciMK.

7.1.1 Classifier

The policy obtained after training of the Task Manager is used as a labeled training set for the classifier implemented in this work. As discussed, the problem can be seen as a classification problem where the system needs to find to which class each current belief state belongs.

7.1.1.1 Support Vector Machine (SVM)

A SVM performs classification tasks by constructing hyperplanes in a multidimensional space that separates cases of different class labels [59; 109]. Specifically, in this thesis, we use Support Vector Classification with a linear kernel [38; 78] to associate a class (i.e., $\mu \in A$) to each belief state updated during the evaluation. In a context of multi-class classification, SVM was also implemented by the authors in [111] and compared to other techniques.

7.1.2 Metrics

The policy obtained after training of the POMDP-based APM is used as a set of potential neighbors. Each time a belief state is updated, if it is not contained in this set, a metric is used to compare this belief state with all the belief states contained in the set (i.e., brute-force search). Among the main metric distance functions that exist [1], we implemented: the Euclidean distance, the Manhattan distance [56], and the correlation distance [77].

7.1.3 Tree based approach structure

7.1.3.1 K-d tree

Given a set of belief points, k-d tree aggregates distance information, acknowledging the fact that if point A is very distant from point B, and point B is very close to point C, then point A is very far from point C, without having to calculate the distance between A and C. Hence, k-d tree allows to reduce the computational cost related to brute-force search, by reducing the number of distance calculations to compute in order to find a neighbor. A more detailed description of the algorithm can be found in [8]. In this work, when k-d tree is implemented, the Euclidean metric is used to calculate distances.

7.2 SciMK

SciMK is a novel algorithm for comparing belief states. It measures the similarity of two belief states by comparing the underlying patterns of their MDP recommendations. A belief state is a probability distribution over the MDP states. Each MDP state is associated with an optimal recommendation μ^* after solving the MDP (see Chapter 5). Hence each belief state can be transformed into a corresponding probability distribution over MDP recommendations. A unique numerical label (NL) is then associated to each MDP recommendation to facilitate the comparison between vectors. The motivation for SciMK is that the similarity between two belief states could be based on the underlying distribution over recommendations rather than MDP states, because the purpose of the process is to allocate recommendations to unseen belief states.
For example, in Figure 7.2:

- Step (0), the belief state is represented as a probability distribution b over MDP states.
- Step (1), the probability distribution over MDP states b is sorted in descending order of probabilities.
- Step (2), the belief state is represented as a probability distribution $\tilde{b}$ over the MDP recommendations corresponding to each MDP state from step (1).
- Step (3), the belief state is represented as a vector $v(\tilde{b})$ of numerical labels (i.e., weights), each of the latter corresponding to a specific MDP recommendation. The vector $v(\tilde{b})$ is referred to as a “NL vector”.

In the example given in Figure 7.2, step (3), $\pi^*(state_2) = \pi^*(state_N)$ and their corresponding Numerical Label is 10.

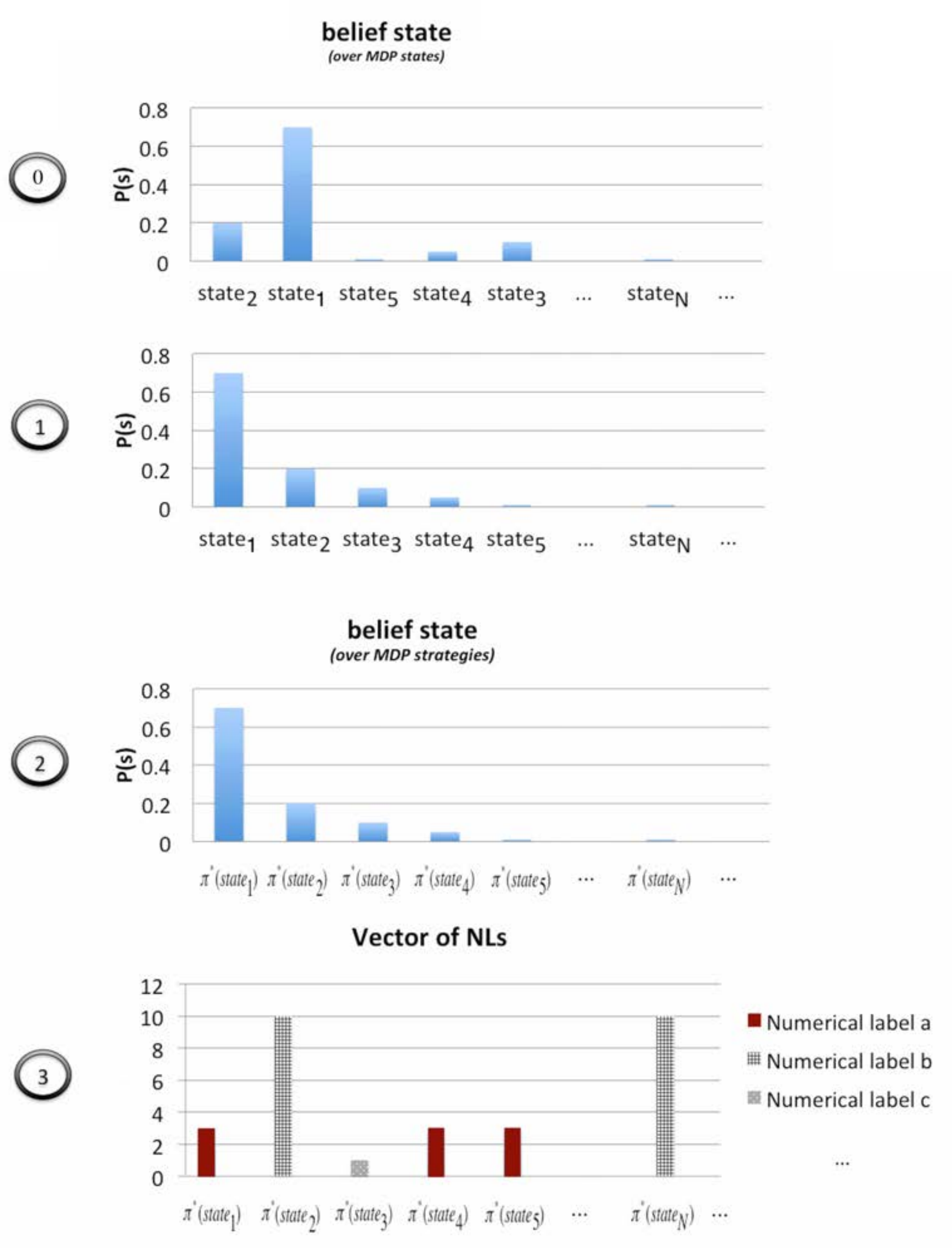


Figure 7.2: Belief state transformation.

A computational advantage of SciMK is that the space of NL vectors can be partitioned in a way that can speed up the KNN process. The partitioning process works as follows:

1. Each belief state contained in the belief subspace is transformed into its corresponding NL vector.

2. The space of NL vectors is partitioned into equivalence classes. NL vectors are grouped into the same equivalence class if, in step 2 (Figure 7.2), their maximum probabilities are associated with the same MDP recommendation.

The partitioning can be done offline, before evaluation. During evaluation, if a newly updated belief state b is not contained in the belief subspace, SciMK searches for an appropriate neighbor in the space of NL vectors, by following those major steps:

1. The belief state b is sorted in descending order of probabilities. All states which have a probability zero are removed.

2. The belief state is transformed into its corresponding probability distribution over MDP recommendations $\tilde{b}$ (see step (2) in Figure 7.2).

3. The MDP recommendation $\mu_p(\tilde{b})$, where $\mu_p(\tilde{b}) = \arg\max_{\mu \in A} \tilde{b}(\mu)$, is recorded. Here, $\mu_p(\tilde{b})$ is the MDP recommendation which has the highest probability in $\tilde{b}$. It is also the MDP recommendation associated to the state s with the highest probability in b.

4. The transformed belief state $\tilde{b}$ is mapped to its NL vector $v(\tilde{b})$.

5. The equivalence class h in the space of NL vectors that $v(\tilde{b})$ belongs to is selected. From a computational point of view, the space of NL vectors is a hash table, where recommendations are the keys and sets of vectors are the values. To select the cluster h that corresponds to $v(\tilde{b})$ means to retrieve the set of NL vectors which had their highest probability associated to $\mu_p(\tilde{b})$ when they were probability distributions over MDP recommendations.

6. The distance between the NL vector v and each vector u in h, when u and v have the same dimension, is obtained using the Euclidean norm $\|v-u\|_2$. Let $dim(v)$ be the size of vector v. If $dim(v) \neq dim(u)$ and $|dim(v)-dim(u)| < \Delta$, where Δ is a threshold to be chosen, the size of the vector with the highest dimension (v or u) is reduced, so v and u have the same dimension. When the dimensionality of a vector needs to be reduced, the algorithm successively removes from it the actions that had the lowest probabilities when they were probability distributions over MDP recommendations (see step (2) in Figure 7.2). If $|dim(v) - dim(u)| \geq \Delta$, then u is rejected and is not considered as a potential neighbor that could be selected.

7. The neighbor u_p is selected such that $u_p = \arg\min_{u \in h} \|v - u\|_2$.

8. The belief state b_p which, once transformed, created the neighbor u_p, is retrieved from the belief subspace B.

9. The POMDP recommendation μ^*, where $\mu^* = \pi^*(b_p)$ is selected by the POMDP-based APM and sent to the user or simulated user.

For example, in Figure 7.2 suppose that $\pi^*_M(state_1) =$"add teabag" (i.e., MDP optimal recommendation). Since $state_1$ has the highest probability in the belief state, the NL vector in step (3) is only compared to other NL vectors whose highest probability was also allocated to "add teabag" in step (2).
Note that the belief states corresponding to the vectors of NLs contained in h are not all associated to the same optimal recommendation found during training of the POMDP-based APM. In other words, the MDP recommendation which has the highest probability in $\tilde{b}$ does not necessarily correspond to the POMDP recommendation associated to b. SciMK does not come down to consider that $\pi^*(b) = \mu_p(\tilde{b})$.

7.3 Evaluation

This section focuses on the performance of the APM implemented in the POMDP-based Task Manager, at varying NNS techniques and ARS error rates. The evaluations were run via simulation and the results obtained were compared with the MDP-based Task Manager described in Chapter 5. The policy $\pi^* : B \rightarrow A$ obtained via grid-based approach is reused by the POMDP-based APM (see Section 6.3).

7.3.1 Trials with a simulated user

The evaluation performed via user simulation was done by implementing $SimU\alpha$ described in Section 4.2.1. $SimU\alpha$ was configured to follow the APM's outputs 100% of the time, and was implemented so it could interact with a Virtualization of the whole CogWatch system (see Figure 6.2). The structure of the system is similar to the one where the MDP-based Task Manager was implemented during evaluation (Figure 5.6). Similarly to the evaluations described in Chapter 6, no clinician was allowed to intervene to correct the ARS outputs.

7.3.2 Evaluation of the APM

Similarly to the evaluation explained in Chapter 6 Section 6.3.4.1, in order to evaluate the APM only, we analyzed the impact of its best next recommendations μ^* on the simulated user's success rate, at varying ARS error rates, but also at varying NNS techniques, and without the effect of the Cue Selector (i.e., Errorless technique).

The scenarios described in Section 6.3.4.1 were re-run at varying NNS techniques. A third scenario was implemented in order to verify that SciMK does not come down to consider that $\pi^*(b) = \mu_p(\tilde{b})$.

7.4 Results

7.4.1 APM performance only

7.4.1.1 First scenario

In the first scenario, the assumption is made that the virtual ARS confusion matrices are perfectly observable from the APM's point of view (see Figure 5.7).

In Figure 7.3, one can see that the impact of the NNS techniques on the *SimU* success rate were similar, except for SciMK and the correlation distance, which allowed the *SimU* to succeed "Black tea" 100% of the time at varying ARS error rates.

During "Black tea with sugar", Figure 7.4, one can note that the choice of the NNS technique influenced the ability of the POMDP-based APM to select the appropriate recommendation for the *SimU*. For example, at 30% ARS error rate, when the POMDP-based APM used the Euclidean metric to select neighbors, it succeeded to make the *SimU* correctly complete its task 55% of the time. By contrast, when SVM or SciMK were implemented, the *SimU* respectively succeeded its task 74% and 83% of the time.

The same phenomenon occurs during "White tea with sugar", where the simulated user's success rate also depends on the NNS technique implemented. In Figure 7.5, one can see that SciMK outperformed the other methods. As highlighted in Chapter 6 Section 6.4, the size of the belief states has an impact on the ability of the POMDP-based APM to select appropriate recommendations. The more complicated is the task, the more the belief states' size increases, the more complicated it is for the POMDP-based APM to correctly guide the simulated user.

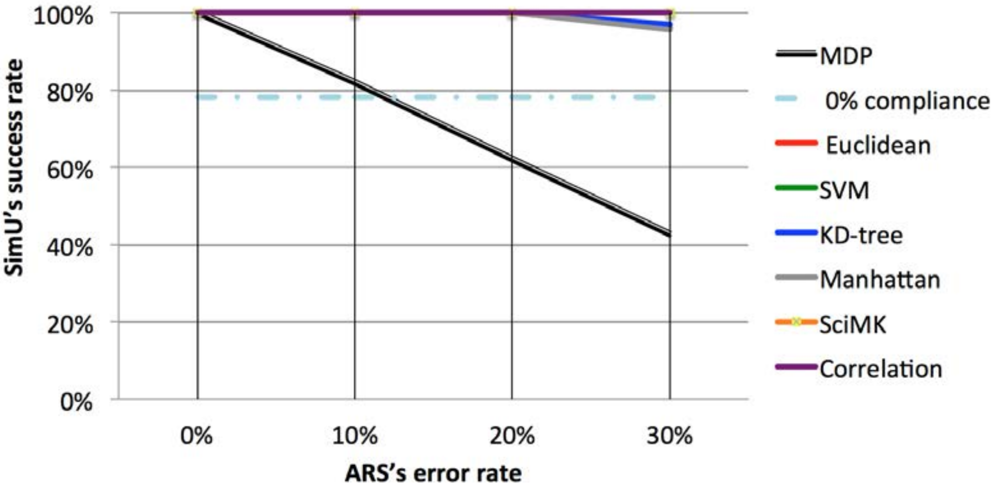


Figure 7.3: SimU α success rate at varying ARS error rate and NNS techniques. ARS confusion matrix is observable. Task: "Black tea".

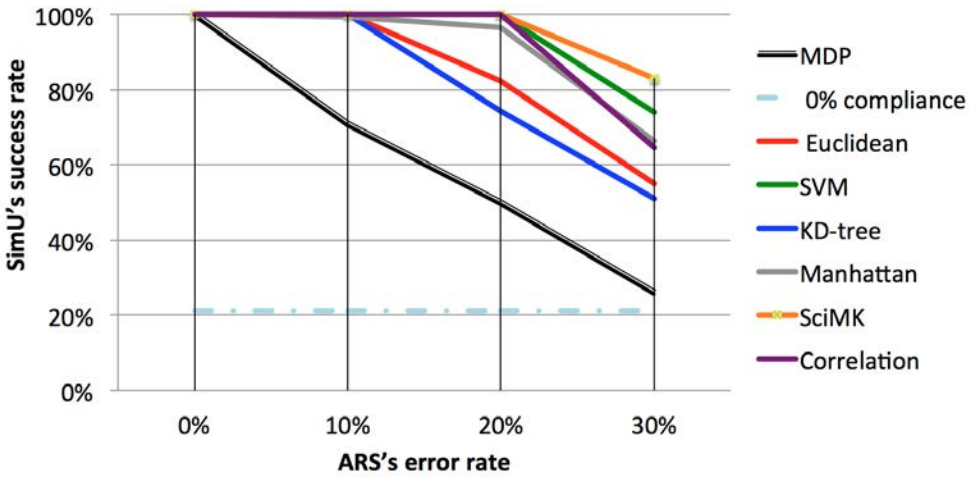


Figure 7.4: SimU α success rate at varying ARS error rate and NNS techniques. ARS confusion matrix is observable. Task: "Black tea with sugar".

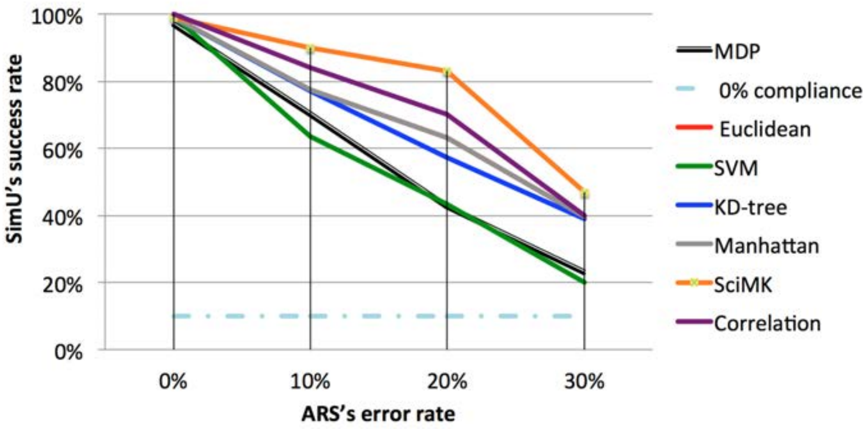


Figure 7.5: SimU α success rate at varying ARS error rate and NNS techniques. ARS confusion matrix is observable. Task: "White tea with sugar".

7.4.1.2 Second scenario

In the second scenario, the assumption is made that the virtual ARS confusion matrices are not fully known by the APM (see Figure 6.6 and 6.7). Figures 7.6, 7.7, 7.8 show $SimU\alpha$ success rate performing "Black tea", "Black tea with sugar" and "White tea with sugar". The horizontal line labeled "0% compliance" corresponds to the $SimU$ success rate when receiving no prompts. The results show the impact of the NNS techniques and SVM that were applied by the APM to choose the best next recommendation that should be retrieved given the current belief state. One can see that the best $SimU$ success rates are consistently achieved with SciMK. For example, during "Black tea with sugar", at 29% ARS error rate, when SciMK is used, the simulated user makes significantly fewer perseveration errors than when the Euclidean distance is implemented. Instead of repeating the same recommendations as discussed in Chapter 6, Section 6.4.1.2, when SciMK is implemented the Task Manager tends to propose alternative recommendations to the simulated user, which then prevents the occurrence of too many task failures. Indeed, SciMK makes the Task Manager more robust toward incorrect observations sent by the ARS.

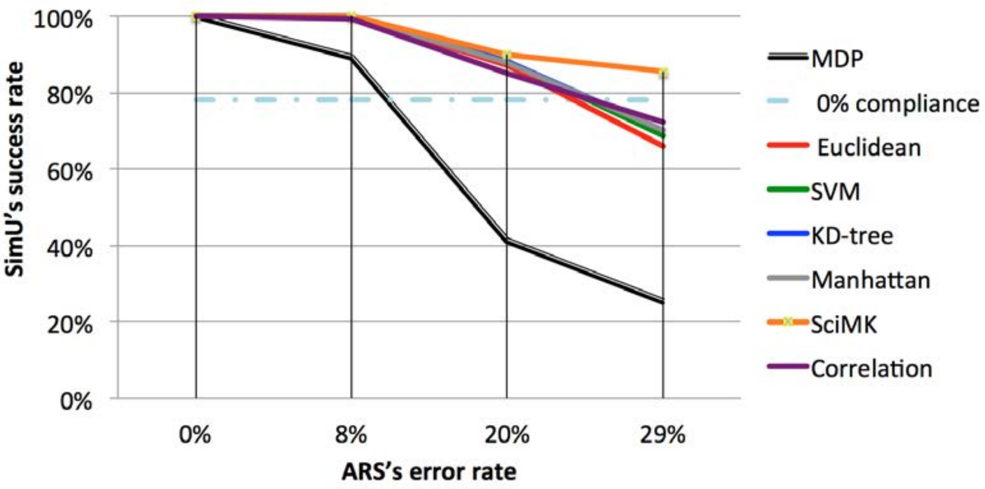


Figure 7.6: SimU α success rate at varying ARS error rate and NNS techniques. POMDP-based TM has an approximation of ARS confusion matrix. Task: “Black tea”.

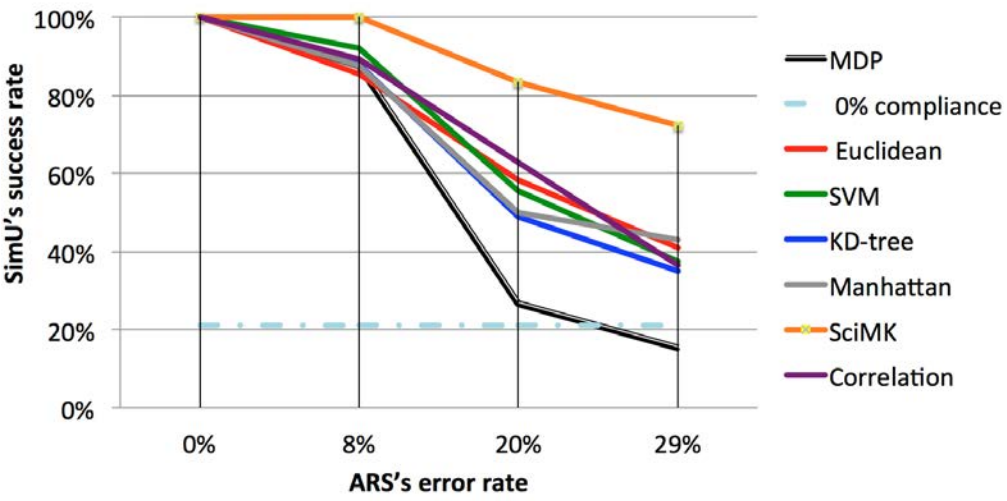


Figure 7.7: SimU α success rate at varying ARS error rate and NNS techniques. POMDP-based TM has an approximation of ARS confusion matrix. Task: "Black tea with sugar".

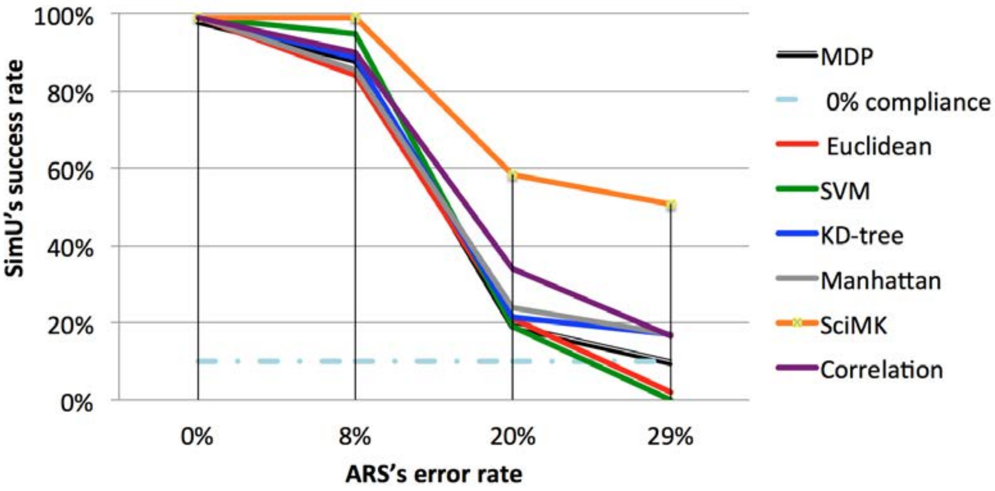


Figure 7.8: SimU α success rate at varying ARS error rate and NNS techniques. POMDP-based TM has an approximation of ARS confusion matrix. Task: "White tea with sugar".

The Figures also show that, given a new belief state, when the Euclidean and Correlation distances, and SVM were used to output a prompt, the latter had a similar impact on the *SimU* success rate at varying ARS error rates, except during "White tea and sugar". Figure 7.8 shows that SVM was more advantageous than the Euclidean distance at 8% ARS error rate, but this advantage disappeared at higher ARS error rate. The POMDP-based system outperformed the MDP except in Figure 7.8, when using the Euclidean distance or SVM at 29% ARS error rate. In this specific case, the POMDP-based APM made the *SimU* fail the task more often than when the *SimU* performed it without guidance (i.e., 0% compliance) or when guided by the MDP-based system. When performing "White tea with sugar" without guidance, the *SimU* success rate was 10%, while it was close to 0% when the POMDP-based APM selected its neighbors with the Euclidean distance or SVM.

Figures 7.6, 7.7, 7.8 also show the impact of the complexity of the task on the ability of the APM (MDP and POMDP) to guide the *SimU* properly. Table 7.1 gives a summary of the domain for each task: $|B|$ corresponds to the number of belief states contained in the subspace after training, $|O|$ the number of observations the Task Manager can receive, $|A|$ the number of recommendations μ the Task Manager can output, and $|b|$ the number of underlying MDP states composing the belief state. In other words, in the case of "White tea with sugar", each time the *SimU* selects an action, the POMDP-based APM potentially considers that the *SimU* is in 1539 different states at the same time.

Table 7.1: Tasks properties

Task	$\|B\|$	$\|O\|$	$\|A\|$	$\|b\|$
Black tea	199	10	10	33
Black tea with sugar	993	10	10	205
White tea with sugar	3982	10	10	1539

Moreover, in the case of brute force nearest neighbor search, the POMDP-based APM may have to look for a neighbor for a given belief state among 3982 candidates. The more the belief state dimensionality increases, the more difficult it is for the APM to output correct prompts. One solution to this issue would be to focus on reducing the size of the MDP state space and at the same time the size of the belief states, by merging similar MDP states based on the pattern of actions they are composed of. As explained in Section 5.2.2, the MDP state space is composed of all sequences of actions that are considered to be "valid", and which could lead to a successful completion of the task. Thus, there are redundancies within the state space, with states like ("add teabag into cup", "add water to the kettle") and ("add water to the kettle", "add teabag into cup") that could be merged together. This dimensionality leverage of the MDP state space may decrease the computational burden from the POMDP-based Task Manager side, which may improve its performance.

Figure 7.9 shows the main differences between SciMK and the Correlation distance. Consider the example of a *SimU* trying to make a "Black tea", and having performed the sequence of actions "add water in the kettle", "add teabag into cup", "boil water", "pour water from kettle to cup", "stir" (N.B., this history of actions is not observable from the Task Manager's point of view). When the observation of the last action of the *SimU* history is sent to the Task Manager, the latter updates its belief state, and generates *b-test*. In the example, this belief state is not contained in the belief subspace B, thus the POMDP-based APM must find a neighbor.
When the APM uses the Correlation distance to do so, it will tend to select a neighbor which has a similar probability distribution over the MDP states than *b-test*, see Figure 7.9 - (3). This is not the case for SciMK, see Figure 7.9 - (1). After belief state transformation using the technique described in Section 7.2, SciMK selects a neighbor based on the pattern of NL similarities, see Figure 7.9 - (2). As seen in Figures 7.6, 7.7, 7.8, SciMK allows the APM to output correct recommendations more often. In Figure 7.9, when using the Correlation distance, the APM outputs "stir", which is not a semantically correct recommendation, as the *SimU* has already done it, see Figure 7.9 - (4).

However, with SciMK, the APM outputs the optimal next best recommendation the *SimU* should follow based on its current state: "remove teabag". Note that if the *SimU* makes actions that are not semantically correct, this will not necessarily lead to an immediate failure (see Table 3.2). However, it increases the probability that the *SimU* may fail the task afterward.

When comparing the results showed in Figures 7.6, 7.7 and 7.8, with those depicted in Figures 7.3, 7.4 and 7.5, it is also possible to note that it is easier for the POMDP-based Task Manager to provide the appropriate recommendations, when it has access to the confusion matrices implemented by the virtual ARS, and when these confusion matrices are sparse (i.e., at the level of actions which are mandatory and cannot be repeated - see Table 3.2). For example, in "White tea with sugar", at 20% ARS error rate, with SciMK, the simulated user's success rate is 83% when the POMDP-based Task Manager has access to the ARS confusion matrix, and when the latter is sparse. By contrast, also with SciMK and at 20% ARS error rate, the simulated user's success rate is only 58% when the POMDP-based Task Manager has an approximation of the ARS confusion matrix, and when the latter is less sparse than the one used in 7.5.

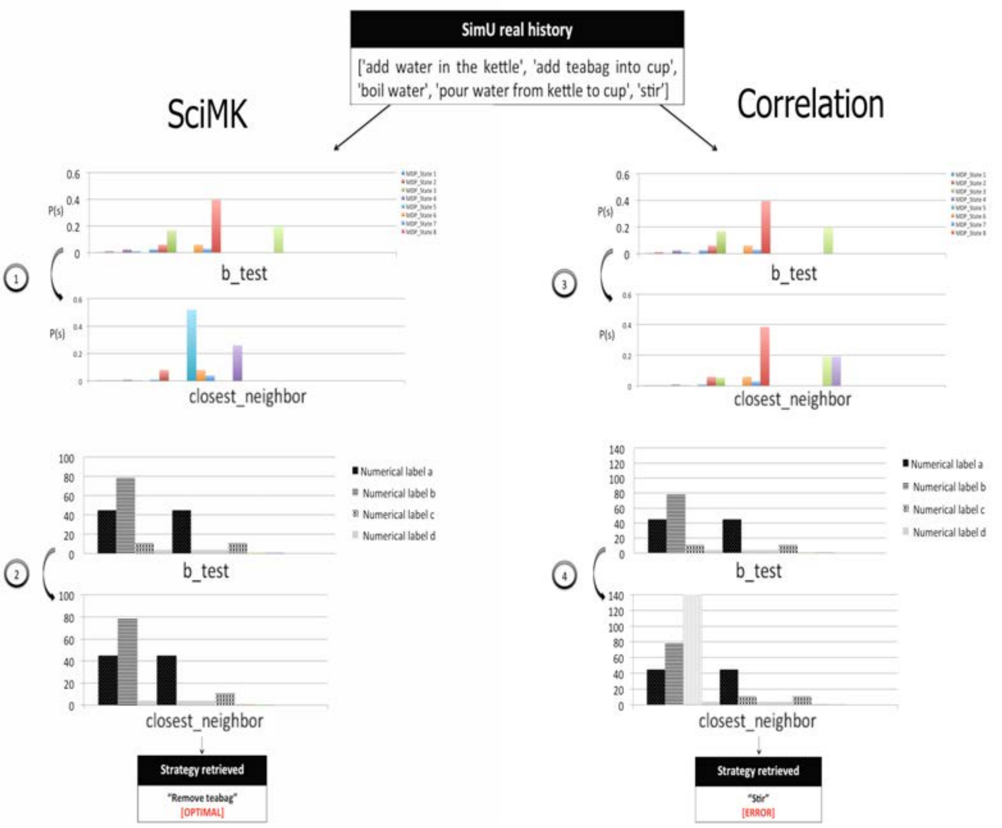


Figure 7.9: Comparison of neighbors selected by different NNS techniques. (1) and (3): Comparison of probabilistic distributions. (2) and (4) : Comparison of NLs vectors.

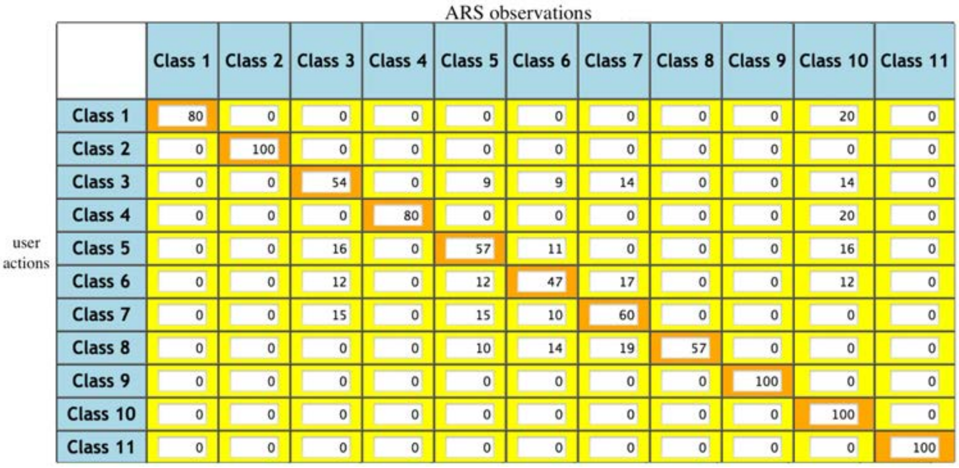
7.4.1.3 Third scenario

In order to show that, for a given belief state, SciMK does not come down to consider that $\pi^*(b) = \mu_p(\tilde{b})$, a third scenario was run. In this scenario, we made the assumption that the POMDP-based APM had an approximate understanding of the ARS's confusion matrix. The true confusion matrix of the ARS was as shown in Figure 7.10. The confusion matrix taken into account by the POMDP-based APM was as in Figure 7.11. These confusion matrices were chosen to simulate the case where the ARS makes more errors than what the POMDP-based APM believes it does (i.e., virtual ARS error rate: 24.1%, ARS error rate as seen by the Task Manager: 19.7%). This choice is also justified by the fact the confusion matrix of the real ARS is currently unknown, and the POMDP-based APM can only base its process on an approximation.

In this case, Figure 7.12 depicts the impact of the POMDP-based APM on the $SimU\alpha$ success rate during "Black tea with sugar", at varying NNS techniques. The technique which consists in considering that $\pi^*(b) = \mu_p(\tilde{b})$ (see Section 7.2) is referred to as the Most Likely State (MLS) policy [19]. With this heuristic, the state with the highest probability is found, and the APM selects the recommendation that would be optimal for that state in the underlying MDP.

We also implemented the method referred to as "Sum MLS". With this method, the probabilities of each similar recommendation in $\tilde{b}$ are summed, and the APM selects the recommendation that has the highest probability after the sum.

One can see that SciMK outperformed all other techniques. When implemented by the POMDP-based APM, SciMK allowed the POMDP-based APM to make the *SimU* succeed its task 68% of the time. By contrast, when "MSL" is implemented, the *SimU* succeeds only 52% of the time, and 62% of the time with "Sum MSL".

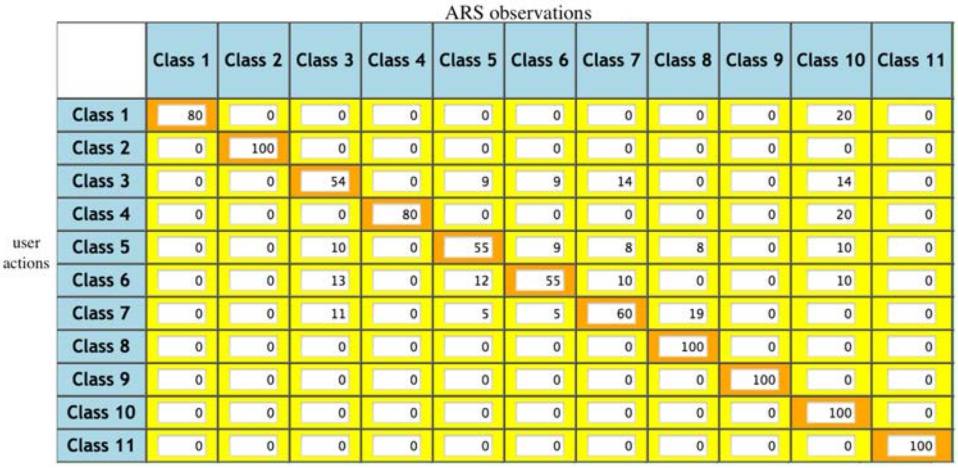


ARS observations

user actions

	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Class 10	Class 11
Class 1	80	0	0	0	0	0	0	0	0	20	0
Class 2	0	100	0	0	0	0	0	0	0	0	0
Class 3	0	0	54	0	9	9	14	0	0	14	0
Class 4	0	0	0	80	0	0	0	0	0	20	0
Class 5	0	0	16	0	57	11	0	0	0	16	0
Class 6	0	0	12	0	12	47	17	0	0	12	0
Class 7	0	0	15	0	15	10	60	0	0	0	0
Class 8	0	0	0	0	10	14	19	57	0	0	0
Class 9	0	0	0	0	0	0	0	0	100	0	0
Class 10	0	0	0	0	0	0	0	0	0	100	0
Class 11	0	0	0	0	0	0	0	0	0	0	100

Figure 7.10: Confusion matrix as implemented in the virtual ARS. ARS error rate 24.1%. Class 1: “Fill kettle”, Class 2: “Boil water”, Class 3: “Add teabag”, Class 4: “Pour kettle”, Class 5:“Add sugar”, Class 6:“Add milk”, Class 7: “Stir”, Class 8:“Remove teabag”, Class 9:“Pour water from jug to cup”, Class 10:“Toying with boiling water”, Class 11: ∅ (no action performed or detected).



ARS observations

user actions

	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Class 10	Class 11
Class 1	80	0	0	0	0	0	0	0	0	20	0
Class 2	0	100	0	0	0	0	0	0	0	0	0
Class 3	0	0	54	0	9	9	14	0	0	14	0
Class 4	0	0	0	80	0	0	0	0	0	20	0
Class 5	0	0	10	0	55	9	8	8	0	10	0
Class 6	0	0	13	0	12	55	10	0	0	10	0
Class 7	0	0	11	0	5	5	60	19	0	0	0
Class 8	0	0	0	0	0	0	0	100	0	0	0
Class 9	0	0	0	0	0	0	0	0	100	0	0
Class 10	0	0	0	0	0	0	0	0	0	100	0
Class 11	0	0	0	0	0	0	0	0	0	0	100

Figure 7.11: Confusion matrix seen by the POMDP-based APM. ARS error rate 19.7%. Class 1: “Fill kettle”, Class 2: “Boil water”, Class 3: “Add teabag”, Class 4: “Pour kettle”, Class 5:“Add sugar”, Class 6:“Add milk”, Class 7: “Stir”, Class 8:“Remove teabag”, Class 9:“Pour water from jug to cup”, Class 10:“Toying with boiling water”, Class 11: ∅ (no action performed or detected).

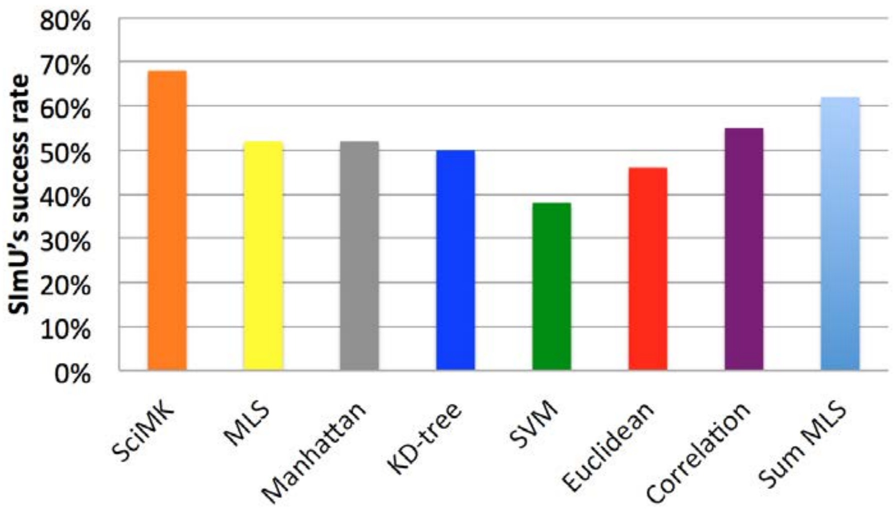


Figure 7.12: *SimU* success rate. ARS confusion matrix implemented Figure 7.10. ARS confusion matrix seen by POMDP-based APM: Figure 7.11. Task: “Black tea with sugar”.

7.5 Summary

In this chapter, the impact of different NNS techniques on the system's performance was analyzed. A specific metric-based algorithm referred to as SciMK was introduced and the results showed its ability to enable the POMDP-based APM to be more robust toward noise and task complexity than other techniques. During evaluations, we highlighted the fact that the choice of the metric used during NNS has an impact on the POMDP-based APM's performance. Moreover, we saw that the dimensionality of the belief states in complex task such as "White tea with sugar" make it hard for the POMDP-based APM to plan under uncertainty. We discussed the possibility of applying a second MDP state space reduction technique in order to potentially tackle this issue.

Chapter 8

Conclusion

8.1 Thesis summary

This thesis has examined the challenge of implementing an artificial intelligent planning module, referred to as “Task Manager”, in an assistive/rehabilitation system for cognition named CogWatch. CogWatch was designed to provide instructional cues to stroke survivors during their activities of daily living. In order to fulfill its goal, the system observes the user during a given task, detects the actions made by the user via an Action Recognition System (ARS), and infers what cues to send in order to assist him or her. In this work, we focused on how a Task Manager can be modeled and implemented in such a system, so it can guide users while they are preparing hot drinks. Hence, the core of this thesis was the Task Manager.

In CogWatch, the aims of the Task Manager are to find what are the best actions the user should follow (i.e., recommendations) in order to successfully continue or finish a task, and to detect when the user's behavior is erroneous. We examined different models the Task Manager could be based on, discussed how these models could be implemented, then specifically focused on the implementation of a Markov Decision Process (MDP) and Partially Observable Markov Decision Process (POMDP) based Task Manager.

MDPs and POMDPs provide a rich framework for sequential action/recommendation-planning. Contrary to the MDP, the POMDP can model uncertainty in the recommendations effects and uncertainty in the user's environment. In realistic situations where uncertainties are taken into account, POMDPs are known to be more advantageous than MDPs. In the case of CogWatch, realistic situations refer to the fact that the system's ability to make mistakes is taken into account. Indeed, when observing a user during a task, CogWatch may generate a wrong interpretation of the user's behaviors and send this false information to the Task Manager. Whether the information received is correct or incorrect, the Task Manager's goal remains the same: it needs to properly guide users, even under uncertainty. In such a case, the POMDP-based Task Manager is expected to perform better than the MDP-based Task Manager. The work presented in this thesis explained how both MDP and POMDP models can be adapted to take into account the specificities of a system like CogWatch, and how the POMDP can be solved via a grid-based approach.

When implementing a POMDP-based Task Manager, the latter maintains a probability distribution over states, called "belief state" during each interaction with the user. Each time the Task Manager receives new information about the user's behavior, it updates its belief state, and at the same time its interpretation of what the user has achieved so far in the task. This thesis proposed an efficient state and belief state representation based on the underlying MDP state space.

One of the goals of the Task Manager (MDP and POMDP) is to find the optimal recommendation the user should follow through the task. In the case of the POMDP, recommendation selection is based on each belief state updated, and is determined by the policy found by the Task Manager after training. This process of learning the optimal policy is called "policy optimization". A grid-based approach was implemented in order to solve the POMDP and obtain such a policy. It is an approximate method, which allows the system to learn how to act optimally only for a set of belief states. In other words, the method allows to find a mapping between a finite set of belief states and optimal recommendations. Many other approaches and approximate methods exist, such as point-based algorithms (see for example [83]). Contrary to point-based algorithms, the approach chosen may rely on Nearest Neighbor Search (NNS) techniques in order estimate the appropriate recommendation to select for a belief state not part of the mapping obtained after training. Different NNS techniques were implemented in order to analyze their impact on the POMDP-based Task Manager's performance when

interacting with a simulated user. This thesis also proposed and detailed how simulated users can be designed based on real user's data, and how a virtualization of the whole system can be built in order for this simulated user to interact with it.
Another goal of the Task Manager (MDP or POMDP) is to detect errors in the user's behavior. A rule-based approach was applied to the MDP-based Task Manager, taking into account the definition of errors users with cognitive deficits tend to make during activities of daily living. In the case of the POMDP-based Task Manager, an algorithm was proposed to enable error detection under uncertainty. Both methods were compared during a specific task.

Results showed that stroke survivors who had difficulties going through their tasks by themselves, made fewer errors when they were guided by CogWatch. The limitations of the MDP-based system under uncertainty were analyzed. As expected, results showed that most of the time, the POMDP-based system has a better ability to cope with uncertainty in its inputs than the MDP-based system. Thus, the focus was put on the impact of NNS techniques on the POMDP-based system performance. We found that the novel algorithm presented in this thesis constantly outperformed the other methods it has been compared with. This novel algorithm, referred to as "SciMK", allowed the POMDP-based Task Manager to select appropriate recommendations more often than other methods, at increasing ARS error rate.

8.2 Open problems and Future work

This research can be extended in different directions:

- The MDP-based Task Manager was evaluated with real participants and via user simulation, while the POMDP-based Task Manager was evaluated via user simulation only. This was due to the unavailability of real participants when the POMDP-based system was implemented. Thus, a part of future work should be dedicated to the evaluation of the POMDP-based system with stroke survivors. The main advantage of evaluation via simulation was the possibility to go through an extensive analysis of the POMDP-based Task Manager's performance at varying Action Recognition System's error rates and NNS techniques.

- A grid-based approximation approach was taken to solve the POMDP, and different NNS techniques were implemented in order analyze the limits of this method at varying ARS error rates. It could also be interesting to compare our grid-based approximation approach to point-based approximation approaches (for example, PBVI [83] or Perseus [103]) in the context of CogWatch.

- In this work, we discussed about the fact that the confusion matrix of the real ARS is unknown. To simulate this situation, the POMDP-based Task Manager was evaluated taking into account an approximation of this confusion matrix. Currently, all confusion matrices assume that when the user does not make an action, no observation is output by the ARS. Hence, another matter for future research could be related to the ability of the Task Manager to act when the user makes an action, but the ARS outputs no observation. One solution could be to model timing in users' behaviors and allow the Task Manager to detect when an action is more likely to have been performed even if the ARS does not communicate any information about it.

- In Chapter 5, we discussed the current impossibility to robustly compare the performance of stroke survivors with the performance of the simulated users implemented in the virtual CogWatch system. This matter could be tackled in the future after running new experiments, which could give some insights about the validity of the methodology used to design the simulated users and the virtual environment. In this case, the notion of real and virtual users' performance could be measured by taking into account the number and type of errors made during the tasks, and users' success rate with and without the CogWatch system. For a robust analysis, both real users and simulated users should go through the same experiments. For example, real and simulated users should perform the same number of trials during evaluation. They should also share a common level of compliance toward the system's recommendations in order to measure the system's impact of their success rate. Moreover, both simulated and real users should interact with the same configuration of the CogWatch system (i.e., same ARS error rate, same EL or EF technique applied).

- Currently, the recommendations the Task Manager can provide correspond to the top level of the actions tree depicted in Figure 2.1. In the future, it would be interesting to increase the systems recommendations level of detail. It would then allow the system to provide more specific information about what the user should do to correct his or her behavior. For example, in the case where the system would detect that the user did not pour enough water in the cup, the system would then retrieve nuanced recommendations such as Add some more water. We can also imagine the case where the ARS would provide more detailed information about the users actions, which would allow the Task Manager to detect more specific errors (for example, when the user holds an object with an incorrect grip).

- A more efficient algorithm could be implemented in order to reduce the dimensionality of the MDP state space. Currently the state space reduction technique focuses on not allowing incorrect sequences of actions to be part of the state space. However, even with this technique, when the number of mandatory actions to be made during a task increases, the size of the state space may increase rapidly. This is due to the fact that some mandatory actions such as “Add sugar” or “Add milk” in “White tea with sugar” can be performed at any moment during the task. Indeed, in such a task, these actions have no sequential constraint (for example it is possible to add milk at the very end, or when the cup is still empty). Currently, the technique implemented takes into account all the possible combinations of correct sequences of actions, even those which are equal as far as the completion of the task is concerned (for example, (“Add water in the kettle”, “Boil water”, “Add milk”) and (“Add water in the kettle”, “Add milk”, “Boil water”)). We can imagine an extended technique that would take into account actions which are mandatory and have no sequential constraint. For example, this new technique would check whether actions such as “Add teabag”, “Add sugar” or “Add milk” are part of the user’s state during tasks where they are mandatory, without keeping any information about when they have been performed. Indeed, this specific information does not affect whether the trials are correct or not.

- Finally, it would be interesting to compare the performance of the CogWatch Task Manager with other assistive systems’ Task Manager, such as COACH [14] and TEBRA [79]. Indeed, CogWatch, COACH and TEBRA, all focus on sequential activities of daily living, and aim to provide appropriate guidance to users with cognitive impairments. They also share a similar architecture as described in Figure 1.1. For example, TEBRA and CogWatch used a camera-based monitoring system, and sensorized objects in the users’ environment in order to capture information about their behavior during the tasks [73; 79]. The latest versions of COACH [36] and CogWatch both implement a POMDP-based Task Manager in order to take into account uncertainties related to the users actions (see Chapter 6). Each system also has its own specificities that make them unique. For example, in COACH and CogWatch the cue generation module (see Figure 1.1) is a stand-alone component of the system, while in TEBRA, it has been designed as a sub-component of the Task Manager [14; 79].

However, more important and intrinsic differences exists between these systems, which currently make the comparison of their Task Manager practically infeasible. Indeed, CogWatch, TEBRA and COACH have been implemented to model different tasks (i.e., tea-making, teeth-brushing and hand-washing); and their performance during these tasks have been assessed in different ways by their authors. Moreover, the Task Manager performance highly depends on the system's ARS error rate, which is also different in all systems. Hence, in order to compare the Task Manager developed for CogWatch with those designed for other systems, a common specific task should be chosen (i.e., teeth-brushing, hand-washing or tea-making), and the code related to all other components part of the systems should be made available, so the CogWatch Task Manager could be implemented in the other system's environment before being evaluated by users with similar cognitive deficits. The development of a common framework will make it possible to have a better understanding of the strengths and limitations of assistive systems for cognition, and help in the improvement of novel techniques that could be beneficial in the field.

References

[1] M.R. Abbasifard, B. Ghahremani, and H. Naderi. Article: A survey on nearest neighbor search methods. *International Journal of Computer Applications*, 2014. 129, 131

[2] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the surprising behavior of distance metrics in high dimensional space. In *Lecture Notes in Computer Science*, 2001. 129

[3] J.L. Ambite and C.A. Knoblock. Planning by rewriting. *Journal of Artificial Intelligence Research*, 2001. 6

[4] Stroke Association. The Stroke Association website, 2014. 1

[5] H. Baier and M.H.M. Winands. Nested Monte-Carlo tree search for online planning in large MDPs. In *ECAI*, Frontiers in Artificial Intelligence and Applications, pages 109–114, 2012. 34

[6] C. Barbieri and E. De Renzi. The executive and ideational components of apraxia. *Cortex.*, 1988. 1

[7] R. Bellman. Dynamic programming. *Princeton University Press, Princeton, NJ*, 1957. 30, 31, 33

[8] J.L Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975. 132

[9] D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 3rd edition, 2005. 31

[10] N. Bhatia and V. Ashev. Survey of nearest neighbor techniques. *International Journal of Computer Science and Information Security*, 2010. 129

[11] W. Bickerton, M.J. Riddoch, D. Samson, A. Balani, B. Mistry, and G.W. Humphreys. Systematic assessment of apraxia and functional predictions from the birmingham cognitive screen. *Journal of Neurology, Neurosurgery, and Psychiatry*, 2012. 1

[12] W.L Bickerton, G.W. Humphreys, and M. J. Riddoch. The use of memorised verbal scripts in the rehabilitation of action disorganisation syndrome. *Neuropsychological Rehabilitation*, 2010. 61

[13] M. Bienkiewicz, G. Goldenberg, J.M. Cogollor, M. Ferre, C. Hughes, L. Mary, and J. Hermsdörfer. Use of biological motion based cues and ecological sounds in the neurorehabilitation of apraxia. In *HEALTHINF*, 2013. 61

[14] J. Boger, J. Hoey, P. Poupart, C. Boutilier, G. Fernie, and A. Mihailidis. A planning system based on markov decision processes to guide people with dementia through activities of daily living. *IEEE Transactions on Information Technology in Biomedicine*, 2006. 6, 12, 14, 156

[15] M. Botvinick and D.C. Plaut. Doing without schema hierarchies: A recurrent connectionist approach to normal and impaired routine sequential action. *Psychological Review*, 2004. xiv, 24, 26

[16] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 1999. 12

[17] R.I. Brafman. A heuristic variable grid solution method for POMDPs. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence*, 1997. 44, 45, 46, 48, 109, 129

[18] A. Di Carlo. Human and economic burden of stroke. *Age and ageing*, pages 4–5, 2009. 2

[19] A.R. Cassandra, L.P. Kaelbling, and J.A. Kurien. Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. In *In Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1996. 147

[20] J.M. Cogollor, C. Hughes, M. Ferre, J. Rojo, J. Hermsdörfer, A. Wing, and S. Campo. Handmade task tracking applied to cognitive rehabilitation. *Sensors*, 2012. 57

[21] Stroke Unit Trialists' Collaboration. Organised inpatient (stroke unit) care for stroke. *Cochrane Database of Systematic*, 2007. 2

[22] R. Cooper. Order and disorder in everyday action: the roles of contention scheduling and supervisory attention. *Neurocase*, 2002. iii, xiv, 24, 25, 26, 36

[23] R. Cooper and T. Shallice. Contention scheduling and the control of routine activities. *Cogn Neuropsychol.*, 2000. iii, 24, 36

[24] H. Cuayahuitl, S. Renals, and O. Lemon. Learning multi-goal dialogue strategies using reinforcement learning with reduced state-action spaces. In *Proceedings of Interspeech*, 2006. 16

[25] L. Cunningham, S. Mason, C. Nugent, G. Moore, D. Finlay, and D. Craig. Home-based monitoring and assessment of parkinson's disease. *IEEE Transactions on Information Technology in Biomedicine*, 2011. 5

[26] H-W. Gellersen, M. Beigl, and H. Krull. The mediacup: awareness technology embedded in an everyday object, 1999. 57

[27] K. Georgila, J. Henderson, and O. Lemon. Learning user simulations for information state update dialog systems. In *Proceedings of Eurospeech*, 2005. 45, 76

[28] K. Georgila, J. Henderson, and O. Lemon. User simulation for spoken dialogue systems: Learning and evaluation. In *Proceedings of Interspeech*, 2006. 76

[29] A. Gillespie, C. Best, and B. O'Neill. Cognitive function and assistive technology for cognition: A systematic review. *International Neuropsychological Society*, pages 1–19, 2011. 6

[30] G. Goldenberg. Apraxia - The cognitive side of motor control. *Cortex*, 57:270 – 274, 2014. 2

[31] G. Goldenberg, M. Daumuller, and S. Hagmann. Assessment and therapy of complex activities of daily living in apraxia. *Neuropsychological Rehabilitation*, 2001. 1

[32] T. Hart, T. O'Neil-Pirozzi, and C. Morita. Clinician expectations for portable electronic devices as cognitive-behavioural orthoses in traumatic brain injury rehabilitation. *Brain Injury*, 17(5):401–411, 2003. 6

[33] J. Hermsdörfer, M. Bienkiewicz, M. Cogollor, J.M. Russell, E. Jean-Baptiste, M. Parekh, A.M Wing, M. Ferre, and C. Hugues. Cogwatch automated assistance and rehabilitation of stroke-induced action disorders in the home environment. In *Proceedings of HCI Aspects of Optimal Healing Environments*, 2013. xiv, 56, 57

[34] A. Hinneburg, C. C Aggarwal, and D.A. Keim. What is the nearest neighbor in high dimensional spaces? In *Proceedings of the 26th International Conference on Very Large Data Bases*, 2000. 129

[35] J. Hoey. Tracking using flocks of features, with application to assisted handwashing. In *Proceedings BMVC*, 2006. 10

[36] J. Hoey, P. Poupart, A. von Bertoldi, T. Craig, C. Boutilier, and A. Mihailidis. Automated handwashing assistance for persons with dementia using video and a partially observable markov decision process. *Computer Vision and Image Understanding*, pages 503–519, 2010. 7, 13, 156

[37] J. Hoey, A. von Bertoldi, P. Poupart, and A. Mihailidis. Assisting persons with dementia during hand-washing using a partially observable Markov decision process. In *Proceedings of the Int. Conf. on Vision Systems (ICVS)*, 2007. 106

[38] http://scikit learn.org/. Scikit-learn, 2014. 131

[39] C.M. Hughes, C. Baber, M. Bienkiewicz, and J. Hermsdörfer. Application of human error identification (HEI) techniques to cognitive rehabilitation in stroke patients with limb apraxia. *Applications and Services for Quality of Life*, 2013. 22

[40] Y. Huizhen. *Approximate Solution Methods for Partially Observable Markov and Semi-Markov Decision Processes.* PhD thesis, Massachusetts institute of technology, 2006. 28, 36

[41] G.W. Humphreys and E.M.E. Forde. Disordered action schema and action disorganisation syndrome. *Cognitive Neuropsychology*, 1998. 22

[42] G.W. Humphreys, M. Wulff, E.Y. Yoon, and M.J. Riddoch. Neuropsychological evidence for visual- and motor-based affordance: effects of reference frame and object-hand congruence. *Exp Psychol Learn Mem Cogn.*, 2010. 61

[43] D. Hyndman and A. Ashburn. People with stroke living in the community: Attention deficits, balance, ADL ability and falls. *Disability and Rehabilitation*, pages 817–822, 2003. 2

[44] E.M.D Jean-Baptiste, J. Howe, , P. Rotshtein, and M. Russell. Cogwatch: Intelligent agent-based system to assist stroke survivors during tea-making. In *Proceedings of the IEEE/SAI Intelligent Systems Conference*, 2015. 16

[45] E.M.D Jean-Baptiste, J. Howe, P. Rotshtein, and M. Russell. Intelligent agent-based system to assist stroke survivors. *Submitted to the Journal of Ambient Intelligence and Smart Environments*, 2015. 16

[46] E.M.D Jean-Baptiste, R. Nabiei, M. Parekh, E. Fringi, B. Drozdowska, C. Baber, P. Jancovic, P. Rotshein, and M. Russell. Intelligent assistive system using real-time action recognition for stroke survivors. In *Proceedings of the IEEE International Conference on Healthcare Informatics (ICHI)*, 2014. 16

[47] E.M.D Jean-Baptiste, P. Rotshein, and M. Russell. POMDP based action planning and human error detection. In *Proceedings of the 11th International Conference on Artificial Intelligence Applications and Innovations (AIAI2015)*, 2015. 16, 110

[48] E.M.D Jean-Baptiste, P. Rotshtein, and M. Russell. Cogwatch: Automatic prompting system for stroke survivors during activities of daily living. *Submitted to the Journal of Innovation in Digital Ecosystems*, 2016. 16

[49] E.M.D Jean-Baptiste, M.J. Russell, and P. Rothstein. Assistive system for people with apraxia using A markov decision process. In *MIE*, Studies in Health Technology and Informatics, pages 687–691, 2014. 16

[50] D. Jurafsky and J.H. Martin. Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition, 1st ed. *Upper Saddle River, NJ, USA: Prentice Hall PTR*, 2000. 14

[51] J.Westin, M. Dougherty, D. Nyholm, and T. Groth. A home environment test battery for status assessment in patients with advanced Parkinson's disease. *Comput. Methods Prog. Biomed.,*, 2010. 5

[52] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 1998. iii, 40, 41, 42, 44, 108

[53] L. P. Kaelbling, M.L. Littman., and A.W. Moore. Reinforcement learning: A survey. *J. Artif. Int. Res.*, 1996. 28

[54] S. Katz, A. B. Ford, R. W. Moskowitz, B. A. Jackson, and M. W. Jaffe. Studies of illness in the aged: The index of ADL: A standardized measure of biological and psychosocial function. *Journal of the American Medical Association*, 1963. ii

[55] H. Kautz, O. Etzioni, D. Fox, and D. Weld. Foundations of assisted cognition systems, 2003. 5

[56] E.F. Krause. *Taxicab geometry: an adventure in non-Euclidean geometry*. Dover Publ., 1987. 131

[57] P. Langhorne and L. Legg. Evidence behind stroke rehabilitation. *Journal of Neurology, Neurosurgery & Psychiatry*, 2003. 2

[58] K.S. Lashley. The problem of serial order in behavior. *Cerebral mechanisms in behavior*, 1951. 22

[59] Y. Lee, Y. Lin, and G. Wahba. Multicategory support vector machines. *Journal of the American statistical association*, pages 67–81, 2001. 131

[60] O. Lemon and O. Pietquin. Machine learning for spoken dialogue systems. In *Proceedings of Interspeech*, 2007. 18

[61] E. Levin, R. Pieraccini, and W. Eckert. A stochastic model of human-machine interaction for learning dialog strategies. *Proceedings of the IEEE transactions on Speech and Audio Processing*, 2000. 27, 29, 34, 45, 76

[62] M.L Littman. *Algorithms for Sequential Decision Making*. PhD thesis, 1996. 29

[63] K. Liu, C. Chen, R. Jafari, and N. Kehtarnavaz. Multi-hmm classification for hand gesture recognition using two differing modality sensors. In *Proceedings of the 10th IEEE Dallas Circuits and Systems Conference (DCAS'14)*, 2014. 58

[64] E. F LoPresti, A. Mihailidis, and N. Kirsch. Assistive technology for cognitive rehabilitation: State of the art. *Neuropsychological Rehabilitation*, 2004. 6

[65] O. Madani, H. Steve, and A. Condon. On the undecidability of probabilistic planning and infinite-horizon Partially Observable Markov Decision Problems. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, pages 541–548, 1999. 44

[66] D. L. McLellan. Functional recovery and the principles of disability medicine. *Clinical Neurology*, 1991. 2

[67] M. McTear. Spoken dialogue technology. *Springer*, 2004. 14

[68] E.L. Middleton and M.F. Schwartz. Errorless learning in cognitive rehabilitation: a critical review. *Neuropsychological Rehabilitation*, pages 138–168, 2012. 2

[69] A. Mihailidis, J. C. Barbenel, and G. Fernie. The efficacy of an intelligent cognitive orthosis to facilitate handwashing by persons with moderate to severe dementia. *Neuropsychological Rehabilitation*, pages 135–171, 2004. 10

[70] A. Mihailidis, B. Carmichael, and J. Boger. The use of computer vision in an intelligent environment to support aging-in-place, safety, and independence in the home. *Trans. Info. Tech. Biomed.*, pages 238–247, 2004. 6

[71] J. Mihailidis, J. N. Boger, T. Craig, and J. Hoey. The coach prompting system to assist older adults with dementia through handwashing: An efficacy study. *BMC Geriatrics*, 2008. 7, 11, 13

[72] K. Morady and G.W. Humphreys. Comparing action disorganization syndrome and dual-task load on normal performance in everyday action tasks. *Neurocase*, pages 1–12, 2009. 1

[73] R. Nabiei, M. Parekh, E. Jean-Baptiste, P. Jančovič, and M.J. Russell. Object-centred recognition of human activity for assistance and rehabilitation of stroke patients. In *Submitted to IEEE Int. Conf. On Healthcare Informatics (ICHI 2015), Dallas, TX, USA*, 2015. xiv, 58, 59, 156

[74] A.M. Nicol, C. Gee Bush, and E. Balka. Internet devices and desires: A review of randomized controlled trials of interactive, Internet-mediated, in-home, chronic disease monitoring programs. *J. Res. Interprofessional Pract. Edu.*, 2009. 5

[75] M. Parekh and C. Baber. Tool use as gesture: New challenges for maintenance and rehabilitation. In *Proceedings of the 24th BCS Interaction Specialist Group Conference*, 2010. 57

[76] Intercollegiate Stroke Working Party. Royal college of physicians national sentinel stroke clinical audit 2010 round 7 public report for England, Wales and Northern Ireland. Technical report, 2010. 1

[77] K. Pearson. Notes on the history of correlation. *Biometrika*, pages 25–45, 1920. 131

[78] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel., B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 131

[79] C. Peters, T. Hermann, and S. Wachsmuth. TEBRA - an automatic prompting system for persons with cognitive disabilities in brushing teeth. In *Proceedings of the 6th International Conference on Health Informatics (HealthInf)*, pages 12–23, 2013. 8, 10, 11, 13, 14, 156

[80] C. Peters, T. Hermann, S. Wachsmuth, and J. Hoey. Automatic task assistance for people with cognitive disabilities in brushing teeth - a user study with the tebra system. *ACM Trans. Access. Comput.*, 2014. 8, 14

[81] B. Petreska, M. Adriani, O. Blanke, and A. G. Billard. Apraxia: a review. In *From Action to Cognition*, volume 164 of *Progress in Brain Research*, pages 61 – 83. 2007. 1

[82] O. Pietquin and T. Dutoit. A probabilistic framework for dialog simulation and optimal strategy learning. *IEEE Transactions on Speech and Audio Processing, Special Issue on Data Mining of Speech, Audio and Dialog*, 2006. 76

[83] J. Pineau, G. Gordon, and T. Sebastian. Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the 18th International*

Joint Conference on Artificial Intelligence, IJCAI'03, 2003. 44, 45, 117, 152, 154

[84] M.E. Pollack. Autominder : A case study of assistive technology for elders with cognitive impairment. *Generations*, 2006. 6

[85] I.M. Proot, H.F. Crebolder, H.H. Abu-Saad, T.H. Macor, and R.H. Ter Meulen. Facilitating and constraining factors on autonomy: The views of stroke patients on admission into nursing homes. *Clinical Nursing Research*, 2000. 2

[86] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley series in probability and mathematical statistics. 1994. iii, 12, 28

[87] S. Quarteroni, M. Gonzàlez, G. Riccardi, and S. Varges. Combining user intention and error modeling for statistical dialog simulators. In *Proceedings of Interspeech*, 2010. 45, 76

[88] L.R. Rabiner. Readings in speech recognition. chapter A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. 1990. 58

[89] V. Rieser and O. Lemon. Cluster-based user simulations for learning dialogue strategies. In *Proceedings of Interspeech*, 2006. 45, 76

[90] V. Rieser and O. Lemon. *Reinforcement Learning for Adaptive Dialogue Systems*. Theory and Applications of Natural Language Processing. Springer, 2011. 33, 75

[91] T.T. Rogers and J.L. McClelland. Parallel distributed processing at 25: Further explorations in the microstructure of cognition. *Cognitive Science*, pages 1024–1077, 2014. 26

[92] M. Roth. *Execution-time Communication Decisions for Coordination of Multi-agent Teams*. PhD thesis, 2007. 39

[93] N. Roy, J. Pineau, and S. Thrun. Spoken dialogue management using probabilistic reasoning. In *Proceedings of ACL*, 2000. 48

[94] M. Rudary, S. Singh, and M. E. Pollack. Adaptive cognitive orthotics: Combining reinforcement learning and constraint-based temporal reasoning. In *Proceedings of the 21st International Conference on Machine Learning (ICML 2004)*, 2004. 6

[95] J. Schatzmann. Statistical User and Error Modelling for Spoken Dialogue Systems, 2008. 76

[96] J. Schatzmann, K. Weilhammer, M. Stuttle, and S. Young. A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *KER*, 2006. 45, 76

[97] J. Schatzmann and S. Young. The hidden agenda user simulation model. *IEEE Transactions on Speech and Audio Processing*, 2009. 76

[98] K. Scheffler and S. Young. Probabilistic simulation of human-machine dialogues. In *Proceedings of ICASSP*, 2000. 76

[99] K. Scheffler and S. Young. Corpus-based dialogue simulation for automatic strategy learning and evaluation. In *Proceedings of NAACL*, 2001. 45

[100] M. F. Schwartz, E.S. Reed, M. Montgomery, C. Palmer, and N.H. Mayer. The quantitative description of action disorganisation brain damage: A case study. *Cognitive Neuropsychology*, 1991. 52, 53

[101] M.C. Silveri and N. Ciccarelli. Semantic memory in object use. *Neuropsychologia*, pages 2634 – 2641, 2009. 2

[102] E. Sondik. *The Optimal Control of Partially Observable Markov Processes.* PhD thesis, Stanford University, 1971. 42, 44

[103] M. Spaan, T.J. Matthijs, and N. Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *J. Artif. Int. Res.*, 2005. iii, 154

[104] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction. Adaptive Computation and Machine Learning.* MIT Press,Cambridge, MA, 1998. xiv, 30, 31, 32, 33, 34, 45, 107

[105] B. Thomson and S. Young. Bayesian update of dialogue state: A POMDP framework for spoken dialogue systems. *Computer Speech and Language*, pages 562–588, 2010. 45, 77

[106] S. Thrun. Monte carlo POMDPs. In *Advances in Neural Information Processing Systems (NIPS 1999)*, pages 1064–1070, 2000. 28

[107] W.H. Turkett. *Robust Multiagent Plan Generation and Execution with Decision Theoretic Planners*. PhD thesis, University of South Carolina, 1998. 48

[108] C. M. van Heugten, J. Dekker, B. G. Deelman, A. J. van Dijk, and J. C. Stehmann-Saris. Outcome of strategy training in stroke patients with apraxia: a phase ii study. *Clinical Rehabilitation*, pages 294–303, 1998. 2

[109] V. Vapnik. *Statistical learning theory*. 1998. 131

[110] C.J.C.H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, 1989. 31

[111] K. Q. Weinberger and L. K. Saul Lawrence. Distance metric learning for large margin nearest neighbor classification. *J. Mach. Learn. Res.*, 2009. 129, 131

[112] M. M. De Werd, D. Boelen, M. G. O. Rikkert, and R. P. Kessels. Errorless learning of everyday tasks in people with dementia. *Clinical Interventions in Aging*, 2013. 2

[113] J.D Williams. Applying POMDPs to dialog systems in the troubleshooting domain. In *Proc HLT/NAACL Workshop on Bridging the Gap: Academic and Industrial Research in Dialog Technology, Rochester, NY, USA*, 2007. 77

[114] J.D Williams, P. Poupart, and S. J. Young. Factored partially observable markov decision processes for dialogue management. In *Proceedings Workshop on Knowledge and Reasoning in Practical Dialog Systems, Int. Joint Conf. on Artificial Intelligence (IJCAI), Edinburgh*, 2005. 38, 48, 50

[115] S. Young, M. Gasic, S. Keizer, F. Mairesse, J. Schatzmann, B. Thomson, and K. Yu. The hidden information state model: a practical framework for POMDP-based spoken dialogue management. *Computer Speech and Language*, pages 150–174, 2010. 15, 19, 27, 41, 45, 48, 77, 111

[116] B. Zhang, Q. Cai, J. Mao, and B. Guo. Planning and acting under uncertainty: A new model for spoken dialogue system. In *Proceedings of Conference on Uncertainty in Artificial Intelligence*, 2001. 48

[117] A. Zwinkels, C. Geusgens, P. van de Sande, and C. Van Heugten. Assessment of apraxia: inter-rater reliability of a new apraxia test, association between apraxia and other cognitive deficits and prevalence of apraxia in a rehabilitation setting. *Clinical Rehabilitation*, 2004. 1