Kingston University London

Real-time Optimization of Working Memory in Autonomous Reasoning for High-level Control of Cognitive Robots deployed in Dynamic Environments

---

Author:

Deon de Jager

This thesis being submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

School of Engineering and the Environment, Kingston University London

May 31, 2020

Declaration of Authorship

I hereby declare that this thesis and the work presented in it are my own. I confirm that: this work was done wholly while in candidature for a research degree at this University and have not been submitted in whole or in part for consideration for any other qualification in this, or any other University. Where I have consulted the published work of others, this is always clearly attributed. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.

Signed:

Date: 31 May 2020

Abstract

High-level, real-time mission control of autonomous and semi-autonomous robots, deployed in remote and dynamic environments, remains a research challenge. Robots operating in these environments require some cognitive ability, provided by a simple, but robust, cognitive architecture. The most important process in a cognitive architecture is the working memory, with core functions being memory representation, memory recall, action selection and action execution, performed by the central executive. The cognitive reasoning process uses a memory representation, based on state flows, governed by state transitions with simple, quantified propositional transition formulae. In this thesis, real-time working memory quantification and optimization is performed using a novel adaptive entropy-based fitness quantification (AEFQ) algorithm and particle swarm optimization (PSO), respectively. A cognitive architecture, using an improved set-based PSO is developed for real-time, high-level control of single-task robots and a novel coalitional games-theoretic PSO (CG-PSO) algorithm extends the cognitive architecture for real-time, high-level control in multi-task robots. The performance of the cognitive architecture is evaluated by simulation, where a UAV executes four use cases: Firstly, for real-time high-level, single-task control: 1) relocating the UAV to a charging station and 2) collecting and delivering medical equipment. Performance is measured by inspecting the success and completeness of the mission and the accuracy of autonomous flight control. Secondly, for real-time high-level control of multi-task autonomous vehicle control: 3) delivering medical equipment to an incident and 4) provide aerial security surveillance support. The performance of the architecture is measured in terms of completeness and cognitive processing time and cue processing time. The results show that coalitions correctly represent optimal memory and action selection in real-time, while overall processing time is within a feasible time limit, arbitrarily set to 2 seconds in this study.

Acknowledgements

I would like to express my sincere gratitude and appreciation to my supervisors, for their unconditional guidance and support throughout this journey. I would like to thank Dr. Yahya Zweiri, for his valuable advice, endless patience and teaching me the important points of research. I would also like to thank Prof. Dimitrios Makris, for his advice, guidance and teaching me to always look at things from different points of view. I am privileged to have worked with Dr. Zweiri and Prof. Makris, and I truly believe I have become a better researcher, as a result.

I would like to thank my sons, Robert and Jacques, for their constant encouragement and moral support throughout this journey. I would particularly like to express my appreciation to my son Robert, for his valuable assistance in the programming of my simulation environment.

I dedicate this thesis to my wife Vicky, for her unwavering support and endless patience throughout this journey. Thank you for always being there, listening to my arguments and explanations. Without your support, this thesis would not have been possible.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial intelligence
AEFQ	Adaptive Entropy Fitness Quantification
AESet-PSO	Adaptive Entropy Set-based Particle Swarm Optimization
AEStd-PSO	Adaptive Entropy Standard Particle Swarm Optimization
AM	Activated Memory
CE	Central Executive
CG-PSO	Coalitional Game Theory-based Particle Swarm Optimization
CPt	Cognitive Processing time
CRP	Cognitive Reasoning Process
EM	Episodic Memory
ENV	Environmental Stimulus
FOA	Focus Of Attention
LTM	Long-Term Memory
MPt	Maximum Processing time
PSO	Particle Swarm Optimization
SPSO	Set-based Particle Swarm Optimization
STM	Short-Term Memory
TPt	Task Processing time
UAV	Unmanned Aerial Vehicle
WM	Working Memory

List of Equations

Chapter 1

Introduction

The application of artificial intelligence (AI) for the high-level control of autonomous vehicles (autonomous vehicles) holds a lot of promise. The ultimate objective is to deploy an autonomous vehicle and leave it to intelligently and successfully perform the tasks it was designed and equipped for [1, 2]. Unfortunately, such level of autonomy is proving hard to accomplish, especially for robots deployed in remote and dynamic environments. For example, consider the Mars rover, Curiosity. Currently, the rover is exploring the surface of Mars. The rover is equipped with a number of on-board scientific testing laboratories, for example, a Sample Analysis at Mars (SAM), Mast Camera (MASTCAM), Mars Hand Lens Imager (MAHLI) and Rover Environmental Monitoring Station (REMS), amongst others. While the rover is equipped with some level of low-level autonomous control (mostly for obstacle avoidance), a human operator or designer, i.e. a domain expert, is still required to control some mission-specific tasks such as, sampling, testing and analysis. This includes the decision about whether an area is interesting and should be explored further. This is high-level mission control and currently, there is a gap in solutions for autonomous AI for low-level control, such as stabilization control [3, 4], and high-level control, such as mission execution [5]. This gap can be partly filled by providing the autonomous vehicle with the cognitive ability to reason about the knowledge provided by the domain expert. This cognitive ability allows the autonomous vehicle to function more autonomously, alleviates the workload of the operator, and allows the domain expert to extend the knowledge of the autonomous vehicle in an intuitive way.

Knowledge representation and structure plays a key role in the reasoning process. Expert knowledge must combine logical and statistical formalisms [6]. Logical formalisms, such as logic programming, symbolic parsing and rule induction is able to handle complexity in the knowledge. Statistical formalisms, such as Bayesian networks, Markov networks, Markov logic networks and neural networks handle uncertainty in the knowledge. In a dynamic environment, knowledge and environmental data may change rapidly and continuously. Autonomous systems, for example, autonomous exploratory robots [7, 8], encounter continuous changes in their environmental data when applied in dynamic environments. Because of the dynamism and degree of certainty in some environments, statistical formalisms, rather than logical formalisms are more suitable for reasoning. Moreover, for quick decision-making, reasoning must be based on an optimal, salient subset of the expert knowledge, given environmental sensory information. Whenever there is a change in the environmental stimuli, the salient knowledge becomes obsolete and needs to be replaced, using the new environmental stimuli. Therefore,

statistical formalisms which rely on discovery, learning and structure generation are inflexible and computationally- and time-expensive and are therefore not suited for real-time high-level autonomous vehicle control. The cognitive reasoning process needs to make a decision in an acceptable time frame. The cognitive reasoning process therefore, relies on a well-structured knowledgebase and real-time, adaptive cognitive reasoning ability. Naturally, researchers are looking towards human neuro-cognitive sciences for guidance towards cognitive reasoning solutions for robots. There is a tendency to develop robots to mimic human decision-making and behaviour. However, there are two conflicting approaches towards the design of cognitive architectures [9]: to create a model of cognition and gain an understanding of cognitive processes and to build useful systems that have a cognitive ability and thereby provide robust adaptive behaviour that can anticipate events and the need for action. The first is concerned with advancing science, the second is concerned with effective engineering.

There has been an ongoing quest to understanding human cognition [10] and a number of computational models [11] and neuro-cognitive architectures have been developed in order to simulate, study and understand human cognition. The best-known architectures are Adaptive control of thought (ACT) [12], Adaptive Control of Thought-Rational (ACT-R) [13], State operator and result (SOAR) [14], Semantic pointer architecture unified network (SPAUN) [15] and Neural Engineering Objects (Nengo) [15]. These architectures were designed to investigate human neuro-cognitive functions of the brain, while some, for example [16], go further and extend the architectures by augmenting it with human emotion characteristics. These architectures provide a modular representation of the cognitive functions of the brain, and is therefore an attractive option for designing similar architectures for cognitive robotics. Computational devices have seen a reduction in cost and size, while increasing in computational power and this have led to an increase in the application of cognitive architectures to robots.

Architectures, such as those described above, focus on the computation (or behavioural outcome) of cognition. However, little attention is given to the foundation of cognitive computation i.e. memory. Memory is only considered as a basic “repository” of information, used by cognitive processes, while there is very little consideration of the dynamical memory-specific processes, even though there appears to be a degree of consensus that cognition should be founded on formation and manipulation of memory and memory as associative and developmental [17]. It is advised that cognition be examined from the perspective of memory (and its associative processes), rather than from a computation or behavioural outcome, point of view.

However, cognitive neuroscience is not trivial, and in [15], four significant challenges are identified, (1) the complexity of building representational structures for semantics, (2) utilizing the semantic structures effectively, (3) an executive, controlling action selection and (4)

representing and recalling memory. Of the challenges, memory representation, memory recall and action selection are, arguably, the most significant, since the autonomous vehicle's behaviour is directly impacted by these actions. Memory representation can be further described in terms of semantic memory (short term and long-term) and episodic memory (experience-based) [18, 19] which, together with memory processing, is often referred to, in neuroscience, as working memory. Working memory is a very complex field of neuroscience and still subject to intense research. Various working memory models, such as Baddeley's Multicomponent model, Cowan's embedded process model and Engle's controlled attention model are presented in [20]. Despite its complexity, working memory is equally important in a robo-cognitive architecture.

Cognitive robotics is described as "the study of knowledge representation and reasoning problems, faced by an autonomous vehicle (or agent) in a dynamic and uncertain world" [21]. Cognitive control architectures typically provide cognitive functions, such as perception, attention, planning, memory, reasoning and learning. In cognitive robotics, where some form of memory representation is used, the initial memory is often provided by a domain expert. This memory is then often stored in a knowledgebase or some other storage structure, such as a file or database.

In some autonomous vehicle control architectures, control models are learned through methods, such as artificial neural networks, to simulate memory representation, memory recall and the executive functions of the brain. The models represent memory through synaptic weight assignment, which is adjusted during a learning process. When presented with an input stimulus, the model "recalls" learned facts by applying the synaptic weight and input stimulus to an activation function.

In a survey [1], it is shown that contemporary AI machine learning techniques, (e.g. artificial neural networks, Reinforcement learning and the deep learning variants of these techniques) are favoured. The survey also identifies that the complexity of machine learning models and parameter calibration, remains a problem for autonomous vehicles in general. Autonomous vehicles must have the capability to reason about, and act fast, on changing environmental stimuli. For example, when a unmanned aerial vehicle (UAV), in formation flight, encounters exceptional aerial disturbances [22] or when an autonomous underwater vehicle encounters exceptional underwater disturbances [23]. Autonomous vehicles are provided with initial knowledge about its environment, and a set of rules on how to behave in that environment. For example, knowledge about the environment or operational rules of autonomous vehicles, patrolling an area [24] for security, may change at any time.

Unfortunately, for many real-world cognitive robotic applications, the approach of a-priori

learning of behavioural models is not always effective. Robots deployed in remote, unknown and dynamic locations, cannot risk catastrophic failure. As mentioned before, they do not have the time to learn new complex solutions from the start every time the environment changes.

In the need for real-time, high-level control in robotics, without the overhead of model re-learning is discussed. In this thesis, the mechanics of a real-time, knowledge quantification and optimization methodology, using a set-based particle swarm optimization (SPSO) algorithm, are developed. However, while the SPSO successfully optimizes the knowledge on a task by task basis, it is not suitable for knowledge optimization in a multi-task environment.

This thesis also introduces a cognitive architecture suitable for robotic applications and, in this study, is referred to as a robo-cognitive architecture for high-level, single and multi-task control of robots. A novel, coalitional games theory-based PSO (CG-PSO) algorithm, which is based on a combination of SPSO and cooperative games theory, forms the cognitive process of the robo-cognitive architecture.

1.1 Problem description

- Memory representation - Augmenting or modifying the working memory of a remotely-deployed autonomous vehicle becomes more convoluted, error-prone and computationally expensive if the structure of the working memory is complex. Limited communication bandwidth also restricts the maintenance of the working memory.
- Timeliness - High-level controllers are often represented by states and state transitions defined as a directed state-flow model with state-action policies, constructed through machine learning techniques. These techniques progressively learn the policies of the state-flow, using user-defined parameters, which are often selected subjectively or derived through experimentation. Changes in the environment are likely to lead to the re-optimization of the parameters and re-learning of the model.
- Dynamism - When machine learning is used to generate models as high-level controllers, the controller (state-flow) is learnt in its entirety. For dynamic environments, a large number of models have to be learnt to handle different scenarios. However, when the underlying information or environment changes, learnt models may become obsolete and need to be replaced. Due to the time it takes to relearn a model, re-generation of high-level controllers in real-time operation becomes infeasible.
- Knowledge Quantification – Particle swarm optimization (PSO) is selected for the optimization of the long-term memory (LTM), in order to provide the optimal, salient subset of knowledge for cognitive reasoning. In order for the PSO to evaluate memory items

selected from the long-term memory for fitness, each memory item needs to have a numeric value for fitness evaluation. Since the long-term memory is a discrete set of memory items, algebraic fitness evaluation is not possible.

- Multi-tasking - Robots deployed in the real-world often have multiple functions they can perform (e.g. the Mars rover), and are therefore multi-task, multi-state platforms. Some of the states may be dependent on other states, while others may be completely independent. This means the autonomous vehicle may transition to states in parallel and thereby execute corresponding actions in parallel. Cognitive reasoning, using SPSO for memory optimization, is capable of finding the optimal transition from a single current state in a specific state-flow (function). However, it is not capable of providing multiple, parallel and independent transitions from multiple, current states in multiple state-flows in real-time.

Given the problem description above, the research questions are:

(1) Can working memory be represented in a form which simplifies augmentation and modification by domain expert?
(2) Can working memory be statistically quantified for the evaluation of optimality, during memory recall in cognitive reasoning?
(3) Is the cognitive reasoning capable of correct action selection for both single- and multiple, independent (or parallel) tasks?

1.2 Aims and objectives

To address the research questions above, this thesis develops and evaluates cognitive architectures, suitable for real-time, high-level autonomous vehicle control in remote and dynamic environments. The objectives of this research study are:

a. To design, implement and test a simple knowledge representation for the working memory, which simplifies knowledge modification and augmentation.
b. To design and develop a statistical, entropy-based quantification algorithm for the quantification of discrete knowledge in the working memory.
c. To investigate a cognitive reasoning process for real-time memory recall by combining knowledge quantification with particle swarm optimization (PSO), for single task execution.
d. To investigate a cognitive reasoning process for real-time memory recall combining game theory and particle swarm optimization (PSO), for multiple action selection for parallel, multi-task execution.

e. To evaluate the cognitive architecture for correctness, control and time efficiency, using simulation.

1.3 Contributions

The main contributions of this thesis are:

1. The creation of a novel working memory representation, structured to simplify modification and augmentation.
2. The design and development of a novel adaptive entropy fitness quantification (AEFQ) algorithm for the statistical quantification of discrete memory items (knowledge).
3. The design and development of a cognitive reasoning process for memory recall, using an improved set-based particle swarm optimization (SPSO) algorithm (which uses the AEFQ algorithm) for action selection for single task execution.
4. The design and development of a cognitive reasoning process for memory recall, using a novel CG-PSO algorithm (which uses the AEFQ algorithm) for multiple action selection for multiple, parallel task execution.
5. Confirmation of the suitability of the robo-cognitive architectures in the execution of four use cases in simulation.

1.4 Publications

Journal papers published

1. Deon de Jager, Yahya Zweiri, Dimitrios Makris (2019). "A Particle Swarm Optimization approach using Adaptive Entropy-based Fitness Quantification of Expert Knowledge for High-level, Real-time Cognitive Robotic Control", SN Applied Sciences, No (12), Vol 1. https://doi.org/10.1007/s42452-019-1697-4
2. Deon de Jager, Yahya Zweiri, Dimitrios Makris (2020). "Real-time Episodic Memory Construction for Optimal Action Selection in Cognitive Robotics", International Journal of Mechanical and Mechatronics Engineering, No (1), Vol 14. doi.org/10.5281/zenodo.3669208

1.5 Methodology

To illustrate and explain the methodology in this study, simulated UAV use cases are used to contextualize some of the theories and concepts. It should be noted that, without the loss of

generality, the methodology could equally apply to other autonomous vehicle (or automation) scenarios.

In order to answer the research questions, two cognitive architectures which focusses on working memory for memory representation, memory recall (quantification and optimization) and action selection and execution, is developed. Figure 1.1 gives an overview of the two cognitive architectures, including the main components, developed in this study.

The long-term memory is defined by a domain expert and formulated in a simple and discrete proposition logic-based structure.

Memory recall is performed by the central executive (CE) and consists of two primary functions, statistical memory quantification and memory optimization using a particle swarm optimization approach. In this research study, two memory recall approaches are examined: 1) memory recall resulting in episodic memory construction for single-task action selection and execution and, 2) memory recall resulting in activated memory and focus of attention for multiple, parallel action selection and execution. Memory quantification is based on environmental stimuli and performed by the novel AEFQ algorithm. The AEFQ algorithm uses the maximum entropy principle (MEP) [25], for the assignment of a probability distribution over a memory item. Memory optimization for the first memory recall is performed using an improved SPSO algorithm. Memory optimization for the second memory recall is performed using the novel CG-PSO algorithm.

Four use cases are defined and executed in simulation, to evaluate the suitability and performance of the cognitive architecture for both single task and parallel, multi-task execution.

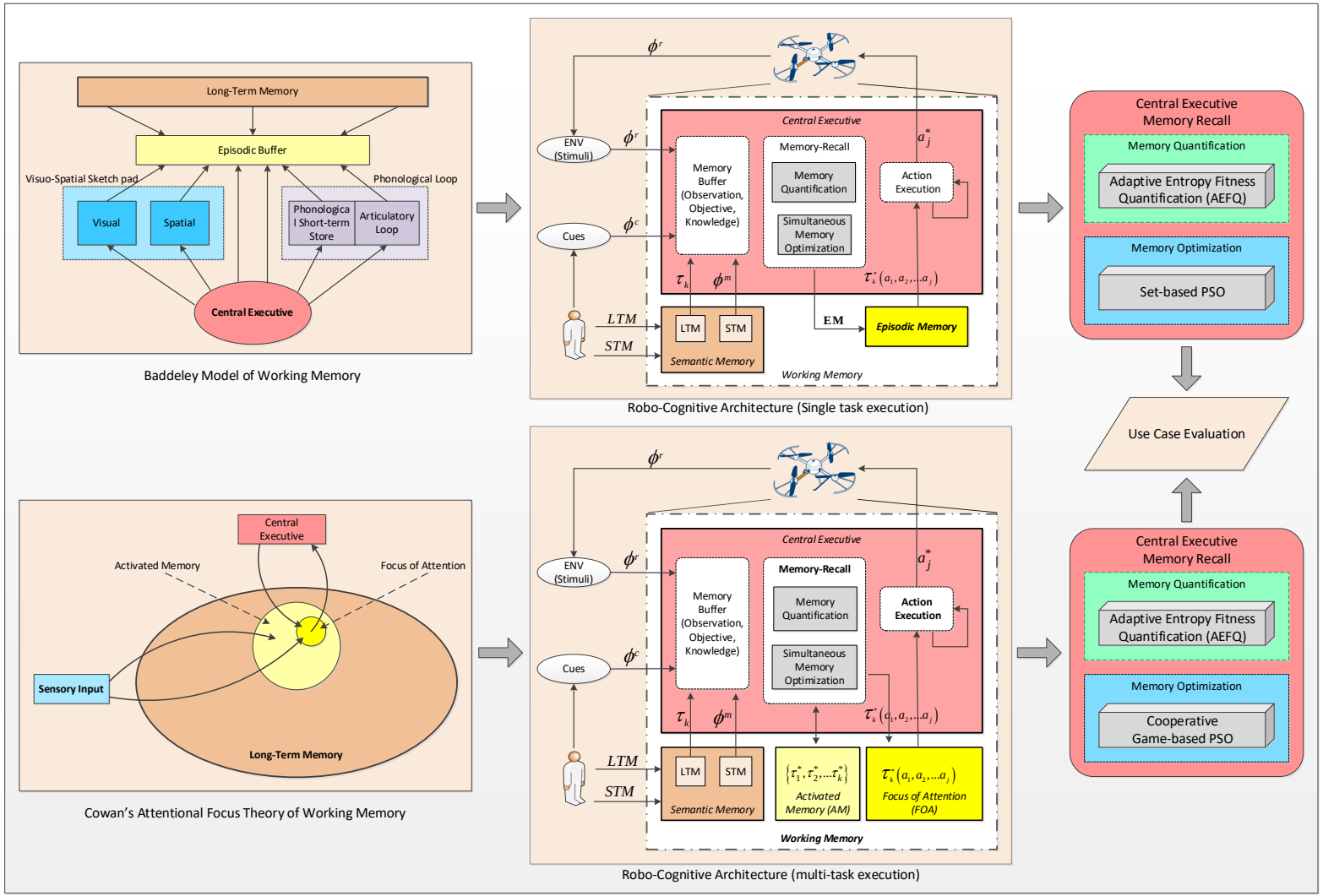

Figure 1.1 The design and evaluation of two robo-cognitive architectures.
The architecture on top is based on Baddeley's model of working memory for the construction of episodic memory, for single tasks selection and execution. The robo-cognitive architecture shown below is based on Cowan's attentional focus theory of working memory. Cognitive reasoning is performed by real-time optimization of working memory optimization, using particle swarm optimization and adaptive entropy memory fitness quantification of memory. Both architectures are evaluated in use case simulations.

To the best of my knowledge, no studies have been conducted in real-time, statistical quantification of memory, using the MEP approach. Moreover, no studies have been found for memory (or knowledge) optimization, using a particle swarm optimization approach.

1.6 Thesis outline

The rest of this thesis is structured as follows:

a. Chapter 2 – Related work

   This chapter discussed research, relevant to the cognitive architecture and the core working memory functions, memory representation, memory recall, action selection and action execution.

b. Chapter 3 – Background Theory

   This chapter discusses the theoretical foundations of the core functions of the cognitive architecture. An overview of the human cognition framework is given to contextualize working memory and the central executive. Various working memory models, relevant to the cognitive architecture proposed in this study, is discussed. PSO is one of the two core functions of memory recall and therefore, a detailed overview of the standard PSO and the set-based PSO is given. The chapter also gives a detailed overview of cooperative games theory with definitions relevant to coalition formation, based on the bargaining set solution concept.

c. Chapter 4 – Working memory representation and quantification

   This chapter describes the contributions (1) to (2) in detail. In this chapter, a memory representation structure, representing the long-term memory used by the cognitive reasoning process in memory recall, is discussed in detail. This chapter also discusses in detail, the memory quantification methodology, used in the optimization step of memory recall. The novel statistical entropy-based quantification algorithm is discussed in detail in this chapter.

d. Chapter 5 – Memory optimization

   This chapter describes the first part (investigation) of contribution (3) and (4), which develops a cognitive process for memory recall. This chapter empirically evaluates both the standard PSO and set-based PSO in detail for the suitability of using PSO for knowledge optimization in memory recall. A detailed empirical evaluation is discussed in this chapter.

e. Chapter 6 – Robo-cognitive architectures

This chapter describes the second part (methodology) of contributions (3) and (4). In this chapter, two methodologies for memory recall in cognitive reasoning is developed for the real-time, high-level cognitive control of an autonomous vehicle. Memory recall is developed for two cognitive architectures. In the first method, an architecture for real-time, cognitive control using SPSO for single-task execution, is developed and in the second method, an architecture for real-time, cognitive control using CG-PSO for multi-task execution, is developed.

f. Chapter 7 – Evaluation by simulation

This chapter describes contribution 5 in detail. Four simulations are evaluated to confirm the suitability of the robo-cognitive architectures for high-level control of autonomous vehicles. The first two simulations (section 7.1) apply an episodic working memory approach for single task execution, while the next two simulations (section 7.2) apply the attentional focus theory of working memory approach for multi-task execution.

g. Chapter 8 – Conclusion and Future work.

This chapter concludes the thesis, highlighting its contributions and proposes areas for future research.

Chapter 2

Associated research

In this chapter, research relating to neuro-cognitive architectures and cognitive robotic architectures, are reviewed. In particular, the role of working memory processes, as opposed to cognitive processes, are reviewed.

In a cognitive architecture, memory represents knowledge obtained either through learning or provided by a domain expert. Memory representation is key to the success and efficiency of the core functions of a cognitive architecture. The effectiveness of the central executive in memory recall, action selection and action execution are directly affected by the structure and content of the long-term memory in working memory. However, associated research in autonomous vehicle control often refers to “knowledge”, “information” or just data, as the input to an autonomous vehicle control process. Controllers govern the behaviour of the autonomous vehicle and are often described in procedural (as opposed to cognitive) terms. In this chapter, *working memory representation* and *cognitive reasoning* are reviewed in terms of *knowledge representation* and *autonomous vehicle control*, respectively.

In section 2.1, popular approaches for the discovery and representation of knowledge are reviewed. section 2.2 review autonomous vehicle control approaches which uses procedural control methods and section 2.3 review autonomous vehicle control approaches which uses cognitive control methods.

A critical review of the various approaches and main differences, between these approaches and the methodology proposed in this study, is given at the end of the chapter.

2.1 Knowledge representation

Knowledge is the basis for successful decision-making, and may be learned (or derived) computationally or explicitly defined by a domain expert. Equally important, is how the knowledge is obtained, formulated and structured. A complicated format will be computationally expensive and error-prone and decision-making may be sub-standard. For example, formats, such as modal logic or Hennessy-Milner logic [26, 27], have complex and unintuitive structures. The importance of simple knowledge representation structures becomes evident if formal reasoning theories are investigated [28]. In this work, three types of reasoning are described: deduction, induction, and abduction. These types of reasoning may be useful in

humanoid robot or human-robot applications, such as [29-33], they are not particularly suitable for remotely-deployed, exploratory autonomous vehicles, because, a subjective overemphasis is often placed on some factors, depending on the preferences of the subject doing the reasoning. This could lead to unacceptable levels of accuracy in inference, when applied to high-level control of autonomous vehicles.

Knowledge is often defined as probabilistic reasoning models (PRMs) and modelled as probabilistic graphical models (PGM) [6, 34]. probabilistic graphical models handle uncertainty well and provide a useful structure for statistical inference. Arguably, the most widely used probabilistic graphical models for statistical inference are Bayesian networks, Markov networks and Markov logic networks.

Bayesian networks are directed acyclic graphs which expresses causal relationships between random variables [6, 34, 35]. The Bayesian network is constructed by creating a node for each random variable in the long-term memory. For each causal relationship between two random variables, a directed edge is created between the two nodes representing the two random variables. Associated with each node is a user-defined conditional probability distribution (CPD) which indicates the probability of its states, given the probability of the states of its parents. The conditional probability distribution is represented as a conditional probability table (CPT). The conditional probability table is a table that has one probability for every possible combination of parent and child states. This is an N+1 dimensional table, where N is the number of parents. The Bayesian network is used to answer queries, for example, probability-of-evidence queries [35]: what is the probability $Pr(e)$ of some variable instantiation, e, given some evidence X and Y? In probability–of-evidence queries, $E = \{X, Y\}$ is the set of evidence variables.

A Markov network is an undirected graph which models the joint distribution of a set of random variables. A node is created for each random variable and an edge between two nodes expresses the dependency between the two random variables [6, 36-38]. Some nodes in the graph form cliques, which are n-vertex subgraphs of the graph where n indicates the number of vertices of the clique [39]. For cliques with more than one vertex, i.e. $n > 1$, each pair of vertices is connected by an edge. Associated with each clique is a user-defined clique potential function which maps instantiations of the random variables in the clique to non-negative real numbers. This mapping process is referred to as clique factorization [37, 38].

A Markov logic network combines Markov networks and first-order logic (FOL) to "soften" the logical constraints of the long-term memory [36, 40]. Formally, a Markov logic network L is defined as a set of pairs (F_i, w_i), where F_i is a formula in first-order logic and a weight, w_i. The weight w_i is a real number, indicating the strength of the logical constraint. The higher the

weight, the greater the difference in log probability between a world which satisfies a formula and one which does not [6, 40, 41]. The weight w_i is determined by a user-defined function. Together with a finite set of constants C (defined by the evidence), a Markov logic network defines a ground Markov network $M_{L,C}$. A binary node is created for each possible grounding of each predicate, appearing in some formula in L. The value of the binary node is 1 if the represented predicate is true, otherwise it is 0. An edge is created between two nodes if, and only if, the two corresponding ground predicates appear together in at least one grounding of one formula in L. The ground Markov network, $M_{L,C}$, also contains one binary feature for each possible grounding of each formula F_i in L. The value of the binary feature is 1 if the represented formula is true, otherwise it is 0.

The main difference between the Bayesian network, Markov network and the Markov logic network is what the nodes represent. In a Bayesian network and a Markov network, the nodes represent the random variables in the long-term memory. In a Markov logic network, the nodes represent the predicates in the long-term memory.

The descriptions above highlight three common activities when using probabilistic graphical models as a statistical formalism:

a. Construct the network structure (Bayesian network, Markov network or Markov logic network) using the formally defined long-term memory and evidence.
b. Define the network parameters (conditional probability tables for Bayesian networks, potentials for Markov networks and weights for Markov logic networks) by applying user-defined functions to the knowledge.
c. Use the network and network parameters to execute statistical inference queries.

In a dynamic environment, an autonomous system can expect both change and uncertainty in the evidence it observes. Therefore, probabilistic graphical models need to be created dynamically, to model the world in real-time, using the latest evidence and expert knowledge. Considering the activities mentioned above (a to c), if the environment is dynamic, the computational cost of using probabilistic graphical models becomes prohibitive [42, 43].

For many years various machine learning approaches, such as statistical relational learning (SRL) [44], inductive logic programming (ILP) [45-47] and knowledge-based model construction (SMMC) [48, 49] have been used to derive expert knowledge from existing data sources. Some machine learning systems have been developed to learn and formulate knowledge, for example, FOIL [50] learns Horn clauses from relational data and MADDEN [49] performs statistical knowledge extraction from textual data. CLAUDIEN [46] is an inductive logic programming engine which computes a set of logically valid clauses from

datasets. In [51] a data mining technique is used for knowledge discovery in a multi-objective optimization topology. Clustering and association rules are applied sequentially to evaluate the Pareto-optimally of potential solutions. Once clustering of the data is complete, the solutions are visualized in the objective space. Discovering knowledge computationally is useful where the knowledge is encapsulated in vast amounts of data. However, for high-level control of autonomous vehicles, bespoke, problem-specific knowledge is required. This knowledge is usually defined by a domain expert, according to the design purpose of the autonomous vehicle. Whether knowledge is discovered computationally of provided by a domain expert, the representation needs to be in a form suitable for computational inference. Popular forms include first-order logic (or predicate logic) formulae and Horn clauses. Horn clauses are particularly useful, as its syntax is similar to programmatic conditional statements, and therefore easier to implement. (Because of its popularity, the syntax for first-order logic formulae and Horn clauses are described in more detail in section 3.2). Once the knowledge is effectively represented, the representation or variants thereof, can be used in high-level and low-level autonomous vehicle controllers.

2.2 Procedural autonomous vehicle control

Linear temporal logic (LTL), is used in [52], as a formal language to define the tasks of an autonomous vehicle, where linear temporal logic is combined with Petri Nets to determine optimal movement planning for multiple robots. The problem of high-dimensionality in the relationship between task planning, using linear temporal logic and robot motion is investigated by Shoukry et al. [53]. Here, linear temporal logic is used to define a set of propositions, applicable to all robots, for each region of the workspace. The robots' movements across regions are controlled by the linear temporal logic propositions.

In addition to high-level autonomous vehicle control, memory representation is also applied to low-level control of robots. For example, improving path planning in dynamic environments, where obstacles are avoided by prioritizing and predicting the future behaviour of the object [54]. In [30], a semi-autonomous high-level controller is proposed for the semi-autonomous control of autonomous vehicle teams in urban search and rescue missions. The objective of the controller is to reduce the workload of the autonomous vehicle operator. Other cognitive robotic approaches, combines inductive logic programming, used for predicate generation, and reinforcement learning, to learn optimal behavioural policies in [55].

A combination of linear temporal logic and Markov decision processes is used to synthesize high-level controllers in [56]. Here the linear temporal logic formulae provide a formal

definition of tasks for the autonomous vehicle and the Markov decision processes govern the execution of those tasks. However, synthesizing high-level controllers in a dynamic environment remains a challenge. A framework to increase the adaptability of the synthesis process, by using a 3-layer top-down hierarchical decomposition of the control problem, is introduced. A three step-approach is used to firstly, solve the linear temporal logic problem on a finite state automaton (FSA), secondly, find the best policy for transitioning and thirdly, synthesize a controller.

Reinforcement learning (or Q-Learning) of Markov decision process type controllers are increasingly being combined with other methodologies to learn high-level controllers to accomplish some task. Generally, the objective of Q-Learning is to iteratively select the best policy, i.e. state-action, which maximizes the expected discounted reward (Q-value), given the current state, the user-defined short-term memory and user-defined rewards. The most popular approach is the use of the Bellman equations [57], which calculates the optimal Q-value over all policies. In [58], Q-learning is used in combination with a Deep Deterministic Policy Gradients (DDPG) algorithm for a UAV to learn a landing task in simulation. In [59], the effectiveness of the Q-learning algorithm for autonomous vehicle path planning, is improved by using a flower pollinating algorithm to initialize the q-values of the algorithm. These approaches mostly rely on traditional, component-based software architecture approaches, to govern high-level control of an autonomous vehicle. Research into cognitive robotics (or autonomous vehicle control), has seen an increase in research into high-level autonomous vehicle control, especially in dynamic and uncertain environments.

2.3 Cognitive autonomous vehicle control

Emulating the power and adaptiveness of human cognitive reasoning in autonomous vehicle control is very attractive. As the understanding of human cognition developed, the interest in computation models for cognitive processes increased. This interest led to the development of a number of cognitive architectures, with the purpose of simulating human cognitive processes. The most well-known are Adaptive control of thought, Adaptive Control of Thought-Rational, State operator and result, Semantic pointer architecture unified network and Neural Engineering Objects architectures mentioned in the introduction. Other cognitive architectures include Sigma ($\sum$) [60], Learning Intelligent Distribution Agent (LIDA) [61], and Connectionist Learning with Adaptive Rule Induction On-line (CLARION) [62], amongst many others. A comprehensive survey of 58 cognitive architectures, spanning 20 years, was conducted in [63], where it is shown that there is consensus on the various cognitive processes. This is mainly due to advancement in research of human cognition; however, the computational architecture of

these processes varies greatly. Although there are many similarities amongst the various architectures, especially regarding the cognitive processes [64], there are also many differences, especially on how memory is represented and processed.

Some cognitive architectures are developed with a specific focus on humanoid robotics, and are closely tied to the physical architecture of the autonomous vehicle, the iCub open-systems platform [65] and Cognitive Architecture (COG) [66] are two such architectures. These architectures were designed to investigate human neuro-cognitive functions of the brain, while some go further and extend the architectures by augmenting it with human emotion characteristics [16]. These architectures, although complex, have opened the door to various cognitive architectures, specifically for autonomous vehicle control.

As described in the introduction, the two key factors of cognitive robotics for autonomous reasoning and decision-making, are working memory representation and cognitive reasoning [67]. However, autonomy in robots deployed in dynamic environments is non-trivial, as the environment may vary greatly, this especially applies to the operational environments of humanoid robotics [33, 68], human-robot interaction [29, 30, 69, 70], Search and Rescue (SAR) [71] and multi-robot systems [72, 73]. Computational architectures, based on neuro-cognitive architectures are complex, inflexible and computationally expensive. These architectures attempt to mimic human cognitive functionality, which involve numerous complex cognitive processes. Moreover, neuro-cognitive processes have to process information dynamically, often under a degree of uncertainty. Computational architectures based on neuro-cognitive architectures are therefore not easily applied to autonomous vehicle solutions, especially when they are deployed in remote locations. It is argued in [74] that high-level autonomous vehicle control, where perception, reasoning and decision-making is required, is best achieved with a cognitive architecture. Five requirements are listed for intelligent high-level cognitive control: 1) represent, integrate and use knowledge; 2) recognizing or learn new patterns of knowledge; 3) reason and solve problems; 4) flexible, adaptive, dynamic, and real-time behaviour; 5) interact with humans in a natural way. For intelligent control, the cognitive architecture is viewed in two parts: the architecture and the content. The architecture stays constant, while the content is dynamic. The architecture is the algorithmic processes common to all robots, while the content is defined as the semantic memory (knowledge), procedural memory (skills) and episodic memory (experience) each autonomous vehicle possesses.

A cognitive architecture to autonomously control a transportation autonomous vehicle for use in a factory or warehouse, is developed in [75]. The architecture is based on and extends the State Operator And Result architecture. Information is processed from the Current Perception

Memory, the Visuo-Spatial Memory and the Goal Memory, prior to passing it to State Operator And Result for processing.

CORTEX, is an autonomous vehicle architecture discussed in terms of use cases and autonomous vehicle applications [76]. CORTEX is composed of a number of computational models which are selected according to the autonomous vehicle control problem. The main feature of the CORTEX architecture is the existence of a unified, dynamic working memory, which can represent environmental data and high-level symbols. The executive module manages action plan generation and execution between all modules.

A cognitive architecture which integrates a number of cognitive modules for concept learning, knowledge acquisition, language learning, and decision making is introduced in [77]. Concepts, language, and actions are learned through trial and error from the state in which the autonomous vehicle has no knowledge of the environment. The integration of the various components is performed through the use of variables and uses learned models to select the appropriate actions.

The performance of working memory, in syntactic sentence realization is investigated in [78]. The role of working memory in grammatical encoding, is experimentally examined using a combination of grammatical theory and a computational psychological account of human cognition. The adaptive control of thought-R architecture is used to represent the human cognitive functionality.

An architecture for ethical robots inspired by the simulation theory of cognition is developed in [79]. The study introduces an architecture which enables robots to autonomously act in a safe and ethical manner. An additional ethics layer is added to the architecture and the robot controller generates a set of prospective behavioural alternatives. Given an initial task, the ethics layer simulates and evaluates the consequences of each alternative, and the results are sent to the robot controller. The architecture was tested using a controlled and static environment, with behavioural alternatives kept unchanged. It is pointed out that the application of behavioural alternatives fits well into the composite state transition presented in this research study in section 4.1.2, figure 4.4.

A review of dynamical approaches to cognitive systems is given in [80]. The focus of the review is on concepts, data analysis methods and computational model. Human-autonomous team stability and adaptiveness are investigated in [81] and [82]. In these studies, a human operator teams up with an autonomous agent, developed using the adaptive control of thought-rational cognitive architecture, in a simulated victim locator task scenario. In [83], the authors point out that that all messages to the synthetic teammate should not be ambiguous or cryptic, otherwise

the synthetic teammate would not understand the text message. This highlights the need for simple, well-structured knowledge representation, when a cognitive architecture is used for autonomous vehicle control.

A standard model for the representation of the human mind, is proposed in [84]. The standard model combines the key aspects of three standard architectures, Adaptive Control of Thought-Rational, State Operator And Result and SIGMA. Computational entities to represent these aspects are proposed to form the basis for cognitive architectures for robotics. A cognitive architecture for inner speech [85], uses the Standard Model of Mind, to simulate inner speech, or inner dialogue. The central executive controls the computation which retrieves information from long-term memory, and constructs conscience thoughts. The memory structure of the solution, is based on the memory structure defined by the Standard Model of the Mind, which consist of three types, the short-term memory (STM), the procedural and the declarative long-term memory (DLTM), and the working memory system (WMS).

A mathematical model which represents the relationship between bottom-up and top-down attention controllers, is presented in [86]. The study focusses on the neuronal functions of focussing attention using bottom-up (BU) and top-down (TD) processes in response to input stimuli, focussing of attention. Although the study focusses on human cognitive brain processes, an important fact is highlighted: for cognitive reasoning, irrelevant memory (knowledge) need to be supressed, while the saliency of relevant knowledge (memory) need to be increased. This point is especially important for effective high-level control in autonomous vehicles and is addressed with knowledge optimization during memory recall, in this research study.

Dual processing of reasoning is defined in two forms: 1) fast-working and implicit, and 2) affect-related and slow-working. In [87], a nonlinear dynamical systems theory approach is used to investigate the dynamic interactions and transitions among the two forms of processing. Dual-process theory treat cognition as informational in nature and as such, human cognition is regarded as modular. In this context, modules are defined as being informationally encapsulated, domain specific, and automatic, rather than connectionist, i.e. connected networks of nodes. The human brain is a nonlinear dynamical system, where interaction is not only on neuronal level, but is also strongly influenced by the state of the environment [88]. In a dynamical approach, the macroscopic state of a system is represented as a set of differential equations with order parameters and control parameters. The order parameters are the dependent variables of the dynamic system and control parameters are the independent variables of the dynamic system. Order parameters and control parameters guide the system's dynamics.

A mobile robot navigation system presents an integrated system using a motivated developmental network (MDN) and radial basis function neural network (RBFNN) to mimic the supervised learning of the cerebellum and reward-based learning of the basal ganglia, respectively [89]. The two systems are integrated to provide a hybrid complex cognition model, to navigate a mobile robot in unknown environment. The experimental results of the study show that by combining the cerebellum model and basal ganglia model, navigation accuracy is improved and learning steps are slightly reduced. Unfortunately, the environment remained static during the experiments, therefore, the effectiveness of the proposed method cannot be evaluated for dynamic experiments.

In [90], a visual strangeness-driven long-term memory with autonomous ant colony learning algorithm, is proposed for the improvement of the visual cognitive function of intelligent robots. The approach combines an incremental self-organizing network as long-term memory structure and visual strangeness internal motivation Q learning method, in working memory. The proposed cognitive computing model is based on hippocampal-prefrontal memory system, learning, pattern recognition and classification, storage and memory. A self-organizing map (SOM) is used to store learned Q-values in a neural network, which, along with long-term memory, represents knowledge. The knowledge is structured as two parts: perceptual knowledge network and perceptual knowledge Q-value network. The perceptual knowledge is stored and accumulated separately, and the corresponding Q value is taken in each step. While the approach is useful for acquiring new knowledge over time, the knowledge of the robot is limited to that of the samples, produced by the domain expert. Therefore, the effectiveness of the approach when presented with unknown information, cannot be evaluated. Moreover, time-sensitive control of the robot will be constrained by the determinism of the contents of its long-term memory, learned from the samples.

2.4 Critical review

There is an infinite number of ways knowledge may be represented. Whatever representation is chosen, the cognitive reasoning process (or inference engine) has to interpret the information contained within the knowledge.

autonomous vehicles and semi-autonomous robots, remotely deployed in unknown and dynamic environments, are often required to make time-sensitive decisions, based on continuously changing information. The methods discussed in section 2.1 may prove to be sufficient for discovery and formulation of knowledge for high-level autonomous vehicle control in a controlled or well-defined environment. However, in an unknown or highly

dynamic environment, environmental stimuli may change constantly. Moreover, the expert-provided knowledge of an autonomous vehicle may have to be modified or augmented in real-time, often over vast distances. In these types of environments, knowledge representation using complex syntax (such as modal logic and linear temporal logic) is computationally expensive and error-prone and remote updates will overload communication bandwidths. The syntax of these representations is not intuitive, difficult to comprehend and often open to interpretation and therefore error-prone. Using these knowledge representations are more suitable in a controlled and reasonably static environment and less suitable for real-time, high-level control of autonomous vehicles (with the possible exception of humanoid robots).

first-order logic is based on propositional logic, combined with quantifiers, to form a first-order logic formula, which is more suitable for knowledge representation for robots. Although the simpler structure will have a reduced cost with regards to communication band-with, it is still error-prone, since the syntax is not intuitive, especially in a dynamic operational environment. In addition, the logical structure may still be complex and will still require some computational resource for inference. Simplifying the first-order logic formula by removing the quantifiers and decomposing the formula into a propositional logic formula, is a more suitable representation of knowledge for robots. This approach is discussed in more detail in section 4.1.

Learning high-level controllers using machine learning techniques, such as Q-learning, may be suitable in well-defined or controlled environments. Here a degree of dynamism may be catered for by learning a large number of models, in order to cater for as many scenarios as possible. However, if the environment is unknown and/or dynamic, it is not always possible to define a-priori, which features to learn.

Many of the approaches reviewed above, use machine learning which rely on the specification of control parameters. The accuracy and efficiency of approaches, such as machine learning or dynamic systems, is directly affected by the parameters supplied. Initially these parameters are specified subjectively and are then optimized during learning cycles or repetitive experiments. While machine learning approaches deal with some degree of noise in the data, any significant noise requires that the parameters be re-optimized. This means most models need to be re-learned as well. For real-time, high-level control of autonomous vehicles, it is expected that the environment may change significantly, leading to a significant change in the information the cognitive reasoning process has to deal with. Moreover, since it is expected that reasoning be done in real-time, there is no time for re-learning of models.

The approaches described above all attempt to provide architectures and solutions which provide high-level control for robots. Some of these approaches are based on procedural

software design. Others are starting to explore machine learning, in combination with procedural processes to introduce some intelligence into the control architectures. Others attempt to leverage cognitive functionality, based on neuro-cognitive architectures, for intelligent control. There are two conflicting approaches towards the design of cognitive architectures [9]: "…to create a model of cognition and gain an understanding of cognitive processes and secondly, to build useful systems that have a cognitive ability and thereby provide robust adaptive behaviour that can anticipate events and the need for action. The first is concerned with advancing science, the second is concerned with effective engineering". It is argued in this study that the advancement of cognitive science should lead the design of a practical architecture for cognitive robotics and that the architecture should contain processes which will result in "robust adaptive behaviour" by the robot.

It is further argued that many of the approaches described in this section are not suitable for real-time, high-level control of autonomous vehicles because the methodologies chosen (for example machine learning) cannot provide the robust adaptive behaviour, which is characteristic in human cognition. When the contemporary methodology is not appropriate, a different perspective is required. This research study takes that different perspective by focussing on cognitive processes for memory representation and memory recall to enable "robust adaptive behaviour" of an autonomous vehicle.

Figure 2.1 gives a summary of the associated research and shows the relationship between the associated research and the research in this thesis. The figure puts both the associated research and the research in this thesis in the context of memory representation and memory recall.

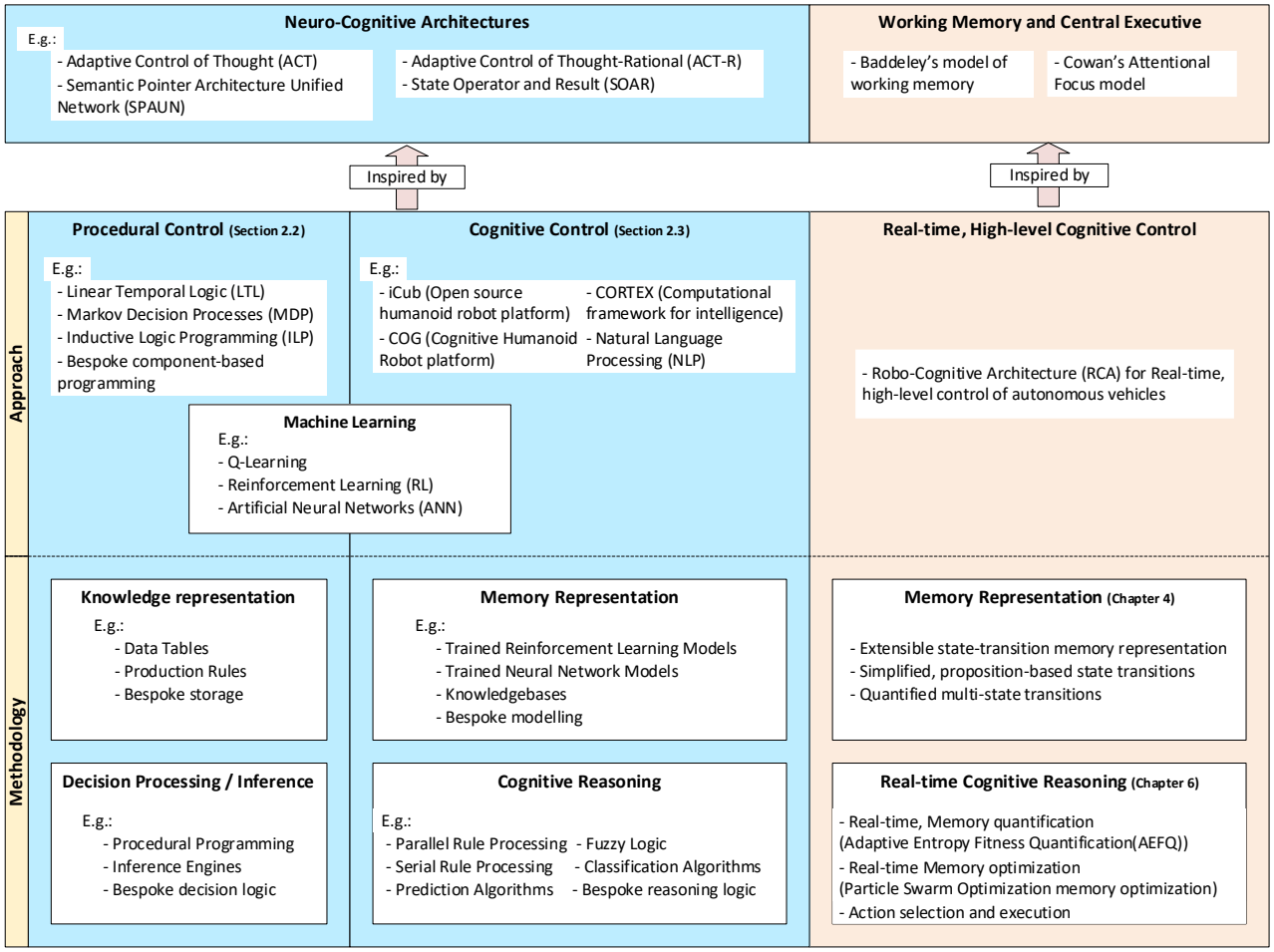

Figure 2.1 Summary of associated research and research of this thesis.
An overview of the different approaches and methodologies are given in terms of memory representation and memory recall in cognitive reasoning. Associated research is indicated in blue and research in this thesis is indicated in orange.

Figure 2.1 shows that there is no general standard for cognitive architectures in robotics and that most architecture designs are problem-specific. Some architectures follow a component-based design model, while others follow a hybrid design model, where component-based design is combined with machine learning models. It is clear however, that both neuro-cognitive architectures and cognitive architectures for robotics, prioritises cognitive processes, with very little emphasis on memory structure and process. The role of memory in the cognitive architecture is often reduced to simple storage structures. The research in this thesis prioritises working memory and working memory process, with a focus on memory representation and memory recall for action selection and execution by the central executive.

Chapter 3

Background theory

The main functions of the robo-cognitive architecture developed in this research study, focusses on working memory for real-time cognitive decision-making. The main functions are memory representation, memory recall and action selection and execution. This chapter introduces the theoretical foundations of the robo-cognitive architecture, including the two most popular working memory models suited for robo-cognitive architectures. PSO is used for memory optimization in the memory recall process and the methodology of real-time, high-level cognitive control of single task robotics, uses a set-based PSO for memory optimization. Therefore, an overview of the standard PSO (which forms the basis of all PSO algorithms), and set-based PSO, are given. The methodology for the real-time, high-level control for multi-task robotics, is based on a cooperative, games-theoretic PSO. An overview of general (but applicable) concepts of games theory, as well as the relevant definitions of coalitional games theory are presented. Section 3.1.1 introduces a common neuro-cognitive architecture, including working memory. Section 3.1.2 describes the Baddeley working memory model, and the Cowan attentional focus theory model of working memory. The working memory models are used in sections 6.1 and 6.2 for memory recall for single task and multi-task execution, respectively. Section 3.2 gives an overview of knowledge representation and section 3.3 gives an overview of information entropy, as a measurement of uncertainty in information. Knowledge representation and information uncertainty forms the basis for memory representation and quantification, developed in sections 4.1 and 4.2, respectively. Section 3.4.1 gives an overview of standard particle swarm optimization. Section 3.4.2 gives an overview of set-based particle swarm optimization, which is used for memory recall in single tasks execution in section 6.1 and section 3.5 introduces cooperative game theory, which is used in multi-task execution in section 6.2.

3.1 Working Memory in Neuro-cognitive architecture

The study of working memory and its constituent workings, have been a challenging field of study for neuro-scientists for many years. While there are still many outstanding questions, there seems to be consensus that working memory is key to cognitive decision-making and action selection. Most neuro-cognitive architectures include working memory in some form or another.

3.1.1 Functional framework for human cognition

Figure 3.1 illustrates a typical functional framework for human cognition [91].

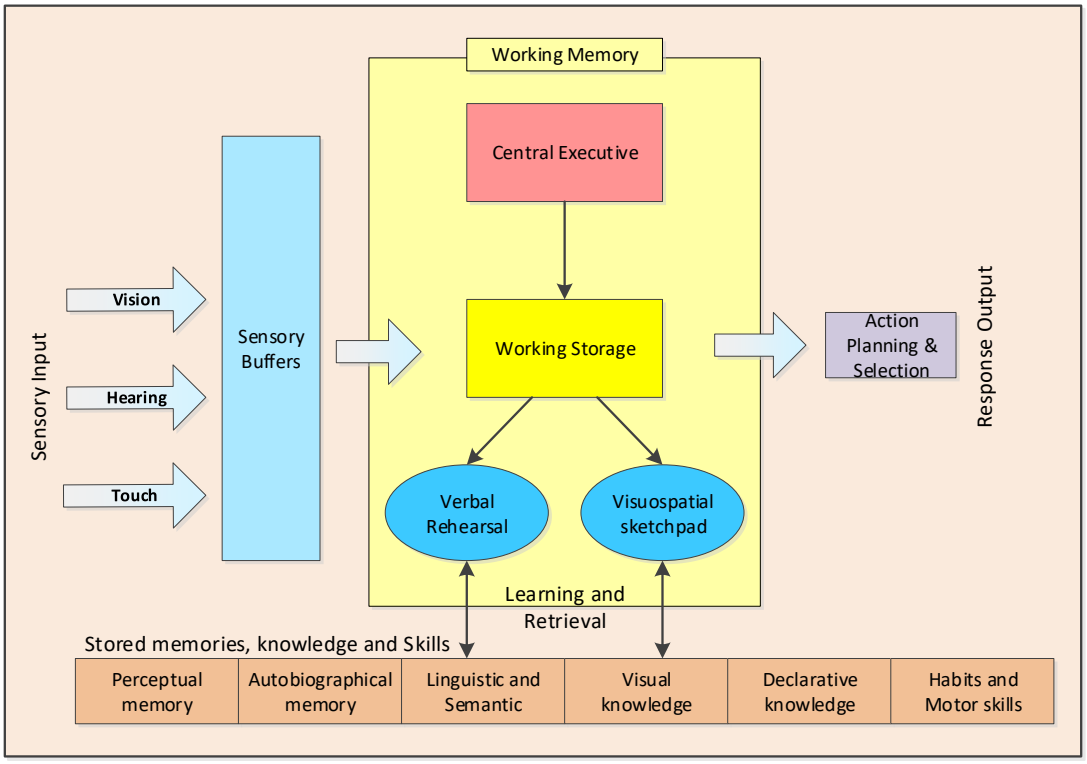

Figure 3.1 Functional framework for human cognition (based on [91]).
Working memory, including the central executive are the core memory processing components, leading to action selection and execution.

The central executive is responsible for the cognitive processes of memory classification, memory representation, recall, action selection and action execution. Collectively, these processes constitute the working memory.

3.1.2 Working Memory Models

A number of working memory models have been defined over the years. In the functional framework for human cognition, introduced by Baars and Nicole [91], working memory consists of the central executive and working storage. The working storage is created from sensory memory (verbal and visuospatial) and long-term (stored) memory, and is used in the action selection and execution process. Memory is defined further as declarative memory (semantic and episodic facts) and procedural memory (actions) in [9, 19]. Semantic facts are knowledge representing the beliefs, relations and intentions of the world, of humans and of objects, provided by a trainer (or domain expert). Episodic memory describes information about events and instances which occurred, e.g., what, where and when an event happened and is

based on personal experience. In this research study, long-term memory is semantic memory and episodic memory is derived from the long-term memory, based on real-time environmental stimuli.

There are many definitions for the different types of memory identified in neuro-science. In this study, the following descriptions will be used:

- *Cognitive cycle* – a period of reasoning, action selection and (possible) action execution.
- *Long-term memory* (LTM) – semi-permanent information, provided by a domain expert.
- *Short-term memory* (STM) – dynamic information either provided by a domain expert or generated during the action execution of a cognitive cycle. This information becomes obsolete after completion of the cognitive cycle.
- *Environmental stimulus* (ENV) – a very short-term memory consisting of information received from sensory input and used only once during a cognitive cycle.
- *Episodic memory (EM)* – limited information, based directly on environmental stimuli, obtained during a cognitive cycle. EM is knowledge with a degree of uncertainty (based on knowledge quantification) and represents "personal" experience while executing a selected action. EM becomes obsolete as soon as new information is received/observed.

A popular and well-referenced model is Baddeley's model of working memory [20] , shown in figure 3.2 and is used in [92] to improve cognitive control in agents. In Baddeley's model, the central executive processes visuospatial, phonological and long-term semantic memory, are used to create the episodic buffer. The episodic buffer performs the same role as the working storage memory in figure 3.1.

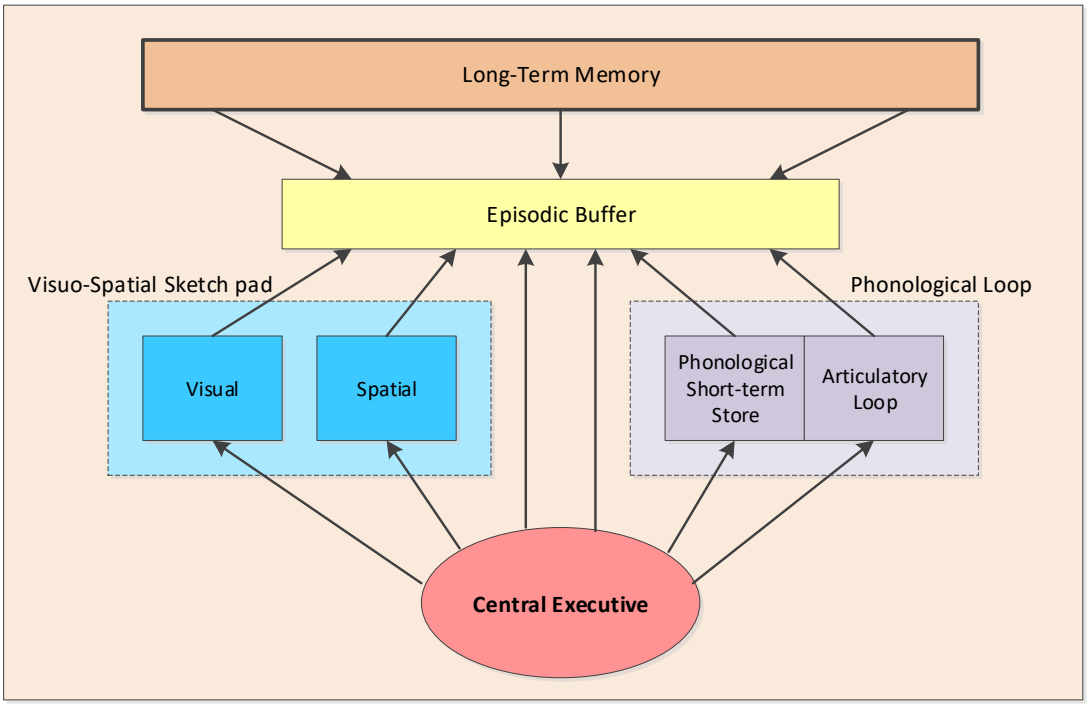

Figure 3.2 Baddeley's Model of Working Memory (based on [20]).
Central to Baddeley's model, is the construction of episodic memory from long-term memory and sensory information, by the central executive for reasoning and decision-making.

The robo-cognitive architecture for real-time, high-level cognitive control in single-task robotics (developed in section 6.1), uses the episodic buffer approach shown in figure 3.2.

A different approach is presented in Cowan's attentional focus theory [20] model. In Cowan's model, shown in figure 3.3, instead of types of memory being classified separately and distributed according to the cognitive functionality, all memory is stored as long-term memory. When memory receives attention, it becomes salient and closely stored memory is activated. Activated memory (AM) is a portion of memory which is relevant to the current environmental context and may become the focus of attention (FOA). The central executive (CE) uses the FOA for action selection and execution.

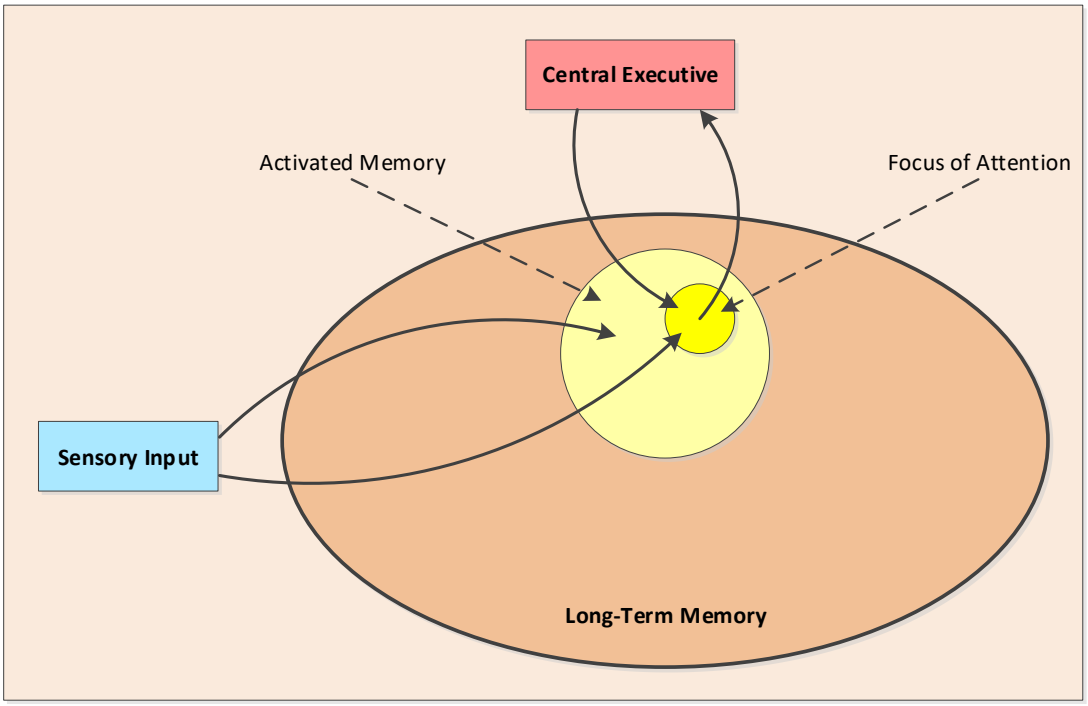

Figure 3.3 Cowan's attentional focus model (based on [20]).
Cowan's model forms "clusters" of activated long-term memory elements, from which suitable elements will become the focus of attention for reasoning and decision-making.

The robo-cognitive architecture for real-time, high-level cognitive control in multi-task robotics (developed in section 6.2), uses the AM and FOA approach shown in figure 3.3. The AM and FOA construction represent memory recall (memory quantification and optimization), performed by the central executive.

For the memory recall cognitive process, the PSO algorithm is selected as memory optimization method, due to the simplicity and scalability of its architecture and control parameters. The next section provides a brief overview of the standard and set-based PSO algorithms. The set-based PSO algorithm forms the basis of the memory recall central executive for cognitive high-level control in single-task robotics.

3.2 Knowledge representation

The structure of memory representation is important for efficient and cost-effective computation. Knowledge acquired, either through learning or provided by a domain expert, are represented by sentences, constructed from logic formula and stored in a knowledgebase. An inference process deduces new facts from the sentences in the knowledgebase.

3.2.1 First-order logic

Another widely used approach is representing knowledge as a set of first-order logic formulae which are based on propositional logic, and stored in a knowledgebase. A formal overview of first-order logic is given below.

To assist in the explanation of first-order logic, the following fictional scenario is used: *"The main objective of a Mars rover is to autonomously explore Mars and finding environmental data of water. On-board is a sample analysis at Mars (SAM) lab and a rock drill. The rover has the following knowledge, provided by a domain expert:*

- If hematite is found in clay, water may be present
- If water is present, the lab system selects a tool to collect samples"

First-Order Language $\mathcal{L}$, is defined as a set of random variables, $V = \{v_1, \ldots, v_n\}$, a finite set of connectives ¬, ∧, ∨, ⇒, ⇔, quantifiers ∀, ∃ and a signature $\Sigma = < P, F, C >$. The finite set of predicate symbols in Σ are represented by P, where each predicate has an arity m and $m \in \mathbb{Z}^+$. A finite set of function symbols, each with arity n and $n \in \mathbb{Z}^+$, is represented by F and a finite set of constant symbols are represented by C.

Given a constant or functor $f \in F$ with arity n and $(t_1, \ldots, t_n) \in \mathcal{L}$, then $f(t_1, \ldots, t_n)$ is a term in $\mathcal{L}$. Atomic formulas in $\mathcal{L}$ are predicates followed by the number of appropriate terms. Given a predicate $\wp \in P$ with arity m and $(t_1, \ldots, t_m) \in \mathcal{L}$, then $\wp(t_1, \ldots, t_n)$ is an atomic formula in $\mathcal{L}$. Atomic formulae $\mathcal{L}$ are also formed by applying the logical the connectives ¬, ∧, ∨, ⇒, ⇔, to two or more atomic formulae.

An objective of this study is to simplify the maintenance of the long-term memory. Therefore, the knowledgebase will only contain predicate atomic formulae $\wp(t_1, \ldots, t_n)$, where each term t_i represents a random variable. No functors will be used.

Another popular format for knowledge representation, are Horn clauses. Horn clauses can be defined in two ways: the disjunction form and the implication form. A Horn clause in disjunctive form, has at most one positive literal, and all remaining literals in negated disjunctive form. In implication form, the Horn clause has one positive consequence literal and all remaining literals in positive conjunctive form. For example, for the sample scenario, the following rules may be defined:

English	Hematite found in clay indicates water is present
first-order logic	$\forall\, m, s \;\; DetectedIn(m, s) \Rightarrow IsPresent(w)$
Horn clause	$\neg DetectedIn(m, s) \;\vee\; IsPresent(w)$
English	If water is present in a hematite sample, the lab system selects a tool to collect samples
first-order logic	$\forall\, m, s, o \;\; DetectedIn(m, s) \wedge IsPresent(o) \Rightarrow Selects(l, t)$
Horn clause (disjunctive form)	$\neg DetectedIn(m, s) \;\vee \neg IsPresent(w) \;\vee\; Selects(l, t)$
Horn clause (implication form)	$Selects(l, t) \leftarrow DetectedIn(m, s) \;\wedge\; IsPresent(w)$

(m=mineral; s = sediment; w = water; l = lab; t = tool)

It is clear from the example that formulae in Horn clauses (implication) format is better suited for computation, since the implication form is similar to the conditional "if…then…" statement used in programming languages. The Horn clause form is used in the knowledgebase, described in section 5.2.1 for the knowledge optimization evaluation.

"Mineral", "sediment", "water", "lab" and "tool" are described as "entities of interest" and form is the environmental stimuli received. This means, environmental stimuli received pertaining to any of the entities, will change the belief (quantification) of the predicate of which the entity is an argument.

3.2.2 Evidence and logical constraints

In this study, information acquired about entities of interest, are referred to as "evidence". For example, the lab on-board the fictional rover detects a "mineral = hematite" and "sediment = clay" in a soil sample. The evidential items are the constants (e.g. "hematite" and "clay"), which are represented by the variable symbols m and s ("mineral" and "sediment"). Before quantification can take place, the predicate arguments (variable symbols) are replaced by the evidence in a process called grounding. Using the example above, the predicate $\neg DetectedIn(m, s)$ will be ground using the evidence "hematite" and "clay" to become the ground predicate, $\neg DetectedIn(hematite, clay)$.

Any combination of predicates in the knowledgebase is called a world [93] of the knowledgebase. The knowledgebase may have any number of worlds, depending on the subset of predicates selected. When the predicates are grounded by the evidence, they become possible worlds of the knowledgebase. A possible world represents the state of the knowledge in the

knowledgebase at a specific point in time, given the evidence. The formulae in the knowledgebase represents a set of hard logical constraints on all possible worlds of the knowledgebase [6, 36]. Each formula in the knowledgebase represents a logical constraint. This means that a world which violates even one formula has zero probability under inference.

When there is no uncertainty in the evidence, expert systems use logical formalisms for inference. This means each predicate of a formula, given the evidence, evaluates to either true or false and thereby either satisfies or violates the formula completely. When there is uncertainty in the evidence, expert systems use statistical formalisms for inference on the knowledgebase. Each predicate evaluates to true, but with a probability, indicating the degree of confidence in the truth of the formula. Therefore, a world which violates a formula is less probable, but not impossible. Statistical formalisms "soften" the hard, logical constraints of the knowledgebase, by creating a probabilistic reasoning model which enables statistical inference on the formulae in the knowledgebase.

3.2.3 Completeness and consistency

Completeness of the knowledge, indicates the degree of representation of expert knowledge within the knowledgebase. If all possible knowledge needed for inference is formulated within the knowledgebase, it is said to be complete. However, in dynamic real-world environments, completeness of the knowledgebase can never be guaranteed. It is possible to encounter un-defined objects or unexpected interactions between objects (known or unknown). Therefore, when modelling an environment, one of the following principles is followed: the closed world assumption (CWA) or the open world assumption (OWA) [6, 94, 95]. The closed world assumption is based on a minimum model of the world and assumes the knowledge about the environment is *complete*. This means that, unless it is known that a formula is true, it must be assumed to be false. The open world assumption however, assumes that the knowledge representing the environment is *incomplete*. This means that any information not explicitly specified, is considered *unknown*, but not false.

The knowledge is considered to be consistent, if there are no changes made to any of the ground predicates, during inference. While consistency is not a requirement for the representation of the knowledge in the knowledgebase, it is important for efficient reasoning. Under the closed world assumption, the knowledge base is considered consistent and, by not allowing new knowledge to be added, remains consistent. However, in a dynamic environment, the knowledge may be augmented with newly encountered environmental data or a domain expert

may provide additional knowledge. Therefore, for an autonomous system functioning in a dynamic environment, the open world assumption is more suitable.

In cognitive processing, the knowledgebase is redefined as the long-term memory (see section 4.1), which is used in the cognitive reasoning process. In memory recall, optimal memory is recalled, to assist in action selection and execution, by the central executive. The PSO algorithm is selected as memory optimization method, because of the simplicity and scalability of its architecture and control parameters. The long-term memory (knowledgebase) is the search space of the PSO. The next section provides a brief overview of the standard and set-based PSO algorithms. The set-based PSO algorithm forms the basis of the memory recall central executive for high-level cognitive control in single-task robotics.

3.3 Information entropy

Shannon's seminal work on information entropy [96], provides a means to quantify the amount of information *gained* (or conversely, uncertainty reduction) from an event, once the outcome (i.e. probability) of that event becomes known.

Formally, for a discrete random variable $X = \{x_1, \dots, x_{n_X}\}$ where $x_i \in X$ is a state of X, and probability mass function $p(x_i)$, the information entropy of the probability distribution of X is calculated as:

$$H(X) = -\sum_i^{n_X} p(x_i)\, log_2\, p(x_i) \tag{3.1}$$

The random variable X may be in any of the n states at any point in time. The set $\{x_1, \dots, x_{n_A}\}$ is the state space of the problem and the states are mutually exclusive, that is X can only be in one state at any point in time. The probability of X being in state i, is represented by $p(x_i)$.

Using the sample scenario for the fictional rover, the quantification of a predicate using information entropy can be described as follows:

Assume the knowledgebase of the rover contains the predicate $IsPresent(X)$. If the objective is the detection of water, then the random variable X represent the state space for the outcome of a sample analysis, e.g. "*water is present*". A state $x_i \in X$ denotes one possible outcome, e.g. "*water detected*" or "*no water detected*". Therefore, the state space for the predicate is the set $\{true, false\}$ and has a dimension of $n = 2$. A probability $P(x_i)$ is a function, mapping each state x_i to a real number [97]. The information entropy, $H(X)$, is then calculated for the probability distribution over the states of X. When the states of X are all equally probable, the entropy is maximized. For example, if the probability of water detected in the sample is $x_1 =$

0.5 and the probability of no water detected is $x_2 = 0.5$. Then, assuming a logarithm with base 2, the entropy $H(X) = 1$ (maximum), i.e. 1 bit of information was gained (or uncertainty has been reduced by 1 bit). If it is known for certain that water was detected, there can only be one state, i.e. "water detected" with a probability of 1. Since the outcome is certain, no new information is gained, and the entropy $H(X) = 0$ (minimum).

Since information entropy is a measurement over a probability distribution, the assignment of the probability is important. There are two schools of thought for probability assignment, the *frequentist* approach and the *Bayesian* approach [98]. The frequentist approach assigns a probability to an event, based on the long-run frequency of an event over a large number of repetitions of an experiment. This approach also implies the availability of data which will be used to count the frequencies. The Bayesian approach subjectively assigns a probability as a degree of belief about an event. The subjectivity of the assignment risks the introduction of unwanted (or incorrect) information during the probability assignment. Neither of these approaches is suitable for the assignment of probabilities to the states of the predicate (i.e. knowledge item), when the information is dynamic. The maximum entropy principle, discussed in detail in section 4.2, provides a more suitable approach for the assignment of a probability distribution over the state space of the predicate. The maximum entropy principle only considers the information received from the environment and is therefore more accurate.

3.4 Particle swarm optimization

Particle swarm optimization (PSO) is a swarm intelligence algorithm, inspired by the movement and behaviour of a flock of birds searching for food, Eberhart and Kennedy developed the standard particle swarm optimization (StdPSO) algorithm [99].

3.4.1 Overview of standard particle swarm optimization

The standard PSO (StdPSO) algorithm is a stochastic optimization algorithm, which has been successfully applied to optimization problems in the fields of engineering and robotics [100-102]. PSO has been successfully applied to problems where the search space is either continuous or discrete.

The swarm of particles moves through a D-dimensional solution space. The position of particle i in the solution space represents a candidate solution, which is defined as a solution vector, $\mathbf{X}_i \in \mathbb{R}^D$. The optimality of the candidate solution is determined by a fitness function, $f(\mathbf{X}_i) \in \mathbb{R}$. The particle's velocity represents the step size and direction of its movement and

is defined by a vector $\mathbf{v}_i \in \mathbb{R}^D$. StdPSO iteratively updates each particle's velocity and position using the following equations:

$$v_{ij}(t+1) = \omega v_{ij}(t) + c_1 r_{1j}\left(y_{ij}(t) - x_{ij}(t)\right) + c_2 r_{2j}\left(\hat{y}_j(t) - x_{ij}(t)\right) \tag{3.2}$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \tag{3.3}$$

where $v_{ij}(t)$ represents the j^{th} element of the velocity vector of particle i, at the t^{th} iteration. An inertia weight ω is applied to the particle velocity. Two key components of the velocity equation are, the cognitive component, $c_1 r_{1j}\left(y_{ij}(t) - x_{ij}(t)\right)$, and the social component, $c_2 r_{2j}\left(\hat{y}_j(t) - x_{ij}(t)\right)$, where $y_{ij}(t)$ represents the j^{th} element of the personal best vector of particle i at the t^{th} iteration and $\hat{y}_j(t)$ represents the j^{th} element of the global best vector of the swarm at the t^{th} iteration. The term, $x_{ij}(t)$, represents the j^{th} element of the current position of particle i at the t^{th} iteration. The two positive real numbers c_1 and c_2 are acceleration constants, used to scale the contributions of the cognitive and social components. The random values, $r_{1j}, r_{2j} \sim U(0,1)$, add a stochastic element to the cognitive and social components. A user-defined inertia weight, ω, is added to the current velocity [103], which, along with the acceleration constants, balances the effect between global search and local search.

The general fitness function for the PSO is defined as

$$f : \mathbb{R}^{n_\phi} \rightarrow \mathbb{R} \tag{3.4}$$

For a minimization problem, the personal best position at the next iteration is calculated as

$$y_i(t+1) = \begin{cases} y_i(t) & if\ f\left(\mathbf{X}_i(t+1)\right) \geq f\left(y_i(t)\right) \\ \mathbf{X}_i(t+1) & if\ f\left(\mathbf{X_i}(t+1)\right) < f\left(y_i(t)\right) \end{cases} \tag{3.5}$$

and for a maximization problem, the personal best position at the next iteration is calculated as

$$y_i(t+1) = \begin{cases} y_i(t) & \text{if } f\left(\boldsymbol{X}_i(t+1)\right) \leq f\left(y_i(t)\right) \\ \mathbf{X}_i(t+1) & \text{if } f\left(\boldsymbol{X_i}(t+1)\right) > f\left(y_i(t)\right) \end{cases} \tag{3.6}$$

For a minimization problem, the global best position at the next iteration is calculated as

$$\hat{y}(t+1) = \begin{cases} \hat{y}(t) & if\ f\left(\mathbf{X}_i(t+1)\right) \geq f\left(\hat{y}(t)\right) \\ \mathbf{X}_i(t+1) & if\ f\left(\mathbf{X}_i(t+1)\right) < f\left(\hat{y}(t)\right) \end{cases} \tag{3.7}$$

and for a maximization problem, the global best position at the next iteration is calculated as

$$\hat{y}(t+1) = \begin{cases} \hat{y}(t) & if\ f\left(\mathbf{X}_i(t+1)\right) \leq f\left(\hat{y}(t)\right) \\ \mathbf{X}_i(t+1) & if\ f\left(\mathbf{X}_i(t+1)\right) > f\left(\hat{y}(t)\right) \end{cases} \tag{3.8}$$

The standard particle swarm optimization algorithm is shown in Algorithm 3.1 below.

Algorithm 3.1 Standard PSO

Initialize an n_x-dimensional swarm, S;
repeat
  for each particle $i \in S$
    -- *set the personal best position*
    if $f(x_i) > f(y_i)$ -- *see eqs. 3.5 - 3.8*
      $y_i = x_i$;
    endif
    -- *set the global best position*
    if $f(x_i) > f(\hat{y}_i)$ -- *see eqs. 3.5 - 3.8*
      $\hat{y}_i = x_i$;
    endif
  endfor
  for each particle $i \in S$
    update the velocity using eq. (3.2)
    update the position using eq. (3.3)
  endfor
until stopping condition is true

The objective of the PSO algorithm is to find the solution in the search space which will minimize or maximize the fitness function, also called the objective function. The fitness function is defined by the designer, based on the type of optimization problem. The algorithm repeatedly iterates through all particles in the swarm, each time evaluating the fitness of the vector, represented by the position of the particle. If the fitness of the current solution (i.e. position) of the particle is better than its personal (previous) best position, the personal best position is replaced with the current position. Similarly, if the fitness of the particle's current solution is better than the fitness of the swarm's best (global best) position, the global best position is replaced with the particle's current position. On conclusion of all iterations, all (or most) of the particles should have converged on the best global solution, which is represented by the global best position vector. The results should be verified to ensure the swarm did get stuck in a local maximum or minimum. This means the swarm has prematurely converged on a position it perceives as an optimum, but a better position exists somewhere else in the search space. The number of iterations and the stopping condition are defined by the designer.

3.4.2 Overview of set-based particle swarm optimization

When the search space is discrete, the velocity and position update eqs. (3.2) and (3.3) cannot be used without re-definition. In [104] and [105] a generic, set-based PSO (SPSO), suitable for optimization problems with a discrete search space, is introduced and a survey of discrete set-based PSO is given in [106]. In SPSO, the search space is the universal set of discourse U, of the optimization problem. The position, $X_i(t)$ of particle i is a subset of elements from U, and represents a candidate solution in the search space. The particle velocity, $V_i(t)$, is defined as a set of operations $\{v_{i,1},\dots,v_{i,k}\} = \{(\pm, e_{n_{i,1}}),\dots,(\pm, e_{n_{i,k}})\}$ where $e \in U$ and $n_{i,j}$ is the index of the jth element of particle i. The number of operations is denoted by k. The operation pair $\left(\pm, e_{n_{i,j}}\right)$ indicates whether the element $\left(e_{n_{i,j}}\right)$ should be added to, or subtracted from, $X_i(t)$. The result of the operation is a new position, $X_i(t+1)$. To remain in accordance with standard PSO velocity and position update equations, new set-based operators are defined for the generic set-based PSO:

- Velocity addition ($\oplus$) : $V_1 \oplus V_2 = V_1 \cup V_2$
- Position difference ($\ominus$) : $X_1 \ominus X_2 = \left(\{+\} \times (X_1 \backslash X_2)\right) \cup \left(\{-\} \times (X_2 \backslash X_1)\right)$
- Velocity - scalar multiplication: $\eta \otimes V$, $\eta \in [0,1]$, is the random selection of $\lfloor \eta \times |V| \rfloor$ elements from V to yield a new velocity.
- Velocity – position addition: $X \boxplus V = V(X)$, where a velocity, V, is applied element by element to the position X. An element is either added to, or removed from, X. The following additional operators perform the addition and removal of elements from X:
  - $\odot^-$ for the removal of elements from a position,

$$\beta \odot^- S = \{-\} \times \left(\frac{N_{\beta,S}}{|S|} \otimes S\right)$$

    A number of elements, specified by β, are randomly selected for removal from the set S, defined by $X(t) \cap Y(t) \cap \hat{Y}(t)$.

  - $\odot^+$ for the addition of elements to a position,

$$\beta \odot_k^+ A = \{+\} \times \textit{k-Tournament selection}\left(A, N_{\beta,A}\right)$$

    A number of elements, specified by β, are selected from the set A, for addition to X. The set of elements, A, are defined by $U \backslash \left(X(t) \cup Y(t) \cup \hat{Y}(t)\right)$. The selection is done using k-tournament selection process, which ensures that the best performing elements are added. The best performing elements are those, which collectively maximize the fitness function.

The set-based velocity equation is:

$$V_i(t+1) = \left(c_1 r_1 \otimes \left(Y_i(t) \ominus X_i(t)\right)\right) \oplus \left(c_2 r_2 \otimes \left(\hat{Y}_i(t) \ominus X_i(t)\right)\right) \oplus \left(c_3 r_3 \odot_k^{+} A_i(t)\right) \oplus \left(c_4 r_4 \odot^{-} S_i(t)\right) \quad (3.9)$$

where $\left(c_1 r_1 \otimes \left(Y_i(t) \ominus X_i(t)\right)\right)$ is the cognitive component and $\left(c_2 r_2 \otimes \left(\hat{Y}_i(t) \ominus X_i(t)\right)\right)$ is the social component. The author added two additional components to the standard PSO equation: $c_3 r_3 \odot_k^{+} A_i(t)$ and $c_4 r_4 \odot^{-} S_i(t)$, where $A_i(t) = U \backslash \left(X_i(t) \cup Y_i(t) \cup \hat{Y}_i(t)\right)$ and $S_i(t) = \left(X_i(t) \cap Y_i(t) \cap \hat{Y}_i(t)\right)$. The acceleration constants are defined as $c_1, c_2 \in [0,1]$ and $c_3, c_4 \in [0, |U|]$. The random numbers, r_1 to r_4, are random values sampled from a uniform distribution, i.e. $r_1, r_2, r_3, r_4 \sim \Omega(0,1)$.

The set-based position update equation is:

$$X_i(t+1) = X_i(t) \boxplus V_i(t+1) \quad (3.10)$$

While the movement of particles through the search space is governed by eqs. (3.2) and (3.3) for a *continuous* search space, eqs. (3.9) and (3.10) govern the movement of particles (sets) through a discrete *set-based* search space. Research involving PSO can be divided into two parts: 1) the application of PSO to optimization problems [107-111] and 2) the improving the efficiency of the PSO algorithm itself [112-116]. An example of the former is the use of the set-based PSO algorithm for optimal placement of virtual machines in cloud and in [117], a saturated control method, using PSO, is developed for three-dimensional spatial tracking of a UAV. An example of the latter is improving the performance of the PSO algorithm for dynamic optimization problems.

There is an intuitive similarity between the individual and social behaviour of the particles in a swarm, and the individual and social rationality of players in a game. However, despite the intuitive similarity, there have been very few attempts to exploit it. Although, PSO has been used to find optimality in game-theoretic problems, for example finding a Nash equilibrium [118]. However, no relevant research could be found which applies game-theory, specifically collaborative game-theory, to the behaviour of a swarm of particles. Therefore, this similarity is exploited in section 6.2, where PSO is combined with cooperative game theory, for the real-time, high-level cognitive control in multi-task robotics. An overview of the applicable games theory foundations and definitions are given in the next section.

3.5 Cooperative games theory

Game theory is a context-free mathematical methodology used to model and analyse interactive decision making among a group of rational decision makers. The decision of each individual, affects the outcome for the group as a whole. Such an interactive scenario is formally modelled as a game being played between all the decision makers, i.e. the players of the game. Referring to eqs. (3.2) and (3.9), by "remembering" the personal-best solution $y_{ij}(t)$, the particle exhibits individually rational behaviour, while "remembering" the social-best solution $\hat{y}_j(t)$, the swarm exhibits socially rational behaviour.

The study of game theory is divided into two subfields: strategic (or non-cooperative) games and coalitional (or cooperative) games [119]. This study will focus on coalitional games which model scenarios where players may choose to cooperate, i.e. form a coalition and binding agreements, which are likely to maximize the utility they will receive, or defect to another more profitable coalition.

Choosing a game model depends, on the specification of the decision problem, which may be strategic or cooperative. However, as noted by [119], some decision problems may require aspects of both coalitional and strategic games. There are also some aspects which are common to both paradigms. An overview of the fundamentals of games theory, as they relate to the robo-cognitive architecture, is given below. Comments throughout the section, help to relate a games theory concept to the proposed robo-cognitive architecture methodology.

Common game theory concepts

An overview of games theory, based on the work of Maschler [119] and Tadelis [120], form the basis of the cognitive process of the robo-cognitive architecture.

The decision problem

For a player to make a decision, he/she must consider the choices he/she has, what the consequence of each choice is and, how the choice will influence his welfare. The decision problem has the following features:

- Actions (A) - represents the player's choices;
- Outcomes (X) - represents the consequence of the actions;
- Preferences, represent the player's ranking of the set of possible outcomes, from most desired to least desired. The preference relation, $\gtrsim$ describes the player's preference, where $x \gtrsim y$ denotes the player's preference of x over y.

In the context of the robo-cognitive architecture, a player is represented by a particle in a swarm in the CG-PSO algorithm, and the preference represents "remembering" the best solution (individually or socially), based on the utility i.e. fitness of the solution. Just like the cognitive and social components govern the trajectory of a particle towards an optimal solution, utility and rationality govern the decision of a player in a game.

Utility and rationality

A player is considered rational when he/she chooses actions which will maximize his well-being. In other words, the player chooses the actions which result in the most favourable outcomes, indicated by the preference relation.

A utility function (also called a payoff function) assigns a real number to each outcome and is used to produce the set of preferences. The payoff function is an ordinal function, defined as:

Definition 3.1*:* A payoff function $u: X \to \mathbb{R}$ represents the preference relation $\gtrsim$ if, for any pair $x, y \in X,\ v(x) \geq v(y)$ iff $x \gtrsim y$. The purpose of the payoff function is to rank a player's preference over various outcomes.

The payoff function extends the features of the decision problem to include the rational preferences (utility) over the outcomes. Given the decision problem description above and the means to numerically evaluate outcomes, a rational player is defined as follows:

Definition 3.2*:* A player facing a decision problem with a payoff function $v(\cdot)$ over actions is rational if he/she chooses an action $a \in A$ which maximizes his utility, that is, $a^* \in A$ is chosen iff $v(a^*) \geq v(a), \forall a \in A$.

Since the objective of the CG-PSO is the formation of coalitions, each with the maximum social utility, the particle endeavours to maximize the social utility of the coalition through its individual contribution. The particle's choice is therefore based on the "promise" of an equal share in the coalition's utility. Obviously, the higher the social utility, the bigger the (promised) individual payoff. This characteristic is similar to a particle swarm converging on the global-best solution.

Static games of complete information

In a static game, a set of players independently choose a set of actions once, which in turn results in a set of outcomes. A static game has two steps:

1. Each player in the game chooses an action independently and simultaneously. This means each player chooses an action without knowing the choices of the other players or interacting with other players.
2. Utility is distributed to each player. Once a player has made a choice, the choice results in a preferred outcome, which in turn result in utility calculated by the payoff function.

A game with complete information extends the knowledge of a decision problem for a single player to include *common knowledge* amongst *all* the players:

- all the possible actions of all the players;
- all the possible outcomes;
- the outcomes of all the players, based on their actions;
- the preferences of every player, over the outcomes, calculated by the payoff function.

The notion of rationality and common knowledge is important and fundamental in the application of games-theory to particle swarm behaviour. Coalition formation through rational individual and social bargaining, forms the foundation of the cognitive process. The social behaviour of the particles is influenced by the common knowledge shared amongst the particles. The cognitive behaviour of each particle is driven by its rationality. The CG_PSO algorithm "shares" the knowledge between all particles by giving each particle a turn to negotiate with every other particle and only evaluates its payoff if there is no objection (see definition 6.7 in section 6.2.2).

3.6 Conclusion

The robo-cognitive architecture proposed in this thesis focusses on the cognitive processes, memory representation, memory recall, action selection and action execution. The overview given in this chapter, forms the theoretical foundation for these cognitive processes. Section 3.1 provided an overview of the working memory models, used in in neuro-cognitive architectures and sections 3.2 and 3.3 gave an overview of knowledge and uncertainty representation which will form the basis for memory representation and quantification developed in the robo-cognitive architecture. Sections 3.4.1 and 3.4.2 gave an overview of both the standard and set-based particle swarm optimization algorithms. The fundamentals of cooperative game theory was discussed in section 3.5. The set-based PSO algorithm and cooperative game theory will form the foundation for memory recall in the design and development of robo-cognitive architectures, developed in chapter 6 and evaluated in chapter 7.

Chapter 4

Working Memory Representation and Quantification

In section 3.2, an overview was given for knowledge representation, using logic formulae. This chapter discusses the development of two robo-cognitive architectures with a novel memory representation structure and a novel memory quantification methodology, for real-time, high level control of a UAV. Both the memory representation structure and the memory quantification algorithm are used in memory recall, developed in sections 6.1 and 6.2. In section 4.1 working memory representation is formally defined and in section 4.2 memory quantification is formally defined and the adaptive entropy fitness quantification (AEFQ) algorithm is developed.

4.1 Working memory representation

The working memory of the robo-cognitive architecture is discussed in the context of high-level control for a UAV. Figures 4.1 and 4.2 show the states and state transitions of the UAV, defined by the domain expert. These state flows are graphical representations of the knowledge about the behaviour of the UAV and defines the LTM of the working memory for the UAV. Each edge between two states represents a memory item (or knowledge) about transitioning from one state to another, and is defined by the domain expert. Each memory item is given an identifier, for example, τ_{10} represents the memory item “arm motors”, i.e. start the motors. The assigned identifier makes computation easier and is arbitrarily defined by the designer (see figure 4.3). The UAV can perform two functions (or tasks), flight control and gripper control. The first diagram represents the flight control task, while the second diagram represent the gripper control task. Although one task may influence the state of the other, each task is independent from the other.

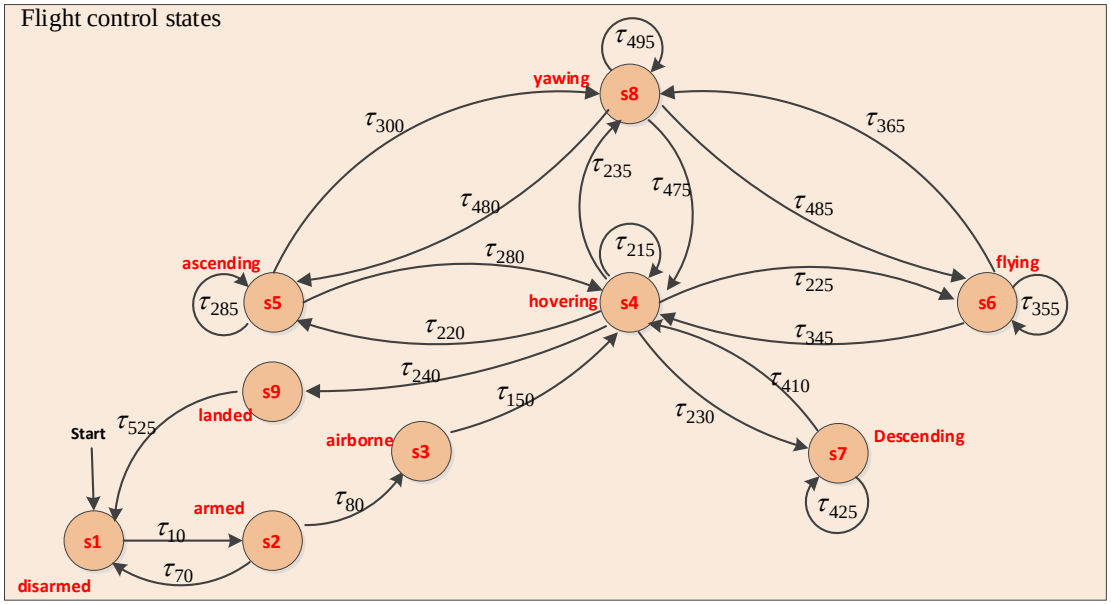

Figure 4.1 UAV Flight control states and state transitions.
The state flow illustrates all the flight control states of the UAV, while the labelled state transitions illustrates valid transitions from one flight state to another.

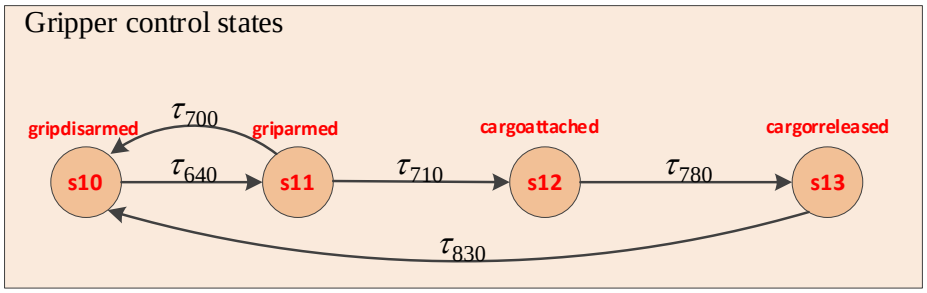

Figure 4.2 Gripper control states and state transitions.
The state flow illustrates all the control states of the on-board gripper, while the labelled state transitions illustrates valid transitions from one gripper state to another.

In order to keep the introduction of the robo-cognitive architecture methodology simple, the LTM used in this study, represent only singleton state transitions. That is, there is only one unique directional edge between two states. However, composite transitions are possible (discussed in section 4.1.2 and shown in figure 4.4).

Given the states and state transitions, shown in figures 4.1 and 4.2, the LTM for the UAV is represented in matrix form. In the LTM, it is assumed that the states are fully connected. In other words, there can be a transition from any state to any other state, including itself. Obviously, this is not always true, for example, there cannot be a transition from state s6 (flying) to state s1 (disarm). The UAV will crash. Therefore, the domain expert defines ("switch on") the valid states by setting the indicator v in eq. (4.3) to "true" or 1, depending on the

implementation. This makes the LTM more flexible, as transitions could be conditionally activated, based on cognitive decision-making. Figure 4.3, shows the LTM, with valid state transitions for the UAV application. Each cell τ_n represents a state transition from a state s_i in row i to a state s_j in column j. Each state transition is assigned a sequence number n to act as identifier during computation. In this thesis, a sequence number in multiples of 5 are assigned to make insertions simpler, but the format of the sequence number is the decision of the designer of the LTM. Valid state transitions are defined by the domain expert and are highlighted in blue in figure 4.3.

		disarmed	armed	airborne	hovering	ascending	flying	descending	yawing	landed	gripdisarmed	griparmed	cargoattached	cargoreleased
		s1	s2	s3	s4	s5	s6	s7	s8	s9	s10	s11	s12	s13
disarmed	s1	τ_5	τ_{10}	τ_{15}	τ_{20}	τ_{25}	τ_{30}	τ_{35}	τ_{40}	τ_{45}	τ_{50}	τ_{55}	τ_{60}	τ_{65}
armed	s2	τ_{70}	τ_{75}	τ_{80}	τ_{85}	τ_{90}	τ_{95}	τ_{100}	τ_{105}	τ_{110}	τ_{115}	τ_{120}	τ_{125}	τ_{130}
airborne	s3	τ_{135}	τ_{140}	τ_{145}	τ_{150}	τ_{155}	τ_{160}	τ_{165}	τ_{170}	τ_{175}	τ_{180}	τ_{185}	τ_{190}	τ_{195}
hovering	s4	τ_{200}	τ_{205}	τ_{210}	τ_{215}	τ_{220}	τ_{225}	τ_{230}	τ_{235}	τ_{240}	τ_{245}	τ_{250}	τ_{255}	τ_{260}
ascending	s5	τ_{265}	τ_{270}	τ_{275}	τ_{280}	τ_{285}	τ_{290}	τ_{295}	τ_{300}	τ_{305}	τ_{310}	τ_{315}	τ_{320}	τ_{325}
flying	s6	τ_{330}	τ_{335}	τ_{340}	τ_{345}	τ_{350}	τ_{355}	τ_{360}	τ_{365}	τ_{370}	τ_{375}	τ_{380}	τ_{385}	τ_{390}
descending	s7	τ_{395}	τ_{400}	τ_{405}	τ_{410}	τ_{415}	τ_{420}	τ_{425}	τ_{430}	τ_{435}	τ_{440}	τ_{445}	τ_{450}	τ_{455}
yawing	s8	τ_{460}	τ_{465}	τ_{470}	τ_{475}	τ_{480}	τ_{485}	τ_{490}	τ_{495}	τ_{500}	τ_{505}	τ_{510}	τ_{515}	τ_{520}
landed	s9	τ_{525}	τ_{530}	τ_{535}	τ_{540}	τ_{545}	τ_{550}	τ_{555}	τ_{560}	τ_{565}	τ_{570}	τ_{575}	τ_{580}	τ_{585}
gripdisarmed	s10	τ_{590}	τ_{595}	τ_{600}	τ_{605}	τ_{610}	τ_{615}	τ_{620}	τ_{625}	τ_{630}	τ_{635}	τ_{640}	τ_{645}	τ_{650}
griparmed	s11	τ_{655}	τ_{660}	τ_{665}	τ_{670}	τ_{675}	τ_{680}	τ_{685}	τ_{690}	τ_{695}	τ_{700}	τ_{705}	τ_{710}	τ_{715}
cargoattached	s12	τ_{720}	τ_{725}	τ_{730}	τ_{735}	τ_{740}	τ_{745}	τ_{750}	τ_{755}	τ_{760}	τ_{765}	τ_{770}	τ_{775}	τ_{780}
cargoreleased	s13	τ_{785}	τ_{790}	τ_{795}	τ_{800}	τ_{805}	τ_{810}	τ_{815}	τ_{820}	τ_{825}	τ_{830}	τ_{835}	τ_{840}	τ_{845}

Figure 4.3 The LTM of the cognitive architecture for the UAV.
Each cell represents a memory item, which is defined as a possible state transition between a state in a row to a target state in a column. Each cell (memory element) is labelled and valid states are highlighted in blue.

In order for the cognitive reasoning process to recall the optimal memory items from the LTM, the central executive needs to quantify the memory item, based on the cue received. The quantification is then used in the evaluation for optimality by the central executive. The LTM

representation and quantification are formally developed in the next sections. Although the working memory representation and quantification discussed in this chapter are described within the context of real-time, high-level control of a UAV, both the representation and quantification may, without the loss of generality, be applied to other autonomous vehicle control problems.

4.1.1 Cue definition

A mission is composed of a set of tasks which must be executed. The tasks of the mission are provided by a domain expert and serve as cues to the cognitive process of the autonomous vehicle. Each cue (task) controls the state to state transitioning of the vehicle, during the mission. The set of cues representing the mission, are defined as,

$$\Phi^c = \left\{\varphi_1^c, \varphi_2^c, \dots, \varphi_{n_{\Phi^c}}^c\right\} \tag{4.1}$$

where φ_i^c, $i = 1,..,n_{\Phi^c}$ is the i^{th} task of the mission. An example of the use of cues is shown in an example in section 4.1.2.

4.1.2 Memory representation

Formally, the LTM is defined as the set of state transitions which governs the behaviour of the UAV (see section 4.1):

$$\text{LTM} = \left\{\tau_1, \tau_2, \dots, \tau_{n_{LTM}}\right\} \tag{4.2}$$

where $\tau_k \in$ LTM, $k = (1, \dots, |\text{LTM}|)$ is a memory item, representing a state transition in the LTM. Each τ_k represents a state transition in the matrix in figure 4.3. The state transition is a tuple,

$$\tau_k = \left(\upsilon, S_\alpha, S_\beta, A_k, F_k, f_j\right) \tag{4.3}$$

where $\upsilon = \{0,1\}$ indicates whether the transition is valid, S_α and S_β are the start and end states of the state transition, respectively, $A_k = \left\{a_1, \dots, a_{n_{A_k}}\right\}$ is a set of actions and $F_k = \left\{p_1, p_2, \dots, p_{n_{F_k}}\right\}$ is the trigger formula for the transition, consisting of a set of simple logic propositions. The function to which τ_k belongs to is indicated by $f_j \in \mathcal{F}$.

The ENV stimuli are defined as,

$$\Phi^r = \left\{\varphi_1^r, \varphi_2^r, \dots, \varphi_{n_{\Phi^r}}^r\right\} \tag{4.4}$$

where φ_i^r, $i = 1,..,n_{\Phi^r}$ is the environmental stimulus received during the cognitive cycle.

The short-term memory items are defined as,

$$\Phi^m = \left\{\varphi_1^m, \varphi_2^m, \dots, \varphi_{n_{\Phi^m}}^m\right\} \tag{4.5}$$

where $\varphi_j^m \in [lb^{m_j}, ub^{m_j}]$, $j = 1,..,n_{\Phi^m}$, defines a short-term memory item, constrained to specified lower and upper boundaries. short-term memory is defined as initial (or default) information provided by the domain expert and may be updated during a cognitive cycle. Both φ_i^r and φ_j^m are used for memory quantification during a cognitive cycle.

Each proposition $p_l \in F_k$, $l = \left(1, \dots, n_{F_k}\right)$ is defined by a domain expert and is a tuple,

$$p_l = \left(\varphi_i^r, logical\_operator, \varphi_j^m\right) \tag{4.6}$$

where φ_i^r and φ_j^m are related by a logical_operator, from the set $\{>, <, =\}$, to form simple propositions of the form:

$$\left(\varphi_i^r > \varphi_j^m\right), \left(\varphi_i^r < \varphi_j^m\right) \text{ and } \left(\varphi_i^r = \varphi_j^m\right) \tag{4.7}$$

Any non-numeric argument is discretized to a numeric value, prior to quantification of F_k.

The indicator υ, the actions $\mathcal{A}$ and all the propositions p_l are defined and maintained by the domain expert. The trigger formula is always conjunctive, i.e. $F_k = (p_1 \wedge p_2 \wedge \dots)$.

The following example illustrates the evaluation of a LTM element, based on cues, environmental (sensory) stimuli and short-term memory during the execution of a mission.

Example: Assume the UAV has taken off and is in a hover state, i.e. $S_\alpha = s_4$. One of the tasks of a mission, $\varphi_1^c \in \Phi^c$ is to fly from a home position H, indicated by the coordinates x_H, y_H and z_H, to a target position T, indicated by the coordinates x_T, y_T and z_T. During runtime, the UAV's current position, indicated by the coordinates x, y, z, are recorded as environmental stimuli. The transition from the "hovering" state to the "flying" state, is subject to some rules. The UAV should only fly if it is explicitly allowed to do so, by the domain expert. This is indicated by setting the indicator υ to 1 (se eq. 4.20). In addition, the UAV should only fly to the specified destination if it is not there already and the current energy level is above a specified minimum level. The state transition, τ_{225}, represents the transitioning from state s_4 (hovering) to s_6 (flying), and is evaluated as follows:

- Input mission cue: $\varphi_1^c = flyto(T)$;
- Read environment stimulus: $\Phi^r = \{\varphi_1^r = (x, y, z), \varphi_2^r = v_{in}\}$;
- Read STM: $\Phi^m = \{\varphi_1^m = (x_T, y_T, z_T), \varphi_2^m = v_{min}\}$;
- Identify states: From state: $S_\alpha = 's4'$ to state $S_\beta = 's6'$;

- Action: $A_{224} = 'flyto(x_T, y_T, z_T)'$;
- Proposition 1: $p_1 = (\varphi_1^r < \varphi_1^m)$;

  (i.e. rule 1: "the current position is less than target position")
- Proposition 2: $p_2 = (\varphi_2^r > \varphi_2^m)$;

  (i.e. rule 2: "current energy levels are greater than the minimum level")
- Function: $f_1 = 1 = "flightControl"$;
- Trigger formula: $F_{224} = (p_1 \wedge p_2)$.

Given the values above, the state transition (see eq. (4.3)) is prepared as follows:

$$\tau_{225} = (1, s4, s6, flyto(x_T, y_T, z_T), (p_1 \wedge p_2), 1)$$

and will be evaluated and actioned (or rejected), based on the quantification of the trigger formula, F_1 (discussed in section 4.2).

However, the robo-cognitive architecture allows for the definition of composite, rule-based transitions between states (shown by the blue line in the example, shown in figure 4.4). This makes the high-level control very flexible, as each composite transition allows for any number of possible transitions between S_α and S_β, each with its own trigger formula. Composite transitions also enable a logical combination of the trigger formulae in disjunctive normal form (DNF), i.e. a disjunction of conjunctions: $((p_1 \wedge p_2) \vee (p_3 \wedge p_4) \vee \ldots)$.

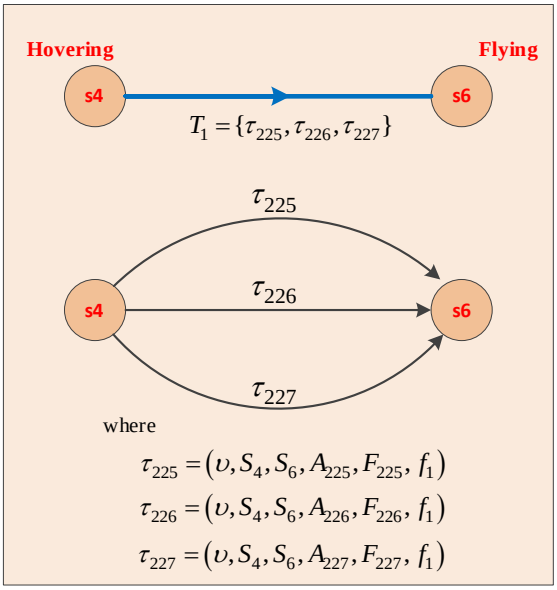

Figure 4.4 Example of a composite state transition.
The diagram illustrates the definition of multiple transitions between state s4 and state s6. The diagram shows that multiple, problem-specific trigger functions and actions may be defined between states.

Composite transitions allow the definition of any number of problem-specific trigger functions, F_k, and corresponding actions, A_k. This capability greatly extends the cognitive reasoning process and therefore, the functionality of the high-level control.

4.2 Working memory quantification

In order to perform the quantification of a state transition $\tau_k \in$ LTM, a problem-specific model is constructed before it is presented to the maximum entropy principle equation for quantification.

4.2.1 Quantification model construction

Given a state transition $\tau_k \in$ LTM the model is formally defined as a tuple:

$$\mathcal{M}_{\tau_k} = (\mathbf{V}, \mathbf{X}, \mathbf{F}, \mathbf{\Lambda}) \tag{4.8}$$

The set of variables are represented by $\mathbf{V} = \left\{\{v^{\mathbb{Q}}\} \cup \{v_1^{\mathbb{P}}, v_2^{\mathbb{P}}, \dots v_{n_{\mathbb{P}}}^{\mathbb{P}}\} \cup \{v_1^{\mathbb{A}}, v_2^{\mathbb{A}}, \dots v_{n_{\mathbb{A}}}^{\mathbb{A}}\}\right\}$ where $v^{\mathbb{Q}}$ is the query variable, $v_p^{\mathbb{P}}$, $p = 1, \dots, n_{\mathbb{P}}$ is a predictor variable, representing a proposition in the trigger formula and $v_l^{\mathbb{A}}$, $l = 1, \dots, n_{\mathbb{A}}$ is an association variable. Note that, since the propositions are independent, they will not have any effect on the query variable, unless relevant associations are defined between the query variable and appropriate predictor variables. The associations are problem-specific and are defined by the user.

Let $m_{\tau_k} = \left|\{v^{\mathbb{Q}}\} \cup \{v_1^{\mathbb{P}}, v_2^{\mathbb{P}}, \dots v_{n_{\mathbb{P}}}^{\mathbb{P}}\}\right|$, and $n_{\tau_k} = 2^{m_{\tau_k}}$, then a $m_{\tau_k} \times n_{\tau_k}$ constraint matrix, $\mathbf{X}$ is the state space of the trigger formula and defines all the joint statements of $\{v^{\mathbb{Q}}\} \cup \{v_1^{\mathbb{P}}, v_2^{\mathbb{P}}, \dots v_{n_{\mathbb{P}}}^{\mathbb{P}}\}$. A binary constraint function, $F(X = x_{ij})$, $i \in n_{\tau_k}$ and $j \in m_{\tau_k}$ assigns a boolean constraint to each variable in the state space. Let $n_V = (1 + n_{\mathbb{P}} + n_{\mathbb{A}})$, then vector $\mathbf{F} = (\langle F_1 \rangle, \langle F_2 \rangle, \dots, \langle F_{n_F} \rangle)$, $n_F = n_V$ are constraint averages for each of the variables in $\mathbf{V}$. The vector $\mathbf{\Lambda} = (\lambda_1, \lambda_2, \dots \lambda_{n_\Lambda})$, $n_\Lambda = n_V$, represents the Lagrange multipliers, calculated for each variable in $\mathbf{V}$.

Each constraint average $\langle F_{n_F} \rangle \in \mathbf{F}$ represents the degree of belief in a proposition and is derived from real-time information (environmental data) received from the environment. The constraint average follows the open world assumption, and is crucial for the accurate quantification of the state transition.

In this thesis, the constraint average is calculated by interpreting a proposition as a degree of believe, (probability), derived from a distance calculation. For example, figure 4.5 illustrates two example state transitions:

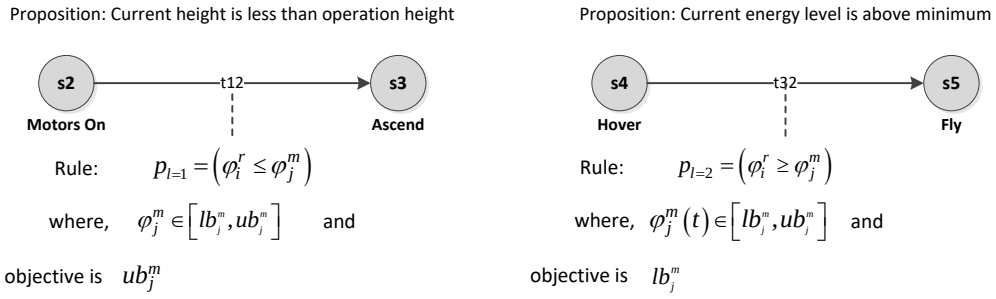

Figure 4.5 Example state transitions with corresponding propositions.
The figure shows two rule definitions for two example state transitions, including the corresponding upper and lower bounds.

A constraint average for the proposition is calculated by measuring the progress of the current runtime parameter φ_i^r, relative to the operational bounds of the mission task. The result is a probability assigned to the proposition. Figure 4.6 illustrates the approach:

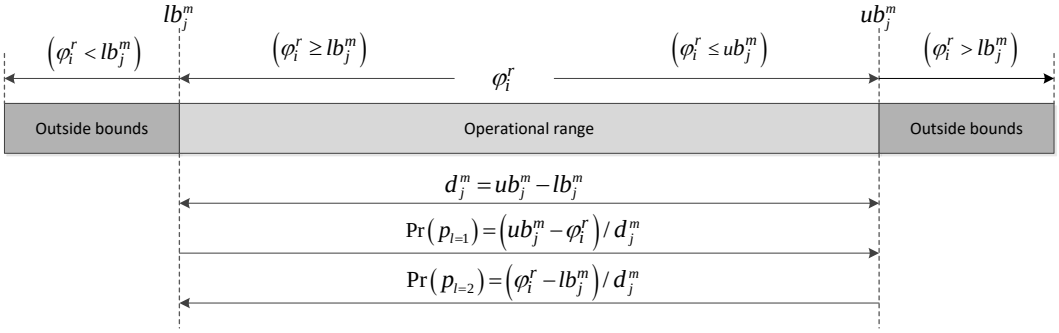

Figure 4.6 Method for constraint average assignment to propositions.
The figure graphically illustrates the method for calculating the constraint average as a probability of the progress, relative to the upper and lower bounds, of the current runtime parameter, as moves towards the objective.

This approach ensures that the constraint average accurately reflects relevant environmental data. This will also ensure that the fitness quantification of the trigger formula for the state transition is based on relevant and correct environmental data.

The rule is translated into a probability as follows:

Firstly, given the proposition p_l, calculate the total operation distance d_j^m, using the upper and lower bounds of the mission argument:

$$d_j^m = ub_j^m - lb_j^m \tag{4.9}$$

Calculate the current distance d_i^r of the runtime argument, φ_j^r with respect to the upper and lower bounds of the mission parameter, φ_i^m, according to the logical operation of the proposition:

$$d_i^r = \begin{cases} \varphi_i^r - lb_j^m \; ; \; if \; p_l = \left(\varphi_i^r \leq \varphi_j^m\right) \\ ub_j^m - \varphi_i^r \; ; \; if \; p_l = \left(\varphi_i^r \geq \varphi_j^m\right) \\ 0 \qquad\quad ; \; if \; p_l = \left(\varphi_i^r \neq \varphi_j^m\right) \\ 1 \qquad\quad ; \; if \; p_l = \left(\varphi_i^r = \varphi_j^m\right) \end{cases} \tag{4.10}$$

Use (19) and (20) to calculate a real valued distance, in the range [0,1], for the proposition:

$$Pr\,(p_l) = \frac{d_i^r}{d_j^m} \tag{4.11}$$

where $Pr\,(p_l)$ represent the relative remaining distance of φ_i^r, within the boundaries lb_j^m and ub_j^m as a probability. Once the distances for each proposition have been calculated, the distances for each of the joint statements can be calculated. To illustrate, let $v^{\mathbb{Q}} = p_0$, $v_1^{\mathbb{P}} = p_1$ and $v_2^{\mathbb{P}} = p_2$, then the state space consists of $2^3 = 8$ joint statements. The joint distances, for the predictor variables are calculated as follows:

$$d_{p_1 p_2} = d_{p_1} + d_{p_2} \tag{4.12}$$

$$d_{p_1 \overline{p_2}} = d_{p_1} + \left(1 - d_{p_2}\right) \tag{4.13}$$

$$d_{\overline{p_2} p_2} = \left(1 - d_{p_1}\right) + d_{p_2} \tag{4.14}$$

$$d_{\overline{p_1 p_2}} = \left(1 - d_{p_1}\right) + \left(1 - d_{p_2}\right) \tag{4.15}$$

The overall distance d_f, represented by the probability distribution over all the propositions of the trigger formula, is calculated by:

$$d_f = \left(d_{p_1 p_2} + d_{p_1 \overline{p_2}} + d_{\overline{p_2} p_2} + d_{\overline{p_1 p_2}}\right) \tag{4.16}$$

With all the joint distances of the joint statements available, the respective constraint averages can now be calculated. Firstly, the constraint average $\langle \mathrm{F}_1 \rangle$ of the query variable p_0 is set to 1.0. The constraint averages for the predictor and association variables are then set as follows:

$$\mathbf{F} = \left(d_{p_0}, \frac{\left(d_{p_1 p_2} + d_{p_1 \overline{p_2}}\right)}{d_f}, \frac{\left(d_{p_1 p_2} + d_{\overline{p_1} p_2}\right)}{d_f}, \frac{\left(d_{p_1 p_2} + d_{p_1 \overline{p_2}}\right)}{d_f}, \frac{\left(d_{p_1 p_2} + d_{\overline{p_1} p_2}\right)}{d_f}, \frac{d_{p_1 p_2}}{d_f}\right) \tag{4.17}$$

Next, the Lagrange multipliers are determined.

The duality between the Lagrange multipliers and the user-defined constraint averages, allows the Legendre transform to be used to derive the Lagrange multipliers:

$$\mathcal{L}_{trans} = \mathbf{\Lambda} = \min_{\lambda_k} \left(ln\, Z(\lambda_1, \lambda_2, \ldots \lambda_k) - \sum_{j=1}^{\mathrm{m}_{\tau_k}} \lambda_j \langle F_j \rangle \right) \tag{4.18}$$

The multipliers are derived by varying the values of λ_k while keeping the constraint average, $\langle F_j \rangle$ fixed, until $\mathcal{L}_{trans}$ reaches a minimum. Table 4.1 shows an example of a

quantification model for a trigger formula containing two propositions, represented by the predictor variables B and C and the query variable, represented by A. The table also shows the association between the query variable and the predictor variables. The associations are represented by AB, AC, ABC. The model contains a $m_{\tau_k} \times n_{\tau_k}$ Boolean constraint matrix, where $m_{\tau_k} = 3$ and $n_{\tau_k} = 8$.

Table 4.1 Illustrative example of a quantification model.

i	*A*	*B*	*C*	*AB*	*AC*	*ABC*
1	1	1	1	1	1	1
2	1	1	0	1	0	0
3	1	0	1	0	1	0
4	1	0	0	0	0	0
5	0	1	1	0	0	0
6	0	1	0	0	0	0
7	0	0	1	0	0	0
8	0	0	0	0	0	0
F	$\langle F_A \rangle$	$\langle F_B \rangle$	$\langle F_C \rangle$	$\langle F_{AB} \rangle$	$\langle F_{AC} \rangle$	$\langle F_{ABC} \rangle$
Λ	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6

For each variable, the vector of constraint averages, **F,** are calculated (eq. 4.17). Each vector element represents the constraint average for a predictive or associative variable. The constraint averages are then used to calculate the vector of Lagrange multipliers, **Λ,** (eq. 4.18). Each element of the Lagrange multiplier vector corresponds to the constraint average for a predictor or associative variable. Once the model is complete, it is used in the fitness quantification, discussed in the next section.

4.2.2 Model-driven quantification

Given the model $\mathcal{M}_{\tau_k}$, the probability distribution, $\mathbf{Q} = \left(q_1, q_2, \ldots q_{n_\mathbf{Q}}\right)$, $n_\mathbf{Q} = n_{\tau_k}$ over the variables (propositions) of the trigger formula can now be calculated. Given the $m_{\tau_k} \times n_{\tau_k}$ constraint matrix and let $i \in n_{\tau_k}$ and $j \in m_{\tau_k}$, the MEP is then formally defined as:

$$\left(q_i | \mathcal{M}_{\tau_k}\right) = \frac{1}{Z(\lambda_1, \lambda_2, \ldots \lambda_k)} e^{-\sum_{j=1}^{m_{\tau_k}} \lambda_j F_j(X = x_i)} \tag{4.19}$$

where $$Z(\lambda_1, \lambda_2, \ldots \lambda_k) = \sum_{i=1}^{\tau_k} e^{\sum_{j=1}^{m_{\tau_k}} \lambda_j F_j(X = x_i)}$$

Z is the partition function which ensures the probabilities are assigned between 0 and 1. The Lagrange multipliers are represented by $\lambda_j, j = 1, \ldots, k$ and $F_j(X = x_i)$ assigns a real-world, domain-specific constraint, to the state i of variable j.

(Refer to [25], chapters 24 and 25 for a detailed discussion on the mathematical derivation of the Legendre transformation and the MEP formula).

Finally, the fitness of the state transition $\tau_k \in$ LTM is calculated as,

$$\Pi = \upsilon \times q_1 \tag{4.20}$$

where $\upsilon \in \tau_k$ and $\upsilon = 1$ indicate a valid state transition and $\upsilon = 0$ indicate an invalid state transition.

Note that any of the resulting probabilities (including marginal probabilities) in the distribution **Q** may now be used in the fitness quantification. However, in this study, only q_1 will be used for fitness quantification, since its value is conditioned on all the predictor variables, i.e. propositions.

Algorithm 4.1 shows the adaptive entropy fitness quantification method:

Algorithm 4.1 Adaptive Entropy Fitness Quantification (AEFQ).			
1:	Input:	: State-transition τ_k	eq. (4.3)
2:		: ENV stimuli, Φ^r	eq. (4.4)
3:		: STM, Φ^m	eq. (4.5)
4:	Output	: Fitness quantification, Π	
5:	Begin		
6:		Initialize model $\mathcal{M}_{\tau_k}$, given $\mathcal{F} \in \tau_k$	eq. (4.8)
7:		Calculate weighted constraint averages **F**	eq. (4.17)
8:		Calculate Lagrange multipliers **Λ**, given **F**	eq. (4.18)
9:		Calculate probability distribution **Q**, given **Λ**	eq. (4.19)
10:		Calculate the fitness Π, given **Q**	eq. (4.20)
11:		Return Π	
12:	End		

During runtime, algorithm 4.1 is applied for the quantification of a state transition. The selected state transition, along with STM and ENV information are passed to the algorithm where the information is used define the quantification model, described in section 4.2. The model is then used by the MEP equation (eq. (4.19)) to assign a probability distribution over the trigger formula of the state transition. The probability distribution is used to assign a fitness to the state transition, which is used during the memory optimization process of memory recall.

Note that, for simplicity, the environmental stimuli are processed as a single set, rather than each individual input element. Prior to the constraint average calculation (line 9), the arguments of the trigger formula of the state transition are ground using the corresponding sensory input parameters. This automation of the grounding process simplifies modification or creation of new propositions.

4.3 Conclusion

This chapter discussed memory representation and the quantification methodology in detail. The methodology described section 4.1 enables LTM to be structured in an extensible way, improving cognitive reasoning and functionality of high-level control. By simplifying the logic structure (rules) of the trigger formulae, maintenance or extension of the LTM is greatly simplified. This is especially important for remotely deployed autonomous vehicles. Moreover, by allowing the multiple state transitions between the same two states, the LTM structure greatly extends the reasoning and functionality of the cognitive architecture, in a flexible way. By including short-term memory and environmental stimuli in the real-time quantification of LTM elements, developed in section 4.2, cognitive reasoning can be performed at a much higher accuracy level, taking real-time information in consideration. Since the quantification produces a probability distribution across all the propositions of the trigger formula, inference can be performed at a much finer level, including using marginal probabilities. The memory representation structure and memory quantification methodology will be used in the cognitive reasoning process during memory recall, developed in sections 6.1 and 6.2.

Chapter 5

Memory Optimization

The two main functions of memory recall are memory quantification and memory optimization. PSO has been chosen for the optimization, due to the simplicity of the algorithm's architecture and the scalability of its parameters. As described in section 3.4.1, PSO is usually applied to numeric optimization problems with an algebraic objective function. The search time of the PSO is directly related to the size of the search space. The larger the search space, the larger the swarm size to cover the search and as a consequence, the longer the swarm will take to converge on the optimal solution. If the search time is constrained too much, the PSO may not find all the optimal solutions. On the other hand, the time the swarm requires to find all the optimal solutions, may be infeasible for a specific problem. These challenges become greater under dynamic conditions. In this research study, the optimization problem is defined as finding the optimal knowledge which will successfully control a UAV in real-time. This means finding the optimal state transitions from the long-term memory, given the ENV information, for every relevant state and in real-time. For this problem, the "objective function" of the PSO is abstractly defined as the optimal knowledge found (AM) in the search space (long-term memory), which is composed of discrete and complex elements (state transitions). The approach, therefore, raises three important questions whether PSO would be suitable for the optimization stage of the memory recall process, under both static and dynamic conditions:

1. Will the PSO algorithm find all optimal solutions (completeness)?
2. Will each memory item be quantified accurately (information gain)?
3. Will the PSO algorithm find the optimal solutions in time (convergence time)?

To answer these questions, two PSO algorithms, AEStd-PSO and AESet-PSO, are developed and discussed in section 5.1. These algorithms are variants of the StdPSO (section 3.4.1) and SPSO (section 3.4.2), adapted for knowledge quantification and optimization. Both algorithms use the adaptive entropy-based fitness quantification method (developed in section 4.2) for fitness evaluation of potential solutions. Fundamentally, the difference between the two algorithms is how particles represent potential solutions in the search space and how the trajectories of particles are calculated, i.e. how the particles moves through the search space. In the AEStd-PSO algorithm, a particle represents a single solution in the search space. The velocity and positioning of the particle is calculated using a sequential index assigned to the solution. On the other hand, in the AESet-PSO, a particle represents a set of solutions. Particle

velocity and positioning is calculated using set-based operators, redefined to retain the cognitive and social influences of the swarm during the optimization process. Two benchmark problems are defined to investigate questions 1 – 3 in section 5.2.2. The first benchmark problem is used to empirically evaluate these questions under uncertain, but static conditions, while the second benchmark problem empirically evaluate these questions under uncertain and dynamic conditions.

5.1 Methodology

Since the objective is to evaluate the PSO algorithms' suitability for knowledge optimization, the data for quantification is synthesized and kept constant. The long-term memory is defined as a set of conjunctive normal form predicate logic formulae (as discussed in section 3.2), converted from a large set of Horn clause formulae. Each memory item in the long-term memory represents one of these logic formulae. Since no inference will be performed in this evaluation, the long-term memory is simplified by removing all logical connectives from the memory items in the long-term memory. There is also no subsumption of memory items in the long-term memory, that is, each memory item in the long-term memory is independent from any other memory item. However, the same predicate may occur in more than one memory item. Note, for this evaluation, the memory item, $\tau_k \in$ LTM is defined differently from eq. (4.3) in section 4.1.2.:

$$\tau_k = \left\{\mathcal{p}_1, \mathcal{p}_2, \dots, \mathcal{p}_{n_{|\tau_k|}}\right\}; \tag{5.1}$$

where $\mathcal{p}_j \in \tau_k$, where $j = 1, \dots, n_{|\tau_k|}$, is a predicate of τ_k.

Each predicate $\mathcal{p}_j$, is defined as,

$$\mathcal{p}_j \triangleq pred\left(\alpha_1, \alpha, \dots, \alpha_{n_{\mathcal{p}_j}}\right); \tag{5.2}$$

and $\alpha_m \in \mathcal{p}_j$, $m = 1, \dots, n_{\mathcal{p}_j}$, is an argument (or random variable) of $\mathcal{p}_j$ (see section 3.2.1).

The predicate symbol, $pred$, is the *relation* of the arguments, $\left(\alpha_1, \alpha_2, \dots, \alpha_{n_{\mathcal{p}_j}}\right)$. The relation and the arguments are specified by the domain expert. The number of arguments m, defines the arity of the predicate, $\mathcal{p}_j$. To simplify the identification and addressing of the memory items in the LTM, each τ_k is assigned an integer index value, of type $\mathbb{Z}^+$. This index represents the position (or address) of the memory item in the LTM and is used by the PSO for particle trajectory calculation.

The objective of the experiments is the evaluation of the PSO algorithms' suitability for knowledge optimization. Two conditions are evaluated: knowledge optimization under static conditions and knowledge optimization under dynamic conditions (see experimental evaluation in section 5.2). The data used in the datasets is synthesized and remains constant for all experiments. During the experiments for static conditions, the environmental stimuli, Φ^r, and short-term memory information, Φ^m, are defined, containing all the arguments of all the predicates, defined in the control set (see section 5.2.1). Each element of Φ^r, is set to 0, while the lb_j^m of Φ^m is set to 0 and the ub_j^m is set to a synthesized value, obtained from an environmental data file. Under static conditions, the values of both Φ^r and Φ^m are kept constant for all test runs. For the evaluation of the algorithms under dynamic conditions, three data files with synthetic environmental data are used. To simulate dynamism (or volatility) in the environmental data, the experiment cycle through the three environmental data files, at a predefined rate, defined by a parameter, $\mathcal{V}_{\text{ENV}}$. Under dynamic conditions, each element of Φ^r, is set to 0, while the lb_j^m of Φ^m is set to 0 and the $ub_j^m$is set to the new synthesised value, obtained from the new environmental data file. The approach is described in detail in the experimental evaluation in section 5.2.

5.1.1 The optimized working memory

The result of the PSO execution is the optimized activated memory. Each element of the activated memory represents the optimal memory item, selected from the LTM. For this evaluation, the activated memory is defined as,

$$\text{AM} = \{\tau_1^*, \tau_2^*, \dots, \tau_{n_{\text{AM}}}^*\} \tag{5.3}$$

where τ_k^*, $(k = 1, \dots, n_{\text{AM}})$, represents an optimal memory item, determined by the fitness quantification of the predicates of the memory item. The memory item τ_k^* is defined as,

$$\tau_k^* = (f(\tau_k), \tau_k) \tag{5.4}$$

where, τ_k is a memory item in the LTM, and $f(\tau_k)$, is the quantification of τ_k, calculated by the AEFQ algorithm. The resulting AM is a reduced set of weighted memory items, where the weight of each memory item, is represented by the quantification $f(\tau_k)$.

Knowledge optimization involves searching the discrete search space (LTM) containing a set of complex solutions (τ_k). The search space has a single dimension, indexed with an integer value. Each potential solution in the search space is a complex memory item which requires quantification, prior to evaluation by the PSO. This optimization problem is significantly different from the optimization problems for which the StdPSO and the SPSO were designed. Therefore, both the StdPSO and SPSO algorithms are modified to be applied to a knowledge

optimization problem. Two variant PSO algorithms, the AEStd-PSO and AESet-PSO, are developed for the StdPSO and SPSO, respectively. These two PSO variant algorithms are developed in sections 5.1.2 and 5.1.3 below.

5.1.2 The AEStd-PSO Algorithm

In AEStd-PSO (Algorithm 5.1), a particle, i, represents a memory item in the LTM, where the position of the particle is defined as, $x_i \in \mathbb{Z}^+$, which is the index value of the memory item. To calculate the step size and direction of a particle i searching a discrete and finite logic search space, eqs. (3.2) and (3.3) are modified:

$$v_i(t+1) = \omega v_i(t) + c_1 r_1\big(y_i(t) - x_i(t)\big) + c_2 r_2\big(\hat{y}_i(t) - x_i(t)\big) \tag{5.5}$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{5.6}$$

Equations (5.5) and (5.6) are similar to eqs. (3.2) and (3.3), except, the terms of the expressions now represent a single dimension only. The term, $x_i(t)$, represents the current position of particle i, which is the integer index of the memory item, in the LTM. The inertia weight, ω, acceleration constants, $c_1$and c_2, and random stochastic parameters, r_1 and r_2, are of type $\mathbb{R}$. The resulting velocity, $v_i(t+1)$, is of type $\mathbb{R}$ as well. However, the particle position is the index of a memory item, and must be of type $\mathbb{Z}^+$. Therefore, before updating the particle's position, $x_i(t+1)$, in eq. (5.6), the velocity, $v_i(t+1)$, is converted to type $\mathbb{Z}^+$. The conversion is performed as follows: if $v_i(t+1)$ is an even number, halfway between two whole numbers, the even number is returned, otherwise the next even number is returned. For example, if the velocity value is 4.5, it is converted to 4, and if it is 5.5, it is converted to 6.

In StdPSO, the search space is iteratively searched for an optimal memory item, given the environmental stimuli. The particles continuously move closer together until they converge on the optimal solution. In AEStd-PSO algorithm, an optimal solution is any memory item which satisfies the open world assumption, given the environmental stimuli, Φ^r. That means any memory item where $f(\tau_k) > 0$.

The AEStd-PSO algorithm is shown below:

Algorithm 5.2 Adaptive entropy-based standard PSO (AEStd-PSO)

(Refer to section 3.4.1 for a detailed overview of the Std PSO algorithm, including variables).

1: Input : Long-term memory, LTM
2: : Environmental information, ENV
3: : Short-term memory, STM

Output : Activated memory, AM

-- Set the PSO parameters

N is the number of particles

w is the inertia weight

c_1, c_2 is the acceleration constants

ξ is the number of iterations

-- Prepare PSO variables

x_i be position of particle i

v_i be the velocity of particle i

y_i be the initial particle best position

$\hat{y}$ be the swarm best position

τ_k be a memory item represented by particle i, at index position x

$\mathcal{p}_j$ be the jth predicate in τ_k

$f(\mathcal{p}_j)$ be the fitness of predicate $\mathcal{p}_j$

$f(x_i)$ be the fitness of particle i

$f(y_i)$ be the pBest fitness of particle i

$f(\hat{y})$ be the gBest fitness of the swarm

-- Initialize a swarm of *N* particles, randomly selected from the LTM

-- Activate the swarm

Repeat

    If there is a change in the environmental stimuli (ENV)

        Reinitialize a swarm of N, randomly selected from the LTM

    Endif

    For $r = 1, \ldots, \xi$

        For $i = 1, \ldots, N$

            Calculate v_i using eq. (5.5)

            Calculate x_i using eq. (5.6)

            -- Evaluate the fitness of the particle i, representing memory item, τ_k

            For each predicate $\mathcal{p}_j \in \tau_k$

                Set parameter Φ^m, using $\mathcal{p}_j$ arguments (see section 4.1.2)

                Set parameter Φ^r, using $\mathcal{p}_j$ arguments (see section 4.1.2)

                -- Call the AEFQ process to quantify the memory item

                $f(x_i) = f(x_i) + AEFQ(\tau_k, \Phi^m, \Phi^r)$ -- see algorithm 4.1

            Endfor

            If $f(y_i) < f(x_i)$ -- see section 3.4.1

                $y_i = x_i$

                $f(y_i) = f(x_i)$

            Endif

            If $f(\hat{y}) < f(x_i)$ -- see section 3.4.1

46: $\hat{y} = x_i$
47: $f(\hat{y}) = f(x_i)$
48: Endif
49: Endfor
50: Endfor
51: *-- Upon convergence of the swarm, construct the optimal memory*
52: *-- item, τ_k^* and add it to the activated memory, AM*
53: If $f(\hat{y}) > 0$
54: Construct τ_k^* by concatenating $f(\hat{y})$ and τ_k -- see eq. (5.4)
55: Add τ_k^* to AM
56: Endif
57: Until cognitive process terminated -- see section 3.4.1

5.1.3 The AESet-PSO Algorithm

AESet-PSO (Algorithm 5.2) is a variant of the SPSO, where particles represent sets of candidate solutions. The SPSO was designed for the optimization of a discrete, random search space, since candidate solutions are randomly selected across the entire solutions space and included in a particle set. The inclusion or exclusion of candidate solutions in a set simulates particle movement through the search space, as shown in the particle velocity and positioning operators defined below. Note that, unlike StdPSO, set-based PSO does not require the LTM to be ordered. AESet-PSO interprets the velocity and positioning equations of the standard PSO (eqs. (3.2) and (3.3)), in terms of set operations. A particle represents a set of memory items and the algebraic operations of eqs. (3.2) and (3.3) are redefined as set-based operations:

Let:

x_i be the *set* of memory items representing the current position of particle *I*,

y_i be the *set* of memory items representing the personal best position of particle *I*,

$\hat{y}$ be the *set* of memory items representing the global best position of the swarm,

c_{cog}, c_{soc} be the cognitive and social accelerators respectively.

then

$$v_i(t+1) = \left[f\left(r_{cog}c_{cog}\right) \cup \left(d_{cog}\right)\right] \cup \left[f\left(r_{soc}c_{soc}\right) \cup \left(d_{soc}\right)\right] \tag{5.7}$$

$$x_i(t+1) = max_\epsilon\left(x_i \cup v_i(t+1)\right) \tag{5.8}$$

where

Cognitive Difference: $d_{cog} : y_i \cup (x_i \backslash y_i)$

The difference between the particle's personal best set y_i and the particle's current set x_i is defined as the unification of y_i and the set-theoretic difference between y_i and x_i. That is, all the elements in the particle's personal best set are retained and the elements in x_i which are not in y_i are included in the difference set.

Social Difference: $d_{soc} : \hat{y} \cup (x_i \backslash \hat{y})$

The difference between the swarm's global best set $\hat{y}$ and particle's current set x_i is defined as the unification of $\hat{y}$ and the set-theoretic difference between $\hat{y}$ and x_i. That is, all the elements the swarm's best set is retained and the elements in x_i which are not in $\hat{y}$ are included in the difference set.

Cognitive Velocity: $v_{cog} : f(r_{cog} c_{cog}) \cup (d_{cog})$

The cognitive velocity is derived by a user-defined function, which selects c_{cog} random elements from the LTM. A random number r_{cog} is iteratively, i.e. c_{cog} times, selected from the range [1, |LTM|] and the element (state transition) at index r_{cog} is added to d_{cog}.

Social Velocity: $v_{soc} : f(r_{soc} c_{soc}) \cup (d_{soc})$

The social velocity is derived by a user-defined function, which selects c_{soc} random elements from the LTM. A random number r_{soc} is iteratively, i.e. c_{soc} times, selected from the range [1, |LTM|] and the element (state transition) at index r_{soc} is added to d_{soc}.

Particle Velocity: $v_i(t+1) = v_{cog} \cup v_{soc}$

The resulting velocity $v_i(t+1)$ is the union of the elements of cognitive velocity v_{cog} and the elements of the social velocity v_{soc}.

Particle Position: $x_i(t+1) = max_\epsilon(x_i \cup v_i(t+1))$

In order to preserve the fittest elements from one iteration to the next, an elitism parameter ϵ, is introduced [48]. The elitism parameter specifies the number of fittest elements to include in the particle's new position set. The new position $x_i(t+1)$ is derived by selecting the top ϵ elements from the union of the current position x_i and the velocity $v_i(t+1)$. The selection of the top ϵ elements is denoted by $max_\epsilon(\cdot)$ and scales the set of solutions.

Note the absence of the inertia weight applied to the particle's current velocity. In the standard PSO, the inertia weight w, along with the accelerator constants c_1, c_2 control the granularity of the exploration. In set-based PSO, the accelerator constants c_1, c_2 control the granularity by specifying the size of the random set of new elements to be added. Similarly, the inertia weight w, would specify the size of the subset of elements (the inertia set) to be selected from

the velocity set. However, it would serve no purpose to add the inertia set again, because when calculating the new position set, the velocity set is already added in full to the current position set. Therefore, when calculating the difference sets d_{cog} and d_{soc} at the next iteration, the new position already includes the velocity elements.

The AESet-PSO algorithm is shown below:

Algorithm 5.3 Adaptive entropy-based set PSO (AESet-PSO)

(Refer to section 3.4.2 for a detailed overview of the Set-based PSO algorithm, including variables).

Input : Long-term memory, LTM
: Environmental information, ENV
: Short-term memory, STM
Output : Activated memory, AM

-- Set the PSO parameters, where
N is the number of particles
Φ is the particle size
c_1, c_2 is the acceleration constants
ξ is the number of iterations

-- Let,
x_i be a set of Φ memory items, represented by particle i
v_i be the velocity set of particle i
y_i be particle i's best set of memory items
$\hat{y}$ be the swarm's best set of memory items
τ_k be a memory item of particle i
$\mathcal{p}_j$ be the jth predicate in τ_k
$f(\mathcal{p}_j)$ be the fitness of predicate $\mathcal{p}_j$
$f(x_i)$ be the fitness of particle i
$f(y_i)$ be the pBest fitness of particle i
$f(\hat{y})$ be the gBest fitness of the swarm
ϵ be an elitism parameter *-- see eq. (5.8)*

-- Initialize a swarm of N particles with φ randomly selected
-- memory items, each

-- Activate the swarm
Repeat
If there is a change in the environmental stimuli (ENV)
--Reinitialize a swarm of N particles with φ randomly selected
--memory items, each
Endif
--Start iterations

35: For $r = 1, \ldots, \xi$
36: For $i = 1, \ldots, n$
37: Calculate the set v_i -- using eq. (5.7)
38: Calculate the set x_i -- using eq. (5.8)
39: *-- Evaluate the fitness of the particle i, representing memory item, τ_k*
40: For each memory item $\tau_k \in x_i$
41: For each predicate $\wp_j \in \tau_k$
42: Set parameter Φ^m, using $\wp_j$ arguments -- see section 4.1.2)
43: Set parameter Φ^r, using $\wp_j$ arguments -- see section 4.1.2)
44: -- Call the AEFQ process to quantify the memory item
45: $f(\wp_j) = f(\wp_j) + AEFQ(\tau_k, \Phi^m, \Phi^r)$ -- see algorithm 4.1
46: Endfor
47: $f(x_i) = f(x_i) + f(\wp_j)$
48: Endfor
49: *-- Update personal and global best values*
50: If $f(y_i) < f(x_i)$ -- see section 3.4.2
51: $f(y_i) = f(x_i)$
52: $y_i = x_i$
53: Endif
54: If $f(\hat{y}) < f(x_i)$ -- see section 3.4.2
55: $f(\hat{y}) = f(x_i)$
56: $\hat{y} = x_i$
57: Endif
58: Endfor
59: Endfor
60: *-- Upon convergence of the swarm, construct the optimal memory*
61: *-- item, τ_k^* and add it to the activated memory, AM*
62: If $f(\hat{y}) > 0$
63: Construct τ_k^* by concatenating $f(\hat{y})$ and τ_k -- see eq. (5.4)
64: Add τ_k^* to AM
65: Endif
66: Until cognitive process terminated -- see section 3.4.1

5.2 Experimental Evaluation

This section describes the experimental evaluation of the performance of the two PSO algorithms. A statistical comparison is performed between the AEStd-PSO and the AESet-PSO algorithms, in section 5.2.7. The null and alternative hypothesis defined below, are statistically evaluated using the performance measures defined in section 5.2.3.

For the statistical comparison, the null hypothesis $\boldsymbol{H_0}$ is defined as:

"*There is no tendency for the performance of one PSO algorithm to be significantly higher (or lower) than the other when optimizing a logical search space under uncertain and dynamic conditions*".

The alternative hypothesis $\boldsymbol{H_A}$ is defined as:

"*There is a tendency for the performance of one PSO algorithm to be significantly higher (or lower) than the other when optimizing a logical search space under uncertain and dynamic conditions*".

The performance of each PSO algorithm is evaluated using three LTMs, increasing in size, using both static and dynamic environmental data. Section 5.2.1 describes the datasets used in the experiments. The benchmark problems and performance measures used in the evaluation are described in sections 5.2.2 and 5.2.3, respectively. Section 5.2.4 describes the PSO parameters selected for the experiment.

5.2.1 Datasets

The Knowledge Base

To be able to evaluate the performance of the algorithms on a large LTM, an extensive set of Horn clauses, produced by the Sherlock system [121], was used in the experiments. The Sherlock system constructed the set of Horn clauses programmatically from the internet. The dataset was first cleansed by removing duplicate clauses and any garbage data in the dataset. The cleansed Horn clauses were then converted to conjunctive normal form formulae, resulting in a test LTM that contains 30,912 memory items, 4,821 relations (predicates) and 137 classes (arguments). All memory items in the source LTM were grouped, based on context, i.e. all related memory items were stored together in the LTM. Since PSO stochastically explores the LTM, two factors influence the performance of PSO: the size $\mathcal{S}_{\text{LTM}}$ of the LTM and the volatility, $\mathcal{V}_{\text{ENV}}$, of the environmental data. $\mathcal{S}_{\text{LTM}}$ is the number of memory items in the LTM and $\mathcal{V}_{\text{ENV}}$ is the frequency at which new environmental data is observed in the environment. For the experiments, three LTMs of different sizes were created: $\mathcal{S}_{\text{LTM}} = \{small, med, large\}$, where $small \cong (10{,}000)$, $med \cong (20{,}000)$ and $large \cong (30,000)$ memory items in the LTM. The volatility of the environmental data was set as $\mathcal{V}_{\text{ENV}} = (low, medium, high)$. The frequencies for $low, medium\ and\ high$ are defined in table 5.3.

The Control Set

In order to evaluate the AM in a controlled manner, a control set of ten predefined memory items was created and inserted at random positions in each of the three test LTMs. The memory

items in the control set act as target memory items for optimization and are used in the performance measures.

Synthetic environmental data

Dynamic environmental data is simulated by three pre-compiled datasets. Each dataset contains a collection of instances with synthesized values, corresponding to the arguments of the predicates in the control dataset. Environmental data values were changed between different datasets.

5.2.2 Benchmark problems

Based on the size ($\mathcal{S}_{\mathrm{LTM}}$) of the LTM and volatility ($\mathcal{V}_{\mathrm{ENV}}$) of the environmental data mentioned above, two types of benchmark problems are defined to evaluate the performance of the PSO algorithms and the AEFQ algorithm:

1. **Benchmark problem 1** - Optimization of different sizes of LTMs, given uncertain environmental data:

In real-world scenarios, there is often a degree of uncertainty about the environmental data received. When quantifying a memory item, this degree of uncertainty impacts the quantification of memory items. Therefore, the performance of the optimization process is impacted in terms of the *completeness* and i*nformation gain* (defined in section 5.2.3 below).

2. **Benchmark problem 2** - Optimization of different sizes of LTMs, given dynamic environmental data:

The AM**,** is produced by optimizing the LTM, using the environmental data available at the time of optimization. In real-world scenarios, new environmental data may be observed at any time. This new environmental data may be completely new or it may be the same environmental data, but with a different degree of certainty. Any change in the environmental data immediately invalidates the current AM, because the memory items in the AM were quantified based on the previous environmental data. Any inference using the AM**,** will therefore be invalid. A new optimized AM**,** has to be created each time the environmental data changes.

Each benchmark problem is applied to both AEStd-PSO and AESet-PSO for each LTM size parameter (*small*, *med* and *large*) to measure completeness and information gain of the resulting AM. The performance measures (defined below) measures the ability of the two PSO algorithms to optimize an AM, using the AEFQ algorithm.

5.2.3 Performance measures

For each of the benchmark problems defined above, the following performance measures are used:

1. The completeness (φ_{AM}) of the AM, represents the number of memory items in the AM. Comparison of this number against the control set indicates how successful the PSO was in finding all the relevant memory items. No difference between the number of memory items in the AM, and the number of memory items in the control set indicates that the PSO has good *exploration* ability.
2. The information gain (ψ_{AM}) of the AM. The maximum information gain of the predicate is the maximum entropy, given the environmental data and is derived from the maximum entropy of its predicates. Therefore, the information gain of the AM, is the cumulative maximum entropy of the memory items it contains. An AM with a high information gain indicates that the PSO has good exploitation ability.
3. The convergence time (τ_{AM}), calculated as the elapsed wall clock time until convergence of the particles. This performance measure is used in the empirical analysis of the execution time of the two PSO algorithms.

5.2.4 PSO parameter selection

The PSO algorithm uses a number of parameters which control the movement of particles through the search space. Table 5.1 shows the standard PSO parameters selected, based on guidelines in [122-124]:

Table 5.1 Inertia Weight and Acceleration Parameters.

Algorithm	Inertia Weight w	Acceleration constant c_1	Acceleration constant c_2	Elitism parameter ϵ
AEStd-PSO	0.715	1.7	1.7	n/a
AESet-PSO	n/a	3	3	15

To select appropriate swarm size and exploration parameters, both the AEStd-PSO and AESet-PSO algorithms were executed for each permutation of the parameters listed in table 5.2.

Table 5.2 Swarm Size and Exploration Parameters.

LTM Sizes	Swarm Sizes	Iterations
10k	5	10,000
20k	20	20,000
30k	50	50,000

The AEStd-PSO and AESet-PSO algorithms were executed for each permutation of the parameters in table 5.2 and the results were evaluated against the control set of memory items.

The graphs in figures 5.1 – 5.3 show the results of the experimental runs of both the AEStd-PSO and AESet-PSO for each of the LTM sizes. The number of particles for each PSO algorithm is shown alongside the algorithm in brackets.

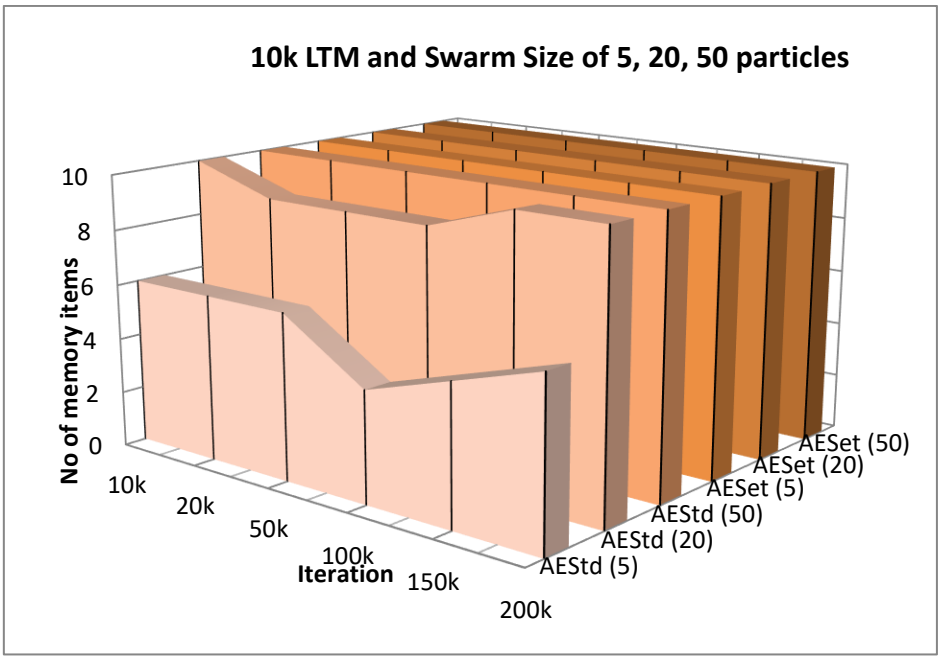

Figure 5.1 Parameters selection results for a **10k** LTM and 5, 20 and 50 particles.
The graph shows the difference in completeness between the AEStd-PSO and AESet-PSO algorithms on a control set of 10 memory items.

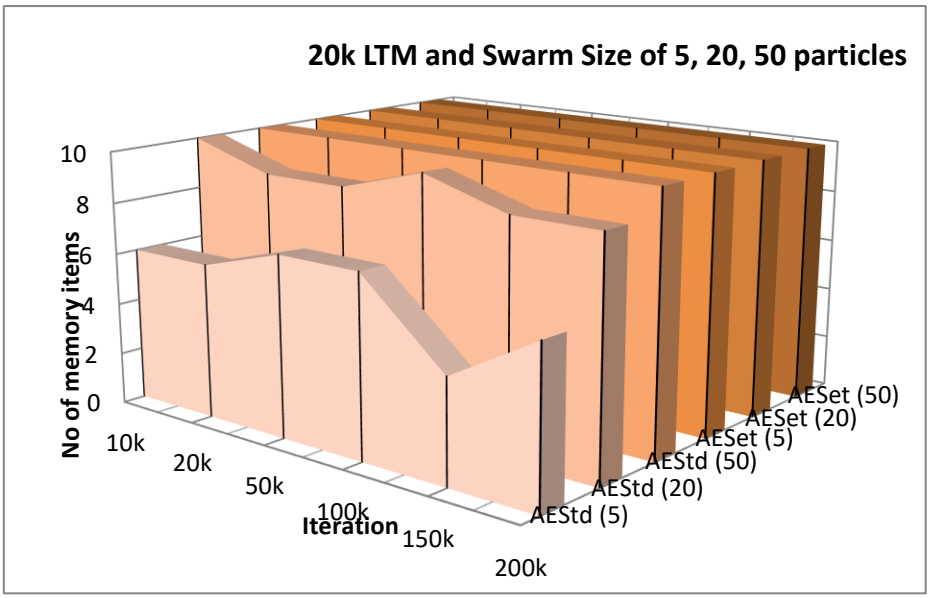

Figure 5.2 Parameters selection results for a **20k** LTM and 5, 20 and 50 particles.
The graph shows that, for the control set of 10 memory items, when the search space increases to 20k elements and swarm size is below 50 particles, the completeness of the AEStd-PSO decreases.

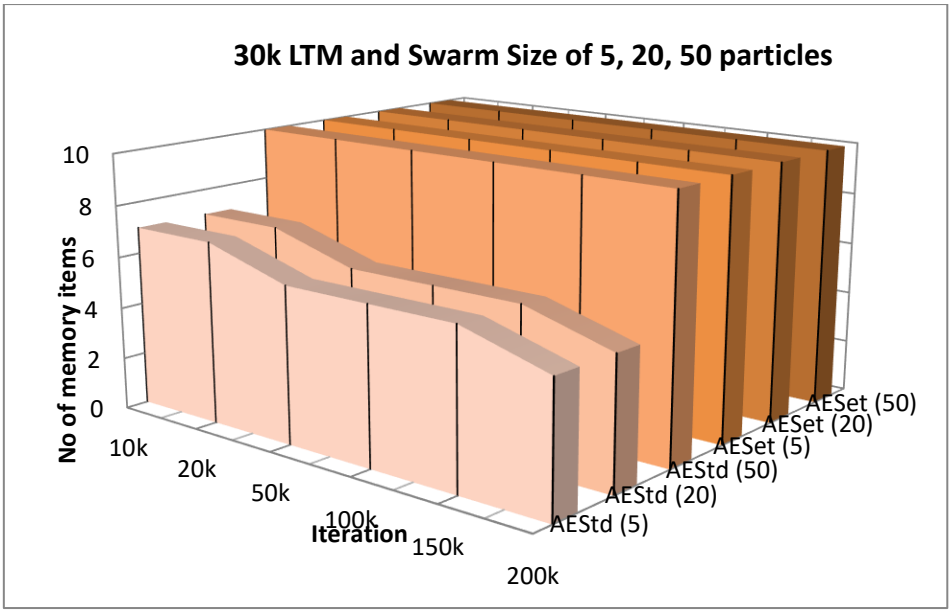

Figure 5.3 Parameters selection results for a **30k** LTM and 5, 20 and 50 particles.
The graph shows that, for the control set of 10 memory items, when the search space increases to 30k elements and swarm size is below 50 particles, the completeness of the AEStd-PSO decreases significantly.

The graphs in figures 5.1 - 5.3 show that the completeness of the AEStd-PSO consistently decreases when the swarm size is below 50 particles and the size of the search space increases. On the other hand, the AESet-PSO consistently achieve completeness for all swarm sizes and all test search spaces. The graphs show that both AEStd-PSO and AESet-PSO are able to successfully find the 10 control memory items with a swarm size of 50 particles and 20,000 iterations. Also, 10,000 iterations would have been sufficient, but since the experiment needs to simulate a dynamic change in the environmental data, as described for benchmark problem 2, 20,000 iterations provide a sufficient time window for the simulation. This means the algorithms successfully finds all the control memory items, before the next dynamic change occurs.

Dynamic change in the environmental data is simulated using the volatility parameters $\mathcal{V}_\varepsilon$ defined in table 5.3.

The volatility values are timed in milliseconds.

Table 5.3 Volatility parameters for environmental data change.				
PSO Algorithm	LTMSize	Volatility(ms)		
		High	Med	Low
AEStd-PSO	10k	10	300	600
	20k	10	500	1000
	30k	10	800	1600
AESet-PSO	10k	100	3500	7000
	20k	100	4500	9000
	30k	100	9000	18000

5.2.5 Experimental architecture and processes

The experiments were executed on an Intel i7 machine with 2.90 GHz Quad Core CPU and 16Gb RAM with MS Windows 8.1 x64 OS. Figure 5.4 shows the core objects and simulation process. A PSO control program uses run-time parameters to input the datasets (LTM, control set and evidence vector) and executes each of the PSO algorithms for each benchmark problem. For benchmark problem 2, the PSO control program simulates the input of dynamic evidence, by periodically introducing a new evidence set, according to the volatility values in table 5.3. The output and execution values of the PSO algorithms are logged by PSO control program for statistical analysis.

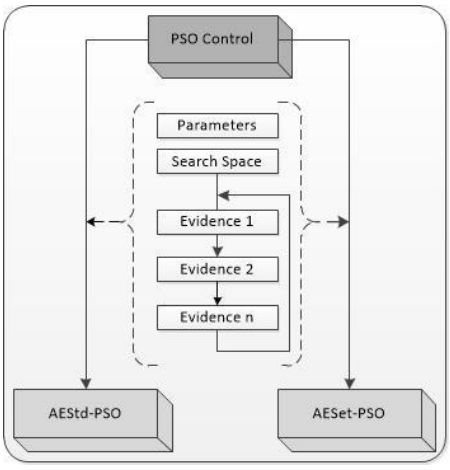

Figure 5.4 Experiment components and simulation process.
A PSO control program governs the application of control parameters, search space and evidence data for the two PSO algorithms, AEStd-PSO and AESet-PSO.

5.2.6 Experimental execution

Empirical analysis is performed using algorithms 5.3 and 5.4. To analyse benchmark problem 1 type problems, i.e. optimization of the LTM under *uncertain* conditions, 30 identical runs are performed for each PSO algorithm. The parameters are selected from tables 5.1 and 5.2. A single set of evidence with pre-defined uncertainty was used for each run. The evidence set remained unchanged between runs. Algorithm 5.3 shows the statistical analysis process for benchmark problem 1. The objective of the process is to test the null hypothesis, H_0, defined in the beginning of section 5.2 and repeated here:

"*There is a tendency for the performance of one PSO algorithm to be significantly higher (or lower) than the other when optimizing a logical search space under uncertain and dynamic conditions*",

and select an algorithm, based on the sum-of-ranks produced by the Mann-Whitney test.

Algorithm 5.4 Preferred PSO algorithm selection for benchmark problem 1

(Refer to the empirical analysis in section 5.2.7 for a detailed explanation of the methods of this algorithm).

```
Begin
    For each LTMSize (δ_LTM)
        For each performance measure     -- see section 5.2.3
            -- Statistically compare AEStd-PSO and AESet-PSO
            Perform Mann-Whitney U test on the performance measure
            If there is no statistically significant difference
                Reject hypothesis H_0
            Endif
            --Select preferred PSO-algorithm using sum of ranks
            Calculate sum-of-ranks η_AEStd−PSO = ∑ R_AEStd−PSO
            Calculate sum-of-ranks η_AESet−PSO = ∑ R_AESet−PSO
            If performance measure = φ_AM OR ψ_AM
                If η_AEStd−PSO > η_AESet−PSO
                    Select AEStd-PSO
                Else
                    Select AESet-PSO
                Endif
            Endif
            If performance measure = τ_AM
                If η_AEStd−PSO < η_AESet−PSO
                    Select AEStd-PSO
                Else
                    Select AESet-PSO
                Endif
            Endif
```

```
26:             Endfor
27:         Endfor
28:     End
```

To analyse the optimization of the LTM under *uncertain* and *dynamic* conditions (benchmark problem 2), 30 identical runs are performed for each PSO algorithm. The parameters are selected from tables 5.1 and 5.2. Three sets of evidence with defined uncertainty were used for each run. The dynamism of the environment was simulated by changing between the three sets of evidence with the frequencies defined in table 5.3. To simplify comparison, the evidence set remained unchanged between runs. This resulted in three sets of results for each run. To quantify the results for each run, the average of the results of the three changes were calculated. Algorithm 5.4 shows the statistical analysis process for benchmark problem 2:

Algorithm 5.5 Preferred PSO algorithm selection for benchmark problem 2

(Refer to the empirical analysis in section 5.2.7 for a detailed explanation of the methods of this algorithm).

```
1:    Begin
2:        For each LTMSize (δ_LTM)
3:            For each Volatility V_ε
4:                For each performance measure          -- see section 5.2.3
5:                    --Statistically compare AEStd-PSO and AESet-PSO
6:                    Perform Mann-Whitney U test on performance measure
7:                    If there is a statistical significant difference
8:                        Reject H_0
9:                    Endif
10:                   --Select preferred PSO-algorithm using sum of ranks
11:                   Calculate sum-of-ranks η_AEStd−PSO = ∑ R_AEStd−PSO
12:                   Calculate sum-of-ranks η_AESet−PSO = ∑ R_AESet−PSO
13:                   If performance measure = φ_AM OR ψ_AM
14:                       If η_AEStd−PSO > η_AESet−PSO
15:                           Select AEStd-PSO
16:                       Else
17:                           Select AESet-PSO
18:                       Endif
19:                   Endif
20:                   If performance measure = τ_AM
21:                       If η_AEStd−PSO < η_AESet−PSO
22:                           Select AEStd-PSO
23:                       Else
24:                           Select AESet-PSO
25:                       Endif
26:                   Endif
```

```
27:                 Endfor
28:             Endfor
29:         Endfor
30:     End
```

5.2.7 Empirical analysis

The objective of the statistical analysis is to assist in the selection of a suitable PSO architecture and parameters, for a specific type of environment. This section discusses the statistical parameters and analysis of the results of the experimental execution of the two algorithms, AE-StdPSO and AESet-PSO. The results are analysed statistically to measure of the performance of the two PSO algorithms under *uncertain* and *dynamic* conditions.

The Mann-Whitney U-test [125] is a two-tailed test which indicates a significant statistical difference between the two PSO algorithms. Using the Mann-Whitney U-test (with a significance level $\alpha = 0.05$ and critical value range [-1.96, 1.96]), statistically significant differences between the results of AEStd-PSO and AESet-PSO are tested. The z-value, based on a sample mean $\bar{x}_U = 450$ and standard deviation $s_U = 67.6$, is tested against the critical value range to determine whether to reject (or not reject) the null hypothesis. The strength of the difference between the AEStd-PSO and the AESet-PSO is also determined by calculating the *effect size* (ϕ) introduced by Cohen [126]:

$$\phi = \frac{|z|}{\sqrt{n}} \tag{5.9}$$

Cohen defines the effect size to be in the range [0, 1] and classified as *small*=0.10, *medium*=0.30 and *large*=0.50.

Table 5.4 contains the analysis results for benchmark problem 1. The hypothesis is tested for each LTMSize/performance measure combination, using the statistical analysis process described in Algorithm 5.3.

Table 5.5 contains the analysis for benchmark problem 2. The hypothesis is tested for each LTMSize/volatility/performance measure combination, using the statistical analysis process described in Algorithm 5.4.

The *convergence time* performance measure τ_{AM} is consistently and significantly higher for the AEStd-PSO than for the AESet-PSO. Therefore, the analysis of *convergence time* (τ_{AM}) is discussed separately, following the analysis of the *completeness* (φ_{AM}) and *information gain* (ψ_{AM}).

The parameter combination represents the conditions of the environment. Both tables show the sum-of-ranks, $\sum R_1$ for AEStd-PSO and $\sum R_2$ for AESet-PSO, as well as the z-score. The sum-of-rank value indicates the success of the PSO algorithm, for a specific parameter combination. The higher the sum-of-ranks value, the closer the PSO came to finding all the solutions, compared to the control set. The z-score indicates the difference in the distributions of the results of each parameter combination. The sum-of-ranks and z-score are both used in the Mann-Whitney U-test and effect size calculation. A *preferred* PSO algorithm is selected by comparing the sum-of-ranks $\sum R_1$ and $\sum R_2$. To assist in the selection of a preferred PSO algorithm, the Mann-Whitney U-test statistically tests if there is a significant difference in the performance of the two PSO algorithms for a specific parameter combination. If there is a significant difference, the null hypothesis H_0 is rejected. The hypothesis is an indication of the level of confidence in selecting the preferred PSO algorithm. If H_0 is rejected, the PSO algorithm with the highest sum-of-ranks value is the most likely to successfully optimize the LTM, for the specific parameter combination. If H_0 is not rejected, the sum-of-ranks values and effect size is considered for each performance measure. The effect size ϕ indicates the size of the statistical difference and is calculated using eq. (5.9). The preferred PSO algorithm is then selected subjectively, based on the user-preference. For clarity, the classification of ϕ is given.

5.2.8 Results

In tables 5.4 and 5.5, values shown in **bold** are the "winning" ones and if there is a significant difference between the "winning" and "losing" value, the "winning value" is shown in ***bold italics***.

Table 5.4 Statistical analysis results for benchmark problem 1.

$\mathcal{S}_{\mathrm{LTM}}$	*Performance measure*	$\sum R_1$	$\sum R_2$	*z-score*	*Reject H_0?*	*Preference*
10k	φ_{AM}	915.0	915.0	0.00	N	equal
	ψ_{AM}	915.0	915.0	0.00	N	equal
	τ_{AM}	***465.0***	1365.0	-6.65	**Y**	AEStd-PSO
20k	φ_{AM}	900.0	**930.0**	-0.22	N	AESet-PSO
	ψ_{AM}	**930.0**	900.0	-0.22	N	AEStd-PSO
	τ_{AM}	***465.0***	1365.0	-6.65	**Y**	AEStd-PSO
30k	φ_{AM}	885.0	**945.0**	-0.44	N	AESet-PSO
	ψ_{AM}	**945.0**	885.0	-0.44	N	AEStd-PSO
	τ_{AM}	***465.0***	1365.0	-6.65	**Y**	AEStd-PSO

The results for benchmark problem 1, in table 5.4, show that there is no *significant* difference between the two PSO architectures (except for τ_{AM}). The null hypothesis is **not** rejected for *completeness* (φ_{AM}) or *information gain* (ψ_{AM}). The reason is, given a *stable* environment where the environmental data do not change, both architectures are able to successfully optimize the LTM. The *small* effect size indicates that the difference between the two PSO algorithms is negligible.

Table 5.5 Statistical analysis results for benchmark problem 2.

$\mathcal{S}_{\mathrm{LTM}}$	$\mathcal{V}_{\varepsilon}$	*Performance measure*	$\sum R_1$	$\sum R_2$	*z-score*	*Reject H_0?*	*Preference*
10k	High	φ_{AM}	***1082.5***	747.5	-2.48	**Y**	AEStd-PSO
		ψ_{AM}	***1201.0***	629.0	-4.23	**Y**	AEStd-PSO
		τ_{AM}	***465.0***	1365.0	-6.65	**Y**	AEStd-PSO
	Med	φ_{AM}	**1016.5**	813.5	-1.50	N	AEStd-PSO
		ψ_{AM}	815.5	**1014.5**	-1.47	N	AESet-PSO
		τ_{AM}	***465.0***	1365.0	-6.65	**Y**	AEStd-PSO
	Low	φ_{AM}	885.0	**945.0**	-0.44	N	AESet-PSO
		ψ_{AM}	**945.0**	885.0	-0.44	N	AEStd-PSO
		τ_{AM}	***465.0***	1365.0	-6.65	**Y**	AEStd-PSO
20k	High	φ_{AM}	**971.0**	859.0	-0.83	N	AEStd-PSO
		ψ_{AM}	**1023.0**	807.0	-1.60	N	AEStd-PSO
		τ_{AM}	***465.0***	1365.0	-6.65	**Y**	AEStd-PSO
	Med	φ_{AM}	***1239.0***	591.0	-4.79	**Y**	AEStd-PSO
		ψ_{AM}	645.0	***1185.0***	-3.99	**Y**	AESet-PSO
		τ_{AM}	***465.0***	1365.0	-6.65	Y	AEStd-PSO
	Low	φ_{AM}	**945.0**	885.0	-0.44	N	AEStd-PSO
		ψ_{AM}	885.0	**945.0**	-0.44	N	AESet-PSO
		τ_{AM}	***465.0***	1365.0	-6.65	**Y**	AEStd-PSO
30k	High	φ_{AM}	***1058.5***	771.5	-2.12	**Y**	AEStd-PSO
		ψ_{AM}	***1092.0***	738.0	-2.62	Y	AEStd-PSO
		τ_{AM}	***465.0***	1365.0	-6.65	Y	AEStd-PSO
	Med	φ_{AM}	***1202.5***	627.5	-4.25	Y	AEStd-PSO
		ψ_{AM}	654.5	**1175.5**	-3.85	Y	AESet-PSO
		τ_{AM}	***465.0***	1365.0	-6.65	Y	AEStd-PSO
	Low	φ_{AM}	870.0	**960.0**	-0.67	N	AESet-PSO

Table 5.6 Continued.							
		ψ_{AM}	**960.0**	870.0	-0.67	N	AEStd-PSO
		τ_{AM}	***465.0***	1365.0	-6.65	Y	AEStd-PSO

The results for benchmark problem 2, in table 5.5, show that when optimizing a 10k LTM under *high* volatility, there is a significant difference between the two PSO algorithms. The AEStd-PSO performs better for *completeness* (φ_{AM}) and *information gain* (ψ_{AM}). The reason for this is the AEStd-PSO evaluates only a single candidate per particle within the limited timeframe, whereas the set-based architecture has to evaluate a set of candidate solutions in the limited time frame. The AEStd-PSO is therefore able to evaluate more candidates before the next environmental data change. When volatility decreases from *Med* to *Low*, there is no significant difference between the two PSO algorithms. This is because both architectures have sufficient time to evaluate the candidate solutions. The sum-of-ranks $\sum R_1$ and $\sum R_2$ indicate the AEStd-PSO performing slightly better on *completeness*, when volatility is Med, but the AESet-PSO performs slightly better on *information gain* when volatility is Low.

When optimizing a 20k LTM under *high* volatility, there is no significant difference between the two PSO algorithms. The AEStd-PSO performing slightly better on *completeness* and *information gain*, but the *small-medium* effect size indicates that the difference is negligible. When volatility is *Med*, there is a significant difference between the PSO algorithms. The AEStd-PSO performs better on *completeness* but the AESet-PSO performs better on *information gain*. When the volatility decreases to *Low*, there is no significant difference. Although the AEStd-PSO performs slightly better on *completeness* and the AESet-PSO performs slightly better on *information gain*, the effect size is *small*, indicating the difference to be negligible.

When optimizing a 30k LTM, there is a significant difference between the two PSO algorithms for both *High* and *Med* volatility. Under *High* volatility, the AEStd-PSO performs better on both *completeness* and *information gain*, while the AESet-PSO performs better on *information gain* when the volatility is *Med*. When the volatility is low, there is no significant difference. Although the AESet-PSO performs slightly better on *completeness* and the AEStd-PSO performs slightly better on *information gain*, the effect size is *small*, indicating the difference to be negligible.

The convergence time τ_{AM} for AESet-PSO is consistently higher than that of AEStd-PSO. (see figure 5.5 below). This is due to the difference in PSO architecture. In the AEStd-PSO, particles

only have to search along a vector (i.e. single dimension) and fitness is evaluated only once for each particle. The particles, therefore, reach convergence faster. The AESet-PSO, on the other hand, needs to perform a number of set-based operations where fitness is evaluated for each element in the set (particle) and more processing time is expended. Figure 5.5 shows the magnitude of the time difference between AEStd-PSO and AESet-PSO to reach convergence, with both algorithms showing a slight increase in convergence time over the 30 runs. This is due to the increase in the size of the LTM.

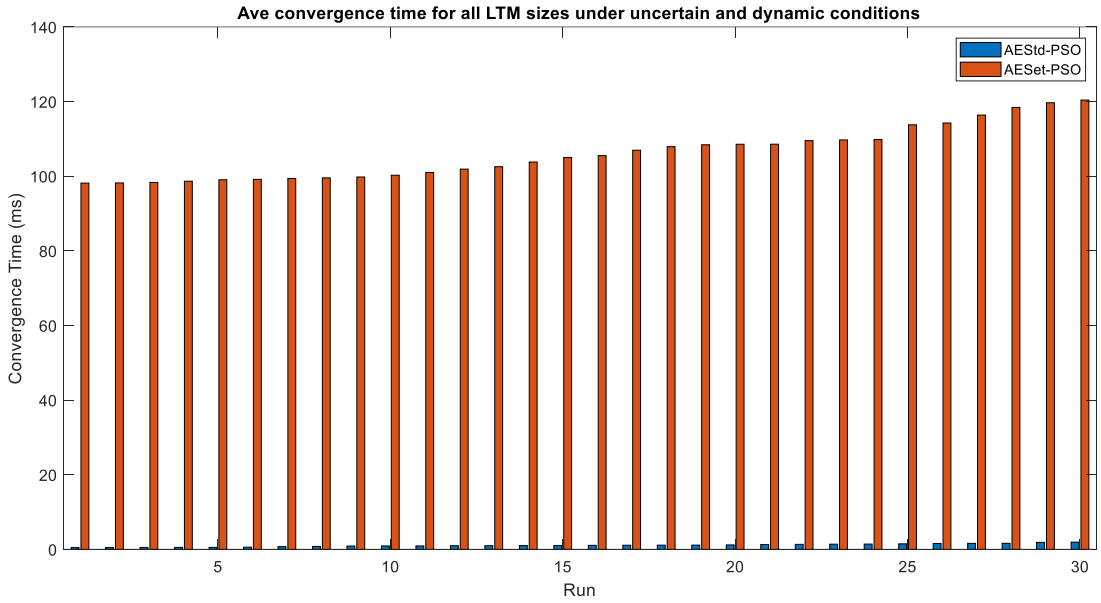

Figure 5.5 Average convergence time of AE-Std-PSO and AESet-PSO.
The graph shows the average convergence time of both the AEStd-PSO and AESet-PSO algorithms.

It is important to note that a short convergence time would be preferable for algebraic optimization problems. However, for optimization problems where the search space consists of discrete and complex items, completeness and information gain of the optimization results are more important. Since the LTM contains discrete and complex memory items, the more important performance measures, such as completeness and information gain. These performance measures are used below to evaluate and select an appropriate algorithm.

The Mann-Whitney U statistic doesn't indicate the magnitude or direction of the difference. Moreover, the difference becomes more important when the search space is volatile. The graphs in figures 5.6 - 5.13 serves to show the effect volatility of the search space has on the completeness and information gain of the two algorithms. The graphs give an indication of the difference between the AEStd-PSO and the AESet-PSO for each statistic (completeness and confidence level), where H_0 is rejected, that is, where there is a significant difference.

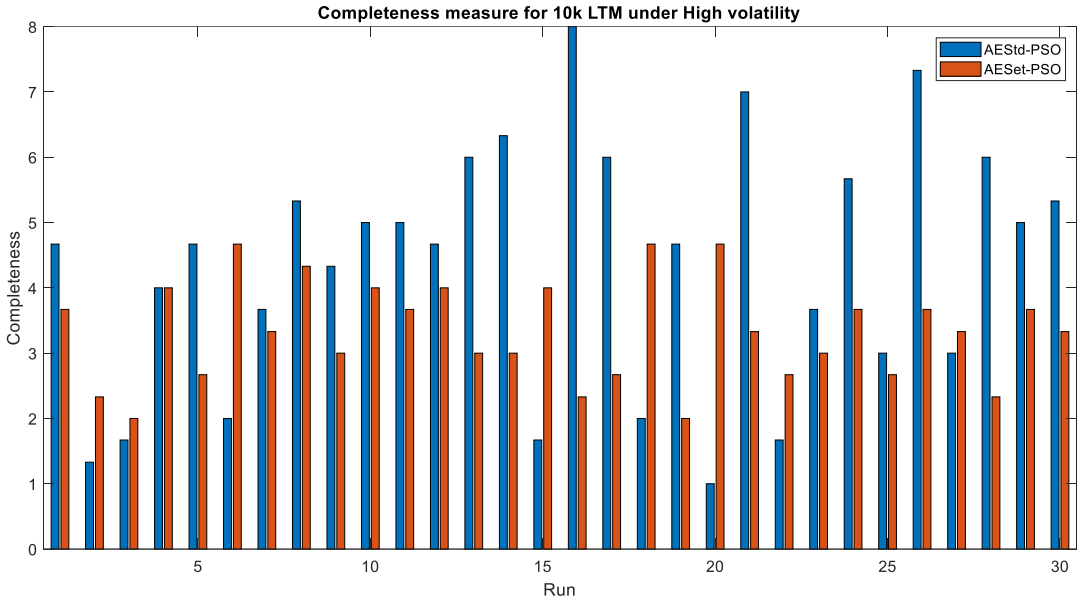

Figure 5.6 Completeness results - small (10k) search space with high volatility.
The results show that volatility has a dramatic effect on the completeness, even when the search space is small. Although the AESet-PSO performs better than the AEStd-PSO algorithm, both algorithms perform poorly under highly volatile conditions.

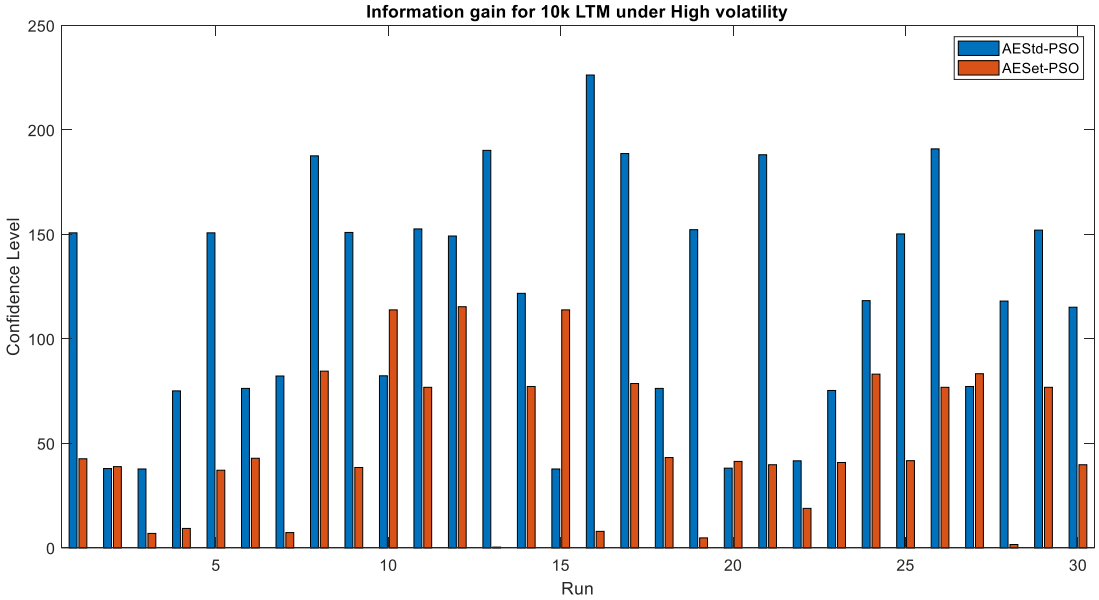

Figure 5.7 Information gain results - 10k search space with high volatility.
The results in this graph show that volatility has a dramatic effect on the confidence level, even when the search space is small. Although the AESet-PSO performs better than the AEStd-PSO algorithm, both algorithms perform poorly under highly volatile conditions.

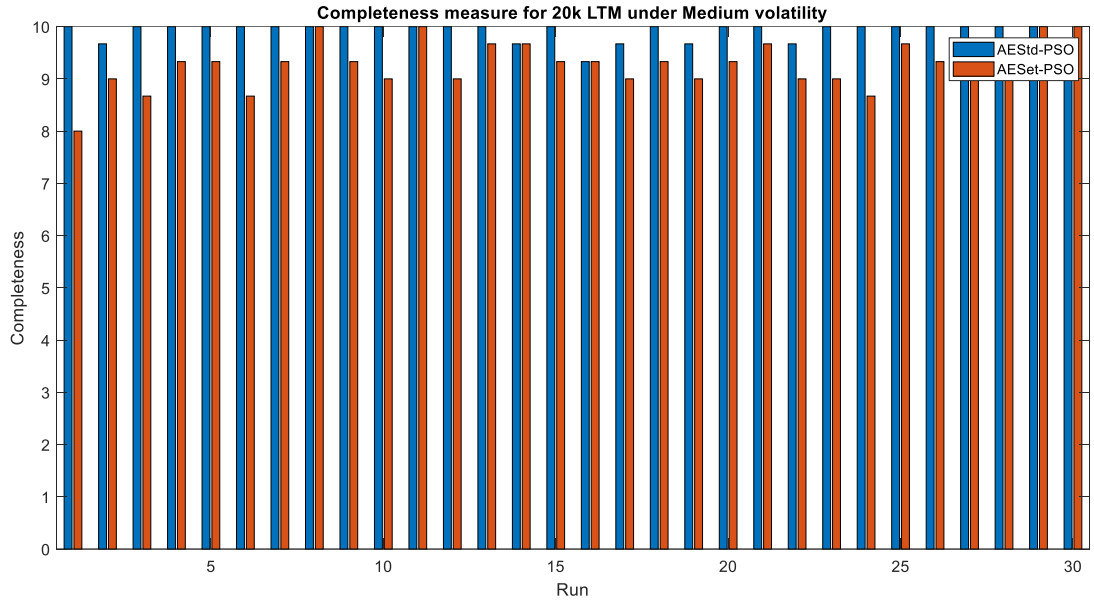

Figure 5.8 Completeness results - medium size search space (20k) with medium volatility. The results show that a reduction in search space size and volatility, significantly improves the completeness of the algorithms, with the AESet-PSO performing marginally better than the AEStd-PSO.

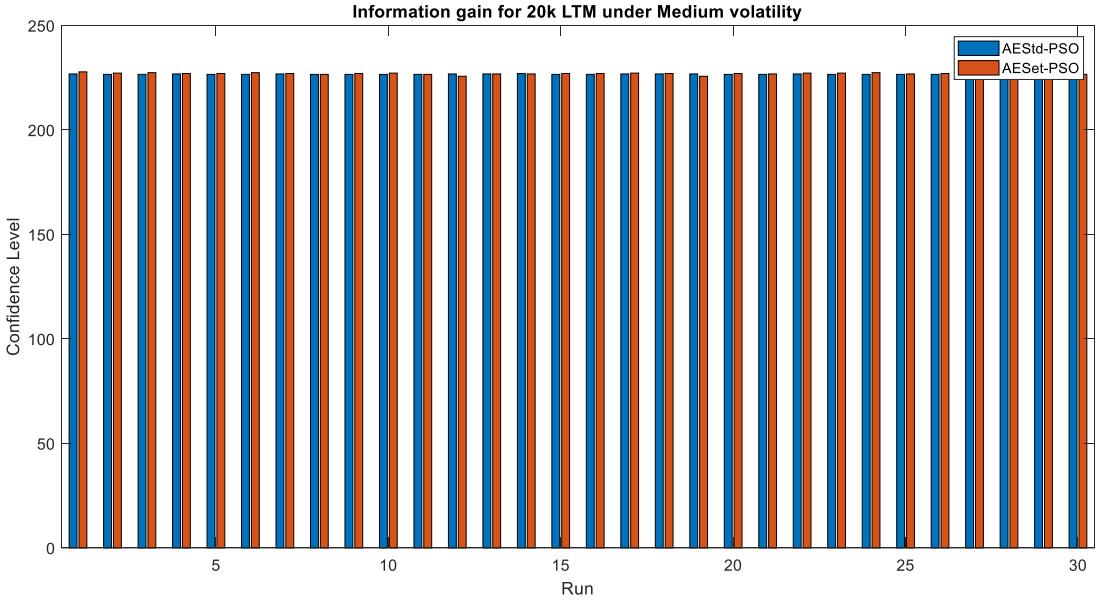

Figure 5.9 Information gain results - medium search space (20k) with medium volatility. The results show that a reduction in search space size and volatility, significantly improves the completeness of the algorithms, with the AESet-PSO performing marginally better than the AEStd-PSO.

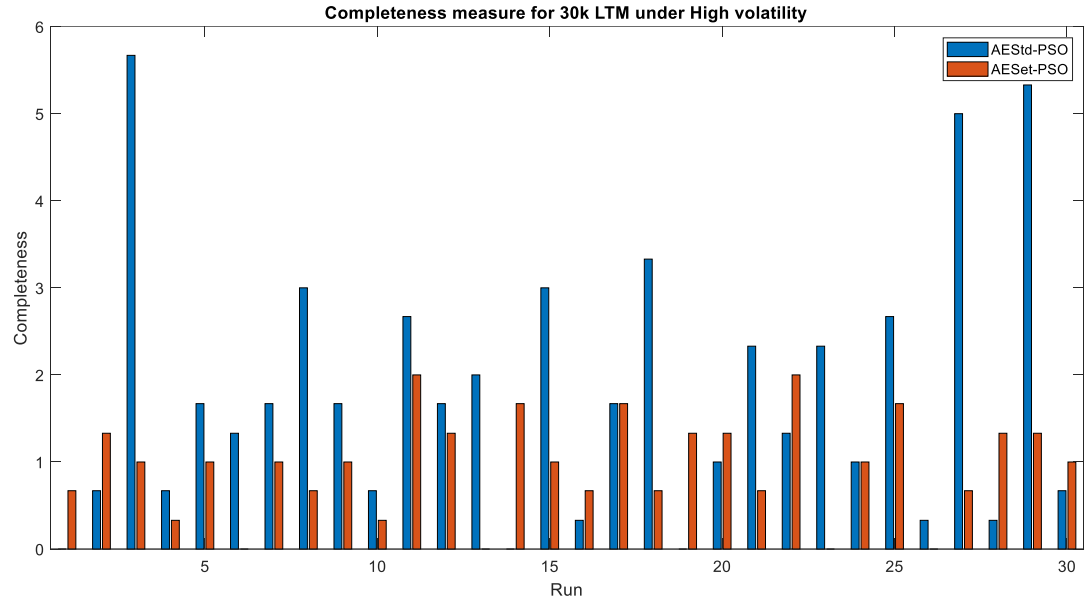

Figure 5.10 Completeness comparison results - large search space (30k) with high volatility.
The results show that, although the AESet-PSO performs slightly better than the AEStd-PSO, both algorithms perform very poorly on a large and volatile search space. when the search space.

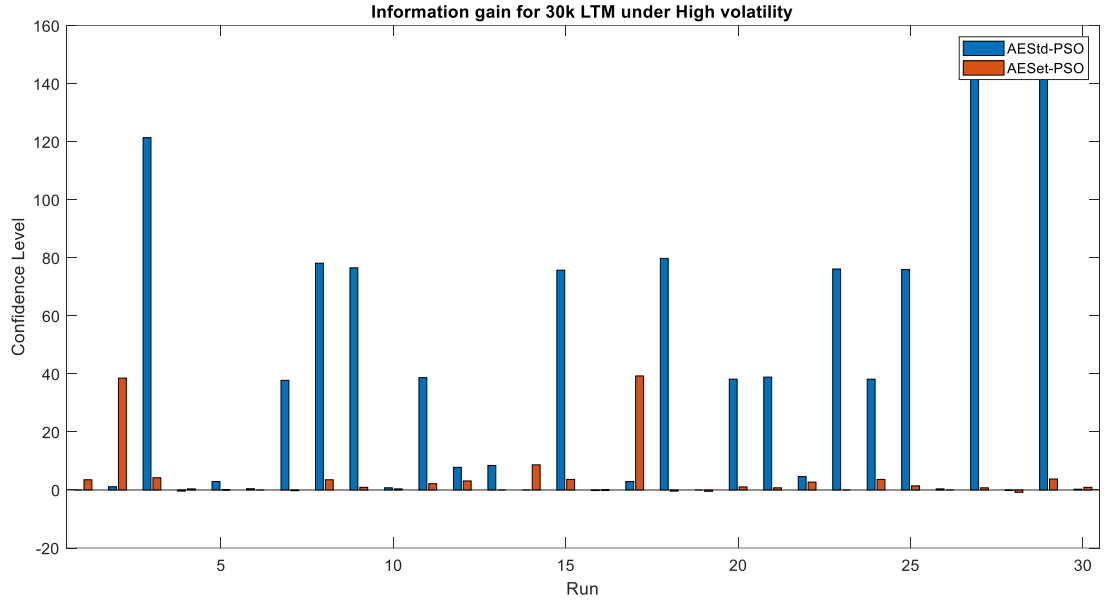

Figure 5.11 Information gain comparison results - of a large search space (30k) with high volatility.
The results show that, although the AESet-PSO performs slightly better than the AEStd-PSO, both algorithms perform very poorly on a large and volatile search space. when the search space.

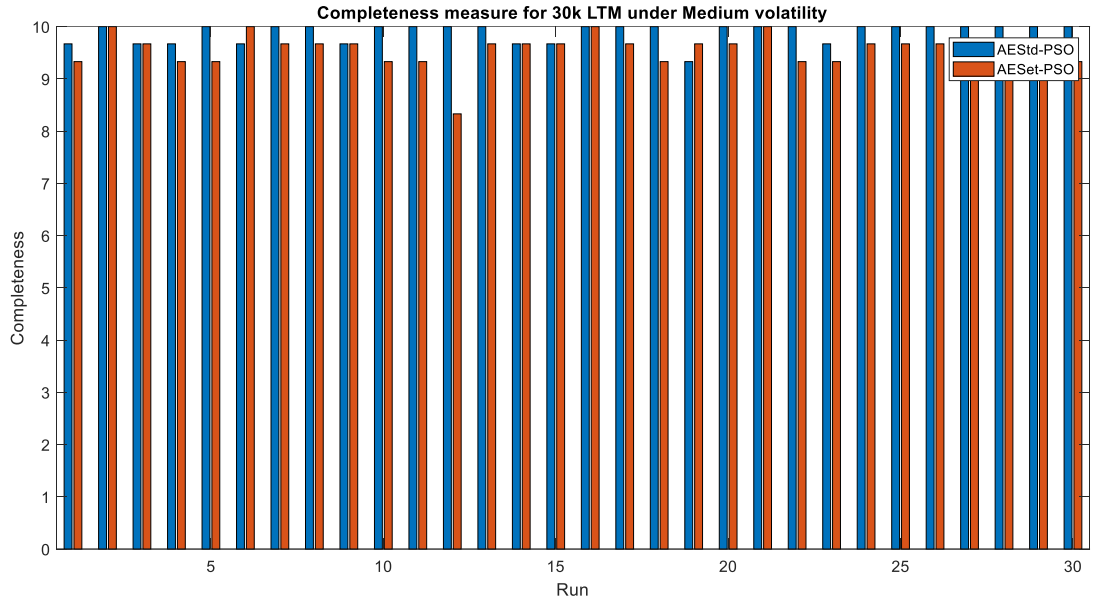

Figure 5.12 Completeness comparison results - of a large search space (30k) with medium volatility.

The results in this graph show that by reducing the volatility on a large search space, the performance is improved, with the AEStd-PSO performing slightly better than the AESet-PSO on completeness.

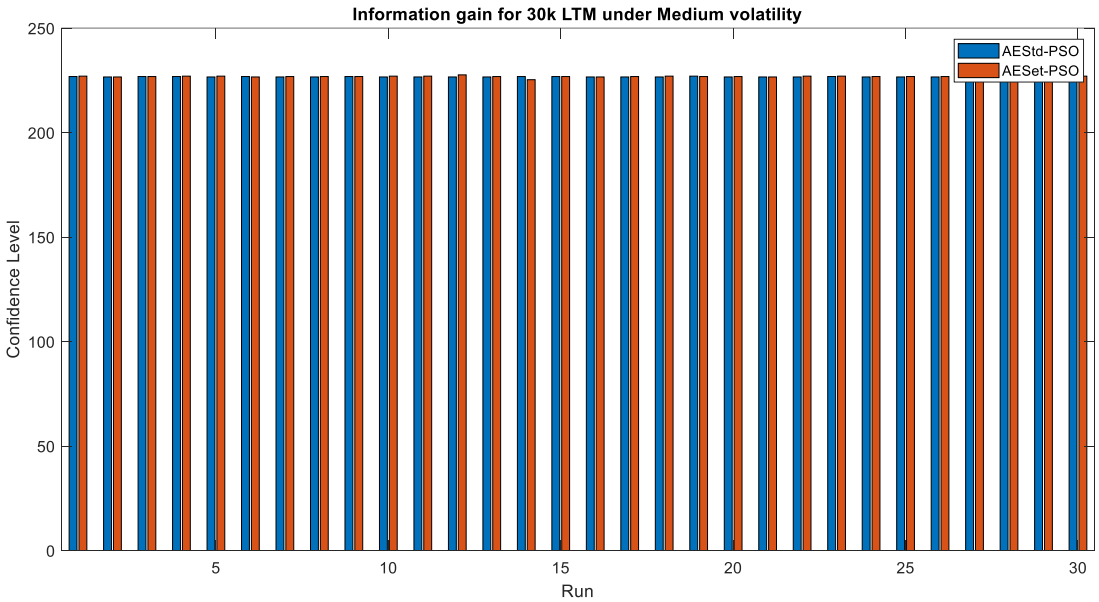

Figure 5.13 Information gain comparison results - of a large search space (30k) with medium volatility.

The results in this graph show that by reducing the volatility on a large search space, the performance is improved, with similar performance between the AEStd-PSO and AESet-PSO algorithms on information gain.

5.2.9 Discussion

There is a significant difference between the two PSO algorithms for the 10k LTM under *High* volatility, for the 20k LTM under *Med* volatility and for the 30k LTM under both *High* and *Med* volatility. The effect size of the difference is *large*. The hypothesis is therefore rejected under these conditions.

When the volatility is *High*, the AEStd-PSO consistently performs better on both *completeness* and *information gain* for all LTM sizes. When the volatility is Med, the AEStd-PSO performs better on *completeness*, while the AESet-PSO performs better on *information gain* for all LTM sizes. When the volatility is Low, there is no significant difference between the PSO algorithms for any of the LTM sizes and effect size is generally *small*.

For benchmark problem 1, the null hypothesis is **not** rejected. There is no significant difference between the AEStd-PSO and AESet-PSO when optimizing an LTM under *uncertain* conditions.

For benchmark problem 2, the null hypothesis is **conditionally** rejected, subject to environmental conditions. Tables 5.6 and 5.7 summarizes the rejection/non-rejection of the null hypothesis, for the simulated environmental conditions.

Table 5.7 Hypothesis rejection for benchmark problem 1.

	Performance measure		
LTM Size	φ_{AM}	ψ_{AM}	τ_{AM}
10k	N	N	***Y***
20k	N	N	***Y***
30k	N	N	***Y***

Table 5.8 Hypothesis rejection for benchmark problem 2.

	Volatility		
LTM Size	***high***	***med***	***low***
10k	***Y***	N	N
20k	N	***Y***	N
30k	***Y***	***Y***	N

The selections are made using sum-of-ranks and effect size in tables 5.4 and 5.5 and the hypothesis rejections, summarized in tables 5.6 and 5.7. Tables 5.8 and 5.9 below, show the preferred PSO algorithm selections. Environmental conditions, for which the null hypothesis is rejected, indicate a significant statistical difference in the performance of the PSO algorithms. The sum-of-ranks indicate the magnitude and direction of this difference, where the magnitude is interpreted as the degree of confidence in the selection. The PSO algorithm with the greatest sum-of-ranks magnitude is selected as the preferred PSO, with a high level of confidence. Environmental conditions, for which the null hypothesis is not rejected, indicate no *significant* statistical difference in the performance of the PSO algorithms. However, the sum-of-ranks may still show a *nominal* difference. The PSO algorithm with the greatest sum-of-ranks magnitude is still selected as the preferred algorithm, albeit with a low level of confidence.

The PSO algorithms in bold italics in table 5.8 indicate the algorithm selected with high degree of confidence for benchmark problem 1. PSO algorithms not in bold indicate the algorithm selected with a low degree of confidence.

Table 5.9 PSO algorithm preference for benchmark problem 1		
LTMSize	**performance measure**	**preferred**
10k	φ_{AM}	either
	ψ_{AM}	either
	τ_{AM}	***AEStd-PSO***
20k	φ_{AM}	AESet-PSO
	ψ_{AM}	AEStd-PSO
	τ_{AM}	***AEStd-PSO***
30k	φ_{AM}	AESet-PSO
	ψ_{AM}	AEStd-PSO
	τ_{AM}	***AEStd-PSO***

The PSO algorithms in bold italics in table 5.9 indicate the algorithm selected with high degree of confidence for benchmark problem 2. PSO algorithms not in bold indicate the algorithm selected with a low degree of confidence.

Table 5.10 PSO algorithm preference for benchmark problem 2				
		Volatility		
LTMSize	**performance measure**	***high***	***med***	***low***
10k	φ_{AM}	***AEStd-PSO***	AEStd-PSO	AESet-PSO
	ψ_{AM}	***AEStd-PSO***	AESet-PSO	AEStd-PSO
	τ_{AM}	***AEStd-PSO***	AEStd-PSO	AEStd-PSO
20k	φ_{AM}	AEStd-PSO	***AEStd-PSO***	AEStd-PSO
	ψ_{AM}	AEStd-PSO	***AESet-PSO***	AESet-PSO
	τ_{AM}	AEStd-PSO	***AEStd-PSO***	AEStd-PSO
30k	φ_{AM}	***AEStd-PSO***	***AEStd-PSO***	AESet-PSO
	ψ_{AM}	***AEStd-PSO***	***AESet-PSO***	AEStd-PSO
	τ_{AM}	***AEStd-PSO***	***AEStd-PSO***	AEStd-PSO

The statistical analysis shows that both PSO algorithms are capable of optimizing a LTM, given temporal environmental data. However, the performance, in terms of *completeness*, *information*

gain and *convergence time* of the PSO algorithm is influenced by the environmental conditions. When selecting a preferred PSO algorithm, the statistical analysis of the performance of each algorithm is used. Tables 5.8 and 5.9 shows the preferred PSO algorithms, based on the statistical results, given the various environmental conditions. The PSO algorithms shown in bold italics, are selected with "*high*" confidence, because there is a significant statistical difference and the null hypothesis is rejected. The PSO algorithms not in bold, are selected with "*low*" confidence, because there is only a nominal difference and the hypothesis is not rejected. When the PSO algorithm is selected with "low" confidence, the effect size is small to medium. The selection is then made subjectively, as the performance of both algorithms is similar.

5.3 Conclusion

The significant difference in the *convergence time* of the two PSO algorithms, for all environmental conditions, is important. If reaction time is a priority for an autonomous system, the lag in convergence time for the AESet-PSO may be prohibitive and the AEStd-PSO is preferred. On the other hand, since *completeness* and *information gain* is more important, the AESet-PSO is preferred.

It is important to note that the level of performance of the AEStd-PSO, as indicated in the statistical analysis, can only be achieved if the LTM is ordered prior to optimization, as stated previously. Then, all the relevant memory items are in close proximity and the density of the converging swarm in the area is able to find all memory items. The AESet-PSO does not have this requirement. However, if the environment is dynamic and diverse, it cannot be guaranteed that the LTM will be ordered. It is therefore concluded that the set-based PSO will be more suitable for the optimization of discrete memory elements in the LTM, since it performed better overall on completeness and confidence level.

Chapter 6

Robo-cognitive architectures

In this chapter, memory representation and quantification, developed in chapter 4, are combined with memory optimization, in the cognitive function of memory recall. In section 6.1, an architecture for real-time, cognitive control using SPSO for single-task execution, is developed and section 6.2, an architecture for real-time, cognitive control using CG-PSO for multi-task execution, is developed. The methodology, simulations and performance analysis for both architectures are presented in detail in this chapter. The performance of both architectures is evaluated using a UAV simulation environment.

Memory representation and quantification

In section 4.1, figures 4.1 and 4.2 represent state flows which represents the valid states and state transitions of two UAV functions, flight controls and gripper controls. These state flow diagrams are provided as a visual reference to the reader, but is implemented in the LTM, shown in figure 4.3. The memory represented as the LTM, is used in the memory quantification, developed in chapter 4. The quantification is then used by both the SPSO and CG-PSO algorithms during memory recall (memory optimization).

6.1 Real-time Episodic Memory Construction in Cognitive Control of Autonomous Vehicles

The architecture developed in this section is based on the Baddeley model for working memory and uses the set-based PSO to construct the episodic memory. The episodic memory represents the optimal set of memory items, i.e. state transitions, from which the CE selects and executes the actions, defined by the memory item (state transition).

6.1.1 Methodology

During memory recall, the CE uses real-time environmental stimuli and cues to statistically quantify and recall memory items from LTM. Memory optimization during memory recall is performed by the SPSO algorithm, and memory item fitness quantification is performed by the adaptive entropy fitness quantification (AEFQ) algorithm. The robo-cognitive architecture and main functions are shown in figure 6.1. To the best of our knowledge, there has been no attempt

to use set-based PSO for real-time optimization of working memory in any robo-cognitive architecture.

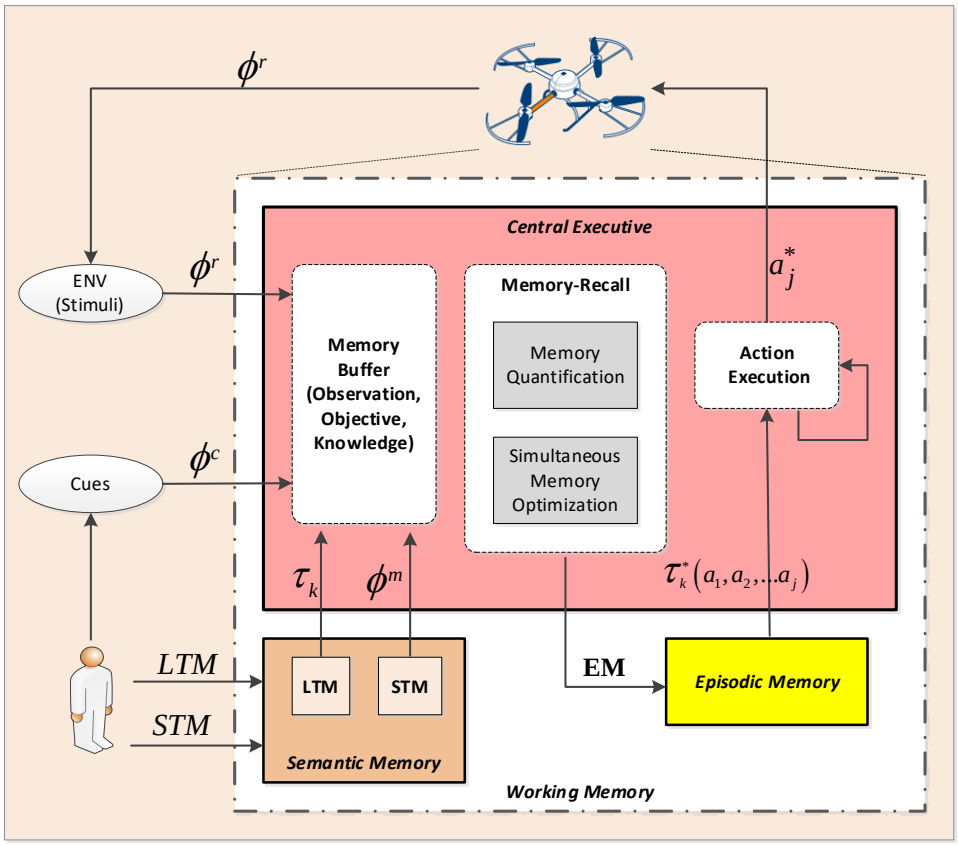

Figure 6.1 A robo-cognitive architecture, using on Baddeley's model of working memory. Episodic memory is constructed during memory recall. The episodic memory is used by the central executive to select the optimal memory element for action selection and execution.

In this approach, the domain expert provides cues (or missions) which defines the objectives of the autonomous vehicle. The central executive recalls, quantifies and optimizes semantic memory in real-time, subject to the cues and stimuli. Since the process is dynamic and in real-time, the optimal memory constructed by the central executive is episodic, and used for selecting and executing the optimal action. Memory optimization is done using the SPSO algorithm. The result of the memory optimization, is the episodic memory, from which the CE selects the optimal memory item. Finally, the action defined by the selected episodic memory item is executed.

In this approach, episodic construction, using the AEFQ algorithm and set-based particle swarm optimization (SPSO) algorithm, is used for real-time memory recall for high-level, single task autonomous vehicle control. Memory representation and quantification, is used in the memory recall cognitive process.

The AEFQ algorithm employs the MEP to provide a probability distribution over all the characteristics of the semantic memory item, for fitness evaluation. In this approach, the episodic memory (see Baddeley's memory model, figure 3.2) represents the optimal set of memory items from which the executive uses the probability distribution of each item to select the best memory item and execute a suitable action. The performance of the approach is evaluated by simulation with two unmanned aerial vehicle (UAV) use cases: 1) flying to a charging station for re-charging and 2) delivering a medical package, before flying to a charging station.

6.1.2 Reasoning in the robo-cognitive architecture

Reasoning in the robo-cognitive architecture is a cognitive process performed by the CE. In figure 6.1, it is shown that the CE is composed of two main functions: cognition and reasoning. The cognition component is tasked with the statistical optimization of the knowledge from the LTM, given the environmental data. The reasoning component is tasked with selecting the optimal action π^*, from the episodic memory, EM.

The CRP uses the optimal set of solutions (state transition) found by the AE-SPSO to select and execute the relevant actions.

Algorithm 6.1 Cognitive reasoning process (CRP) algorithm

```
Input     : LTM    -- Domain expert knowledge
          : Mission parameters, Φ^m with tasks, φ_j^m
          : Runtime parameters Φ^r with evidence , φ_i^r
Begin
    -- Execute the mission
    For each task, φ_j^m ∈ Φ^m
        Repeat
            Input all φ_i^r ∈ Φ^r from sensory input
            -- Call the AE_SPSO algorithm to find the optimal solutions
            EM = AE_SPSO(LTM, Φ^m, Φ^r)        -- see (Algorithm 5.2)
            -- Select and execute action/s from optimal solution/s
            For each action τ_k ∈ EM
                For each action 𝒜 ∈ τ_k
                    For each action a_n ∈ 𝒜
                        Execute action a_n
                    Next action
            Next state transition
        Until Task completed
    Next Task
End
```

The robo-cognitive architecture developed in this section enables action selection and execution from episodic by the central executive. Each task of the mission is executed sequentially, task-by-task. The next section discusses the development of a robo-cognitive architecture for multi-task execution, based on a coalitional game-theoretic approach.

6.2 Real-time Activated memory Construction for Cognitive Control of Autonomous Vehicles

This section introduces a coalitional game theory-based PSO (CG-PSO) algorithm, based on a combination of PSO and coalitional games theory. During memory recall, the CE follows Cowan's attentional focus memory model (see figure 3.3), where the CG-PSO produces an optimal AM, from which multiple FOAs can be selected for action selection and execution.

The performance of the CG-PSO algorithm is evaluated by simulation, with two unmanned aerial vehicle (UAV) use cases: delivering medical equipment to an incident, and flying a security surveillance support mission.

6.2.1 Methodology

During memory recall, the CE uses real-time environmental stimuli and cues to statistically quantify and recall memory items from LTM. Memory optimization during memory recall is performed by the CG-PSO algorithm, and memory item fitness quantification is performed by the adaptive entropy fitness quantification (AEFQ) algorithm. The robo-cognitive architecture for multi-task execution and its main functions are shown in figure 6.2. To the best of our knowledge, there has been no attempt to combine cooperative game theory and PSO for real-time optimization of working memory in any robo-cognitive architecture.

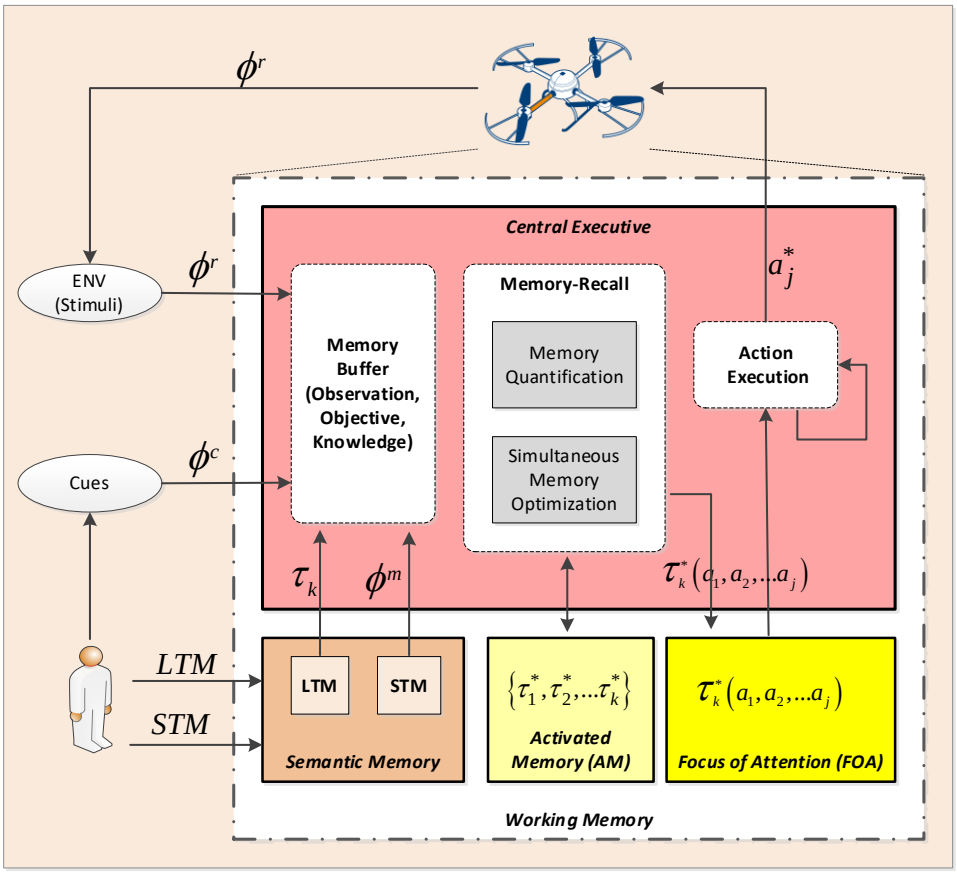

Figure 6.2 A robo-cognitive architecture, using Cowan's attentional focus theory of working memory.
Activated memory and focus of attention is constructed during memory recall. From the focus of attention, actions are selected for execution by the central executive.

To assist in the description, the architecture is described using a UAV with two functions: Flight control and Gripper Control. The robo-cognitive architecture, illustrated in figure 6.2, shows the structure and components of the working memory, specifically:

- Central Executive (CE) – the cognitive process, governing the cognitive components in reasoning and decision-making,
- Long-term memory (LTM),
- Short-term memory (STM),
- Environmental stimulus (ENV),
- Cues represent a mission, composed of a collection of tasks, defined by the domain expert,
- Cue represents a specific task of the mission,
- AM represents the Activated Memory,
- FOA represents the Focus of Attention,

- Cognitive cycle (C) – a period of memory recall, action selection and action execution.

6.2.2 Reasoning in the robo-cognitive architecture

Reasoning in robo-cognitive architecture is a cognitive process, implemented as a coalitional game played by the particles in a swarm. The process governs the construction of AM, from which the FOA is identified for action selection and execution by the CE.

Formally, the coalitional game (with transferable utility) is defined as follows:

When the worth of the coalition can be distributed amongst its members, the game is called a transferable utility (TU-Game). A coalitional game with transferable utility is defined as follows:

Definition 6.1: A coalitional game is a pair $(N; v)$ such that:

- $N = \{1,2, \dots, n\}$ is a finite set of players. A subset of N is a coalition S and the collection of all coalitions is denoted by 2^N.
- $v: 2^N \rightarrow \mathbb{R}$ is a function associating each coalition S with a real number $v(S)$, satisfying $v(\emptyset) = 0$. This function is also called the *characteristic function* of the game and $v(S)$ is the social welfare of the coalition.

Contrary to real-world practice, in this study, coalitions may consist of a single player. For ease of computation, every particle in a game will initially be in a coalition by him- or herself.

The maximum amount a coalition S can generate through the cooperation of its members is the *social welfare* or *social utility* $v(S)$, of the coalition. The coalition's social welfare is distributed amongst its members. The amount of utility a member x receives is referred to as the *individual welfare* or *individual utility* $v(x)$ the member receives from the coalition; it chooses to join. A user-defined payoff function (see definition 3.1), calculates the utility a player will *potentially* receive, when forming a coalition with another player. A player cannot receive a higher payoff than the worth of the coalition.

In this study, the individual utility $v(x)$, is assigned by the AEFQ algorithm.

Definition 6.2: Let $\mathcal{U}$ be a family of coalitional games. A solution concept over $\mathcal{U}$ is a function φ associating every game $(N; v) \in \mathcal{U}$ with a subset $\varphi(N;\ v)\ of\ \mathbb{R}^N$.

A *single-valued solution concept*, also called a *point solution*, is a function which assigns to each coalitional game, a payoff vector in $\mathbb{R}^N$, indicating the individual welfare of each player in the game. This function is performed by an arbitrator (i.e. the designer), which decides how

to divide the social welfare amongst the players. A solution concept specifies the payoff each member receives in a game and defines the players in terms of the coalition structures they form and corresponding payoff of both coalitions and players.

Figure 6.3 shows an example of a coalition structure. In a game, memory items (see eq. 4.3) selected from the LTM, during memory recall, are defined as quantified assets. The assets "owned" by the player, determine the player's utility (or worth) and is used during the bargaining process. Given a swarm $\mathbb{S}$ of N particles, the objective of the swarm is to maximize its collective (social) welfare by cooperatively accumulating the best assets. To achieve this, all the particles engage in a coalitional game, $(N; v)$, possibly resulting in a coalition S, with social utility $v(S)$, for each function of the problem.

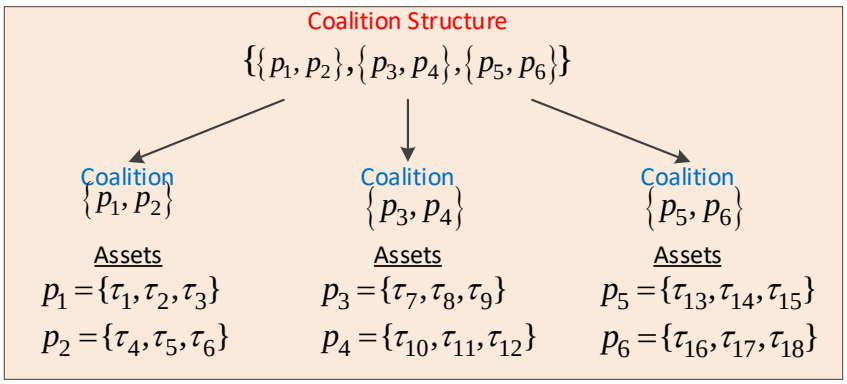

Figure 6.3 An example of a coalition structure.
In the figure, a coalition structure, composed of three coalitions, each with two members, is shown. In the diagram, p = particle (member) and τ = asset (state transition).

The example shows a coalition structure with 3 coalitions: $S_1 = \{p_1, p_2\}$, $S_2 = \{p_3, p_4\}$ and $S_3 = \{p_5, p_6\}$. formed by a swarm of 6 particles, each with 3 assets. An asset represents a memory item, selected from the LTM, during memory recall. The coalition structure represents the AM for a specific function of the problem and from which the FOA (optimal asset) is selected for action execution.

Coalition structure formation

During the cognitive cycle, the CG-PSO constructs an imputation (see definition 6.6), by firstly constructing the coalition structure $\mathcal{B}$, using algorithm 6.2. The coalition structure contains coalitions of particles, where the individual utility (fitness), $f(p)$, of each particle, is determined by the AEFQ algorithm. Secondly, the payoff vector of the coalition structure is derived through the defection process (algorithm 6.3), where particles remain in a coalition or defect to another (more profitable) coalition.

Definition 6.3: A coalition structure $\mathcal{B}$ is a collection of disjoint sets, where each set is a coalition $S \in \mathcal{B}$ of players. The coalition structure $\mathcal{B}$ is therefore a partition of the set of N players.

The solution concept $\varphi(N; v; \mathcal{B})$ represents a set solution concept for the coalition structure $\mathcal{B}$. Associated with $\mathcal{B}$, is a set of payoff vectors, where each payoff vector corresponds to a coalition $S \in \mathcal{B}$. The rationality of the payoff distribution is important, as it influences the decision of a player to form a coalition or defect from a coalition.

Definition 6.4: A payoff vector is *socially rational* if $x(S) \geq v(S)$, that is the *total* social welfare of the coalition S is divided amongst its players.

In the proposed methodology, there is no a-priori "budget" available for distribution amongst the members. The social utility of a coalition, defined in definition 6.1, is calculated as the sum of the individual utility of its members and each member is "awarded" only the utility he/she contributed. Therefore, in the robo-cognitive architecture, coalitions are always socially rational, as each member receives at least what he contributed and there is no "unallocated" utility.

Definition 6.5: A payoff vector is *individually rational* if $x_i \geq v(i)$. Since every player can guarantee at least his current individual welfare $v(i)$ if he/she doesn't join, it is reasonable to assume he/she will demand at least this amount when joining the coalition. (Also see comment – definition 3.2)

In the proposed methodology, this definition is ignored for performance reasons. When a particle defects from a coalition it may transfer some of its assets before defecting, thereby reducing its individual utility. However, since the particle joins a coalition, relevant to the category of its assets, it may be argued that the particle will receive a bigger payoff because its assets are worth more to the coalition it joins.

When a payoff vector is socially and individually rational, it is called an *imputation* of the coalition and is defined in definition 6.6 below.

Definition 6.6: Let $(N; v)$ be a coalitional game, and let $\mathcal{B}$ be a coalitional structure. An imputation x, for the coalitional structure $\mathcal{B}$ is a vector $x \in \mathbb{R}^N$ which is both socially and individually rational for $\mathcal{B}$. The set of all imputations for $\mathcal{B}$ is denoted by $X(\mathcal{B}; v)$:

$$X(\mathcal{B}, v) \coloneqq \{x \in \mathbb{R}^N : x(S) = v(S) \quad \forall S \in \mathcal{B}, \quad x_i \geq v(i)\ \forall i \in N\}$$

where $x(S)$ is a vector of payoffs to all players in coalition S and $v(S)$ is the total value of the coalition.

An imputation, $x \in X(\mathcal{B}, v)$ is denoted by,

$$x = (x_1, x_2, \ldots, x_N) \tag{6.1}$$

where x_i ; $i \in [1, N]$ represent the payoff player i receives.

The set of imputations forms the basis for rational bargaining, amongst the particles in the swarm. The bargaining set, is a coalitional solution concept and is the set of all imputations in $X(\mathcal{B}, \upsilon)$ at which every objection of one player against another player in the same coalition in the coalitional structure $\mathcal{B}$, is met by a counter objection. In other words, it is the set of all imputations against which unjustified objections are raised and forms the basis for negotiation amongst the players. Bargaining may be described as an iterative, negotiation-objection-counter objection, process. However, in this study, the performance of the cognitive process is important for the high-level control of a UAV. Therefore, bargaining will be limited to negotiation-objection process only, where there is no counter-objection raised by the initiating player. This prevents intractable and expensive recursive negotiations between particles.

Definition 6.7: Let $(x, \mathcal{B})$ be an imputation, and let $k, l (k \neq l)$ be two players belonging to different coalitions in $\mathcal{B}$. An ***objection*** of player k against player l at x is a pair (C, y) such that:

1. $C \subseteq N$ is an alternative coalition containing k but not l: $k \in C, l \notin C$.
2. $y \in \mathbb{R}^{|C|}$ is a vector of real numbers satisfying $y(C) = v(C)$, and $y_i > x_i$ for each player $i \in C$.

where y is an alternative imputation (see definition 6.6), $y(C) = v(C)$ indicates the social rationality (see definition 6.4) of the alternative coalition C and $y_i > x_i$ indicates the individual rationality (see definition 6.5) of all the players in the alternative coalition. The alternative coalition is the coalition that will be formed if k decides to join.

This definition states: player k raises an objection to player l's offer since he/she can potentially receive a larger payoff (y_i), when he/she joins an alternative coalition C, than he/she would have received (x_i), if joining l's coalition. The objection is a result of the rational behaviour of player k in maximizing his utility.

Definition 6.8*:* Let (C, y) be an objection of player k against player l at x. The objection is ***justified*** if player l has no counter objection to it.

As an example of the bargaining process, consider a swarm of four particles: A, B, C and D. Particle A requests particle B to join its coalition. Particle B checks with particles C and D, to establish whether he/she can do better (rational behaviour) if he/she forms/joins a coalition with one of them. If he/she will be better off joining either C or D, he/she will object to particles A's offer. Particle A then approaches particle C with the same request and the process is repeated.

How the particles join coalitions are random, based on the potential payoff the candidate particle will receive, i.e. the offer of the recruiting particle.

Let particle i be the recruiting particle, and particle j be the candidate particle. The potential payoff particle i offers particle j, is calculated as,

$$offer_{ij} = v(S_{p_i}) + v(p_j) \tag{6.2}$$

where $offer_{ij}$ represents the payoff "promised" to particle j if he/she joins particle i's coalition. The utility $v(p_j)$ of particle j is calculated by summing the total fitness of all assets (memory items) owned by particle j. The fitness of each asset is quantified using the AEFQ algorithm. The social utility of particle i's coalition is represented by $v(S_{p_i})$ and is the sum of the utility (fitness) of all particles in the same coalition as particle i. The utility, $offer_{ij}$, which is offered to particle j, is the social utility of particle's i's coalition plus the utility particle j will contribute to the coalition if it joins the coalition.

Given definitions 6.3 – 6.8, the bargaining set can now be defined.

Definition 6.9: Let $(N; v)$ be a coalitional game, and let $\mathcal{B}$ be a coalitional structure. The *bargaining set* relative to the coalitional structure $\mathcal{B}$, is the set $\mathcal{M}(N; v; \mathcal{B})$ of imputations in $X(\mathcal{B}, v)$ at which no player has a justified objection against any other player in the same coalition.

From the bargaining set, a point solution concept [119] can now be defined as,

$$\varphi = (x;\ S_1, S_2, \ldots, S_{n_{\mathcal{F}}}) \tag{6.3}$$

where x is the imputation (definition 6.6) and $S_j\ ;\ j \in [1, n_{\mathcal{F}}]$ represent the jth coalition in the coalition structure $\mathcal{B}$, representing a function in $\mathcal{F}$.

Each iteration of the CG-PSO is a cognitive cycle, during which bargaining takes place. Each particle in the swarm, in turn, bargains with (or requests) every other particle in the swarm to join its coalition. During the cognitive cycle, the decision of the particle to join or not join, is made based the rationality of the particle. The particle's individual utility is determined by the worth (i.e. fitness) of the quantified assets (i.e. memory items) it owns. The particle's rationality is driven by the offer (eq. 6.2), which is influenced by the particle's individual utility and the coalition's social utility.

The bargaining process for the construction of the coalition structure of the imputation performed by the CG-PSO algorithm, shown in Algorithm 6.2.

Algorithm 6.2 Coalitional game-theoretic PSO (CG-PSO) algorithm

```
Input : Cues, LTM, ENV, STM
Output: coalition structure, 𝓑

Begin
    Initialize a swarm, 𝕊 of N particles, each particle contains n
        randomly selected state transitions, τ_k ∈ LTM.

    Initialize coalition structure, 𝓑 with each particle in its own coalition:
        for each particle p_i ∈ 𝕊
            set S_i = {p_i}
            add S_i to 𝓑
        endfor
    -- Start negotiation (bargaining) cycle
    repeat
        for each particle i ∈ 𝕊
            for each particle j ∈ 𝕊 where (j ≠ i)
                -- particle i negotiate with particle j to form {i, j}
                -- calculate an offer from particle i to particle j using eq. (6.2)
                calculate offer_ij using (6.2)
                for each particle k ∈ 𝕊 where (k ≠ j) and (k ≠ i)
                    -- particle j negotiates with particle k to form {j, k}
                    -- calculate an offer from particle j to particle k
                    -- using eq. (6.2)
                    calculate offer_jk
                    if offer_jk > offer_ij
                        particle j objects to particle i's offer   -- see def. 6.7
                    else
                        particle j defects from S_j      -- using algorithm 6.3
                        particle j joins S_i
                    endif
                endfor
            endfor
        endfor
      until end condition
      return 𝓑
end of working memory optimization
```

The bargaining process causes coalitions of particles to form in a way which maximizes the social utility of the coalition. This is similar to the behaviour of the swarm in PSO, where particles converge on the global best solution.

In order to keep the response time for memory recall as low as possible, bargaining is limited to objections only. However, for problems without strict performance constraints, the negotiation-objection-counter objection may prove useful for coalition-formation, based on more complex negotiations and formation rules. For completeness of the role of the bargaining set in coalitional games theory, the definition of a counter-objection is given below.

Definition 6.10: Let (C, y) be an objection of player k against player l at payoff x_i of the payoff vector. A ***counter objection*** of player l against player k is a pair (D, z) satisfying:

1. D is a coalition where $l \in D$ and $k \notin D$.
2. $z \in \mathbb{R}^D$ and $z(D) = v(D)$.
3. $z_i \geq x_i$, for every player $i \in D \backslash C$.
4. $z_i \geq y_i$, for every player $i \in D \cap C$.

This definition states: a counter objection is raised by player l if he/she can find another coalition *D* of which he/she (but not k) is a member and the worth of coalition *D* is divided in such a way that each member of $D \backslash C$ receives at least what he/she receives under x, and each member of $D \cap C$ receives at least what he/she receives under y (offered by k in his objection to x).

The solution concept can now be completed by constructing the payoff vector (imputation).

Payoff vector construction

To assist in the construction of the payoff vector x, each coalition is categorized according to the assets possessed by its members. Each asset in the search space belongs to a specific function f_j and therefore coalitions will be formed which fully represent a single function, $f_j \in \mathcal{F}$, (referred to as P-coalition, indicating it is a "pure" coalition) or a mix of functions (referred to as a D-coalition, indicating members will defect in to join another P-coalition.

A coalition whose members possess assets only from a single function f_1 is a P-coalition and is assigned type A, a coalition whose members only possess assets only from function f_2 is also a P-coalition and is assigned type B and so on. A coalition whose members possess assets from various functions, is a *D*-coalition and is assigned a unique type D (reserved for these types of coalitions). (Category codes used are arbitrary and the decision of the designer).

Let,

$$\omega_{S_j} = \begin{cases} 1 & ; if\ coalition\ S_j\ is\ not\ a\ D\_coalition \\ 0 & ; if\ coalition\ S_j\ is\ D\_coalition \end{cases} \tag{6.4}$$

be a payoff weight assigned to a coalition S_j, based on the assets of its members. The social utility of coalition S_j is then defined as,

$$v(S_j) = \omega_{S_j} * \sum_{p_i \in S_j} v(p_i);\ i = 1,..N \tag{6.5}$$

which is shared in full with each member particle:

$$x_{p_i} = v(S_j)\ ;\ p_i \in S_j \tag{6.6}$$

The payoff weight controls the behaviour of particle's movement towards maximizing the social utility. A positive payoff by the coalition encourages particles to remain in the coalition, while particles who receive no payoff is incentivised to defect and join a more profitable coalition.

The payoff vector (imputation) x, for coalition structure $\mathcal{B}$, is constructed:

$$x = (x_{p_1}, x_{p_2}, \dots, x_{p_i});\ x_{p_i} \in S_j \text{ and } S_j \in \mathcal{B} \tag{6.7}$$

The coalition structure $\mathcal{B}$ represents the AM for a function. The FOA can only be selected from AM represented by P-coalitions. Therefore, for any D-coalitions, a negotiation must take place between its members and other relevant P-coalitions. The objective is for particles in a D-coalition to defect to other, more "profitable" coalitions. Algorithm 6.3 shows the defection process:

Algorithm 6.3 Particle defection process

```
1:  begin
2:      given coalition structure ℬ ∈ φ
3:          For each D-Coalition ≠ ∅
4:              for each particle in type D-Coalition
5:                  Separate assets into function-type sets
6:                  Retain function-type set with the largest number of assets.
7:                  Transfer all other assets to particles in relevant coalitions
8:                  Defect from D-Coalition
9:                  Join relevant P-coalition according to retained assets
10:             endfor
11:         endfor
12: end defection
```

Finally, after defections, the point solution concept φ is complete and each coalition $S_j \in \varphi$ represents the AM for a specific function $f_j \in \mathcal{F}$.

6.2.1.3 Action selection and execution

The cognitive process of the robo-cognitive architecture concludes with action select and execution by the CE. With the solution concept (containing the coalition structure) constructed, the FOA can be set and the CE selects the action/s from $A_k = \{a_1, \ldots, a_{n_{A_k}}\}$, defined by the asset. The relevant command/s is/are derived from the selected action/s and sent to the UAV controller for execution.

The complete cognitive reasoning process of the robo-cognitive architecture is shown in Algorithm 6.4.

Algorithm 6.4 Cognitive reasoning process (CRP)

Initialize:
    Cues, Φ^c -- see eq. (4.1)
    Long Term memory, LTM -- see eq. (4.2) and (4.3)
    Short term memory, STM -- see eq. (4.5)
    Environmental info, ENV -- see eq. (4.4)
    Activate memory, AM = {}

begin -- reasoning process, given a cue from Cues
    for each *cue* in Cues
        Input environment stimuli, ENV
        Given the cue and ENV, retrieve related STM

        -- Construct the coalition structure $\mathcal{B}$ of φ
        $\mathcal{B}_x = CG\_PSO(CUE, \text{LTM}, \text{ENV}, \text{STM})$ using Algorithm 6.2

        -- Construct the payoff vector of φ
        for each $S_j \in \mathcal{B}_x$
            Categorize coalition S_j as P-coalition or D-coalition
            Calculate payoff $v(S_j)$ using eq. (6.5)
        Endfor

        -- Assign utility to each member of $\mathcal{B}_x$
        for each $p_i \in S_j$ and $S_j \in \mathcal{B}_x$
            Assign individual utility using eq. (6.6)
        endfor

        -- Process defections and complete coalition structure construction
        Process defections from D-coalition using Algorithm 6.3
        -- Calculate payoff vector (imputations) (see eq. (6.7))
        AM = $Imputation\ x$

31.
32. -- Construct Focus of Attention set
33. for each $\tau_k^* \in$ AM
34. if $\tau_k^* = cue.objective$
35. Add τ_k^* to FOA
36. endif
37. endfor
38. endfor
39.
40. -- Action selection and execution
41. for each $\tau_k^* \in$ FOA
42. for each $a_j^* \in A_{\tau_k^*}$
43. execute action a_j^*
44. endfor
45: endfor
46. end cognitive process

6.3 Conclusion

Two robo-cognitive architectures were developed in this chapter. The methodology, developed in section 6.1, constructs optimal episodic memory, based on Baddeley's model of working memory, for the real-time, high-level control of an autonomous vehicle.

The methodology, developed in section 6.2, constructs a focus of attention from activated memory, based on Cowan's attentional focus model of working memory, for the real-time, high-level control of an autonomous vehicle.

The cognitive reasoning processes of both architectures, uses the memory representation (LTM) and memory quantification (AEFQ), developed in chapter 4. Both architectures will be evaluated by simulation in the next chapter.

Chapter 7

Evaluation by Simulation

In order to evaluate the suitability of the two architectures developed in sections 6.1 and 6.2, four practical uses cases are defined. Each use case is executed in an unmanned aerial vehicle simulation. This chapter defines the use cases and simulation setup in detail. Each simulation is then executed and the results are evaluated and discussed in detail.

7.1 Real-time Episodic Memory Construction in Cognitive Control of Autonomous Vehicles

Two use cases are designed for the evaluation of the robo-cognitive architecture, developed in section 6.1. The first use case executes a recharging mission and the second use case executes a medical package delivery mission. The architecture provides real-time, high-level control using episodic memory construction, based on Baddeley's working memory model.

7.1.1 Use cases

Use case 1

From the Home (I) location, arm (turn on) the motors, ascend to a specified operational height and fly to the Charging point (IV), passing over the Collection and Delivery locations. Descend on the charging point and disarm (turn off) the motors. As collection and delivery are not performed in the mission, these destinations are ignored by the UAV.

Figure 7.1 Use case 1 - unmanned aerial vehicle recharging.

Use case 2

From the Home (I) location, arm (turn on) the motors, ascend to a specified operational height and fly to the Collection point (II). Descend and collect the cargo, then ascend to the specified operational height and fly to the Delivery point (III). Descend at the delivery point and deliver the cargo. Ascend to a new operational height and fly to the Charging point (IV) for recharging. Descend on the charging point and disarm (turn off) the motors.

Figure 7.2 Use case 2 - unmanned aerial vehicle medical equipment delivery.

7.1.2 Simulation setup

The unmanned aerial vehicle in the simulations, have the states defined in section 4.1, figures 4.1 and 4.2 and the LTM for the robo-cognitive architecture of the unmanned aerial vehicle is the set of state transitions, defined in section 4.1.2 and illustrated in figure 4.3.

The simulations were executed on an Intel i7 laptop computer with 2.97GHz quad-core CPU, 16Gb RAM and an Intel HD Graphics 4000 video adapter. Figure 7.3 illustrates the system architecture of the simulation environment. The cognitive architectures developed in section 6.1 are implemented in a C# program, representing the cognitive reasoning process. A simulation client program, which passes relevant control commands to the simulator module, is developed in C++. Communication between the C++ simulation components and the CRP is performed via a Redis cache database, thereby providing functional abstraction. The simulation environment for the unmanned aerial vehicle is the Unity games engine.

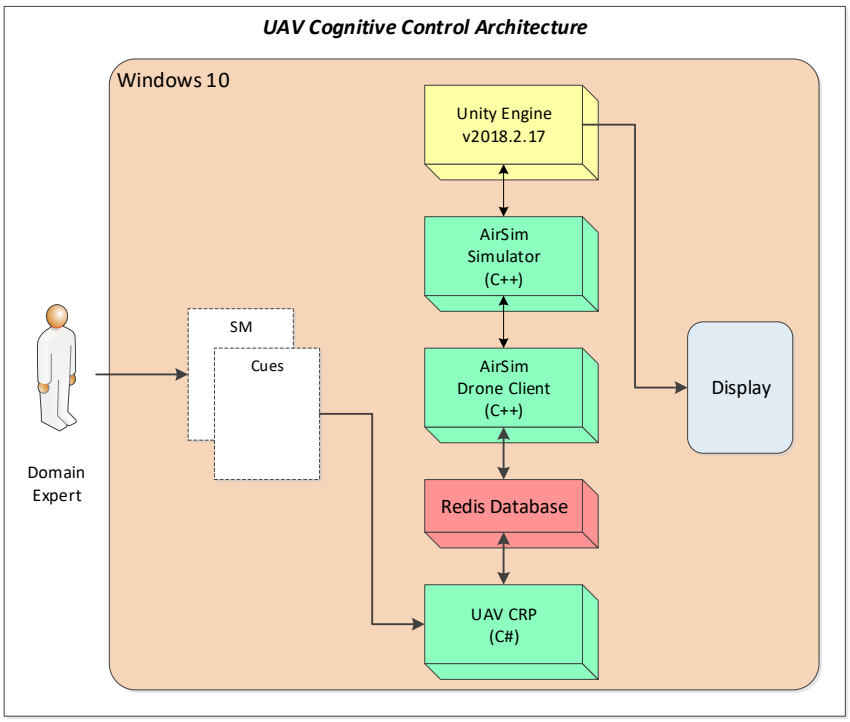

Figure 7.3 The main components of the simulation platform architecture.
The CRP passes messages based on actions selected by the central executive, to the simulation client, via the Redis database middleware. The simulation client formulates and passes the control command the simulation engine, which executes the commend. The result, i.e. behaviour of the UAV is displayed in the Unity games engine simulation.

For each of the use cases, an annotated video of the simulation is recorded and published to YouTube:

1. Use case 1 - Cognitive Robotics - Autonomous UAV recharging [127]
2. Use case 2 - Cognitive Robotics - Autonomous medical supplies delivery [128]

7.1.3 Evaluation criteria

The methodology is evaluated by simulation, where a UAV autonomously executes two use cases, one simple and one more complex. The performance measures for each of the use cases are:

1. Success – Measured by inspecting the completeness of the learned state flow, for each mission and;
2. Reasoning – Measured by inspecting the level of velocity control of the UAV, based on reasoning about the statistical fitness of each state transition.

7.1.4 Simulation results

7.1.4.1 Use case 1 - results

Figure 7.4 shows the resulting state flow for use case 1, dynamically constructed by the central executive during the execution of use case 1. The start state is state s1, i.e. Motors Off.

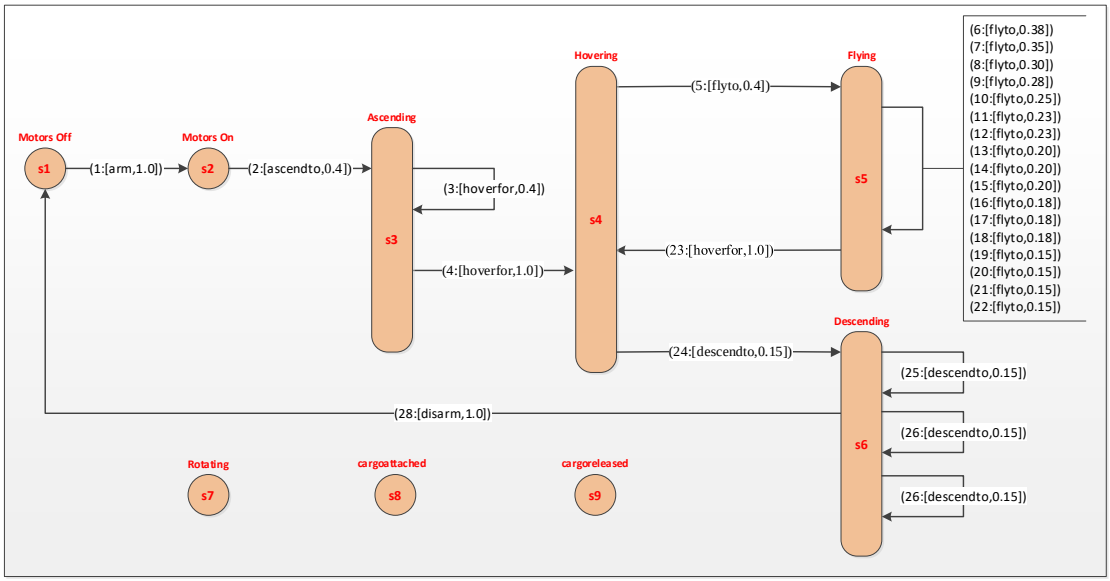

Figure 7.4 Resulting state flow of use case 1.
The diagram shows the applicable states and state transitions, with each transition labelled with the task number, relevant command (action) and state transition fitness.

The resulting state flow can be saved and, provided the mission and operational conditions remain the same, may be used as a high-level controller to execute similar, subsequent missions.

To demonstrate the usefulness of the memory item quantification in the dynamic control of the velocity of the UAV, the resulting probability distribution produced by the quantification was used. In this study, the velocity of the UAV (eq. 7.1), is dynamically derived by multiplying the fitness, Π (eq. (4.20)) by an arbitrary factor, chosen by the designer.

$$velocity = 0.3\Pi \tag{7.1}$$

After running some simulations and observing the behaviour of the UAV, a factor of 0.3 was chosen.

Figure 7.5 below shows the dynamic control of the velocity, derived from the state transition fitness. The graph shows the reduction in velocity, in accordance with the reduction in fitness of the “fly” state transition, as the UAV nears its destination.

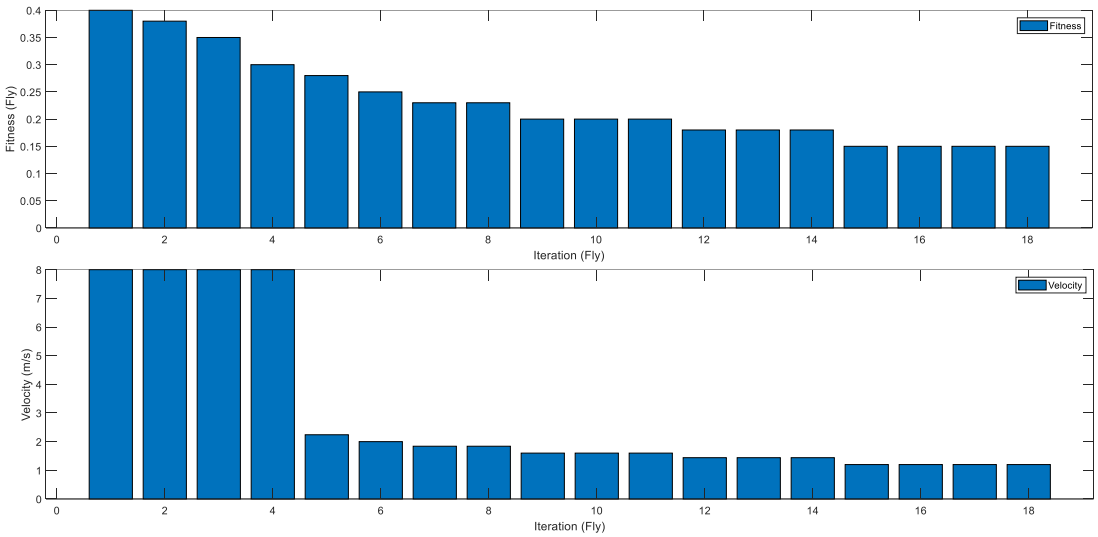

Figure 7.5 Dynamic velocity adjustment during use case 1.
The graph shows automatic adjustment of the velocity (bottom) which corresponds to the fitness (top).

On the graph, the target destination (charging) of mission 1 can be seen at task 18. The graph shows that the UAV proportionally reduces its velocity as it approaches its destination (see eq. 7.1).

Figures 7.6 and 7.7 show some key stages in the simulation for use case 1. The window at the bottom shows the central executive finding the optimal state transitions and sending the corresponding actions to the simulator. The window on the left shows the results of the simulator as it performs the actions received from the central executive.

Figure 7.6 below, shows the UAV reducing its velocity, using the dynamic velocity adjustment derived from the fitness quantification, as the UAV approaches its target. This behaviour is used to evaluate performance measure 2 (defined in see section 7.1.3).

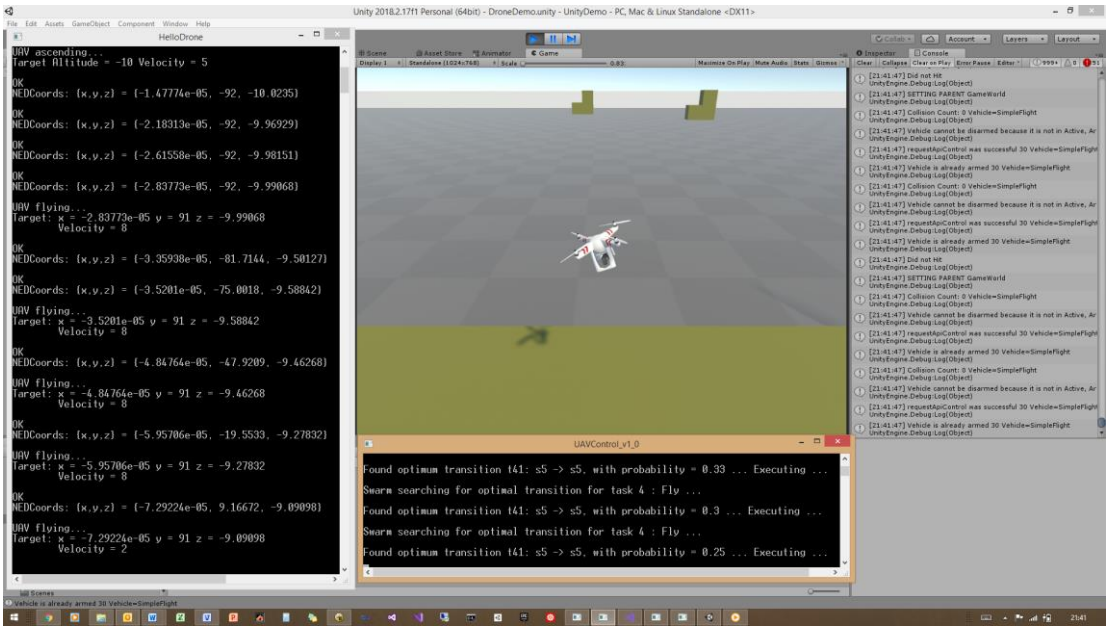

Figure 7.6 UAV adjusting its velocity.
The image shows the UAV reducing its velocity, based on the real-time calculation of the fitness of the "flyto" state transition, as it approaches its target destination

As the UAV approaches its target, $\Pr(\varphi_i^r < \varphi_j^m)$ is reduced from 0.3 to 0.25 and the velocity of the UAV (indicated in the window left) is adjusted accordingly from 8.00 m/s to 2.00 m/s.

Figure 7.7 below, shows the successful completion of the mission. This behaviour is used to evaluate performance measure 1.

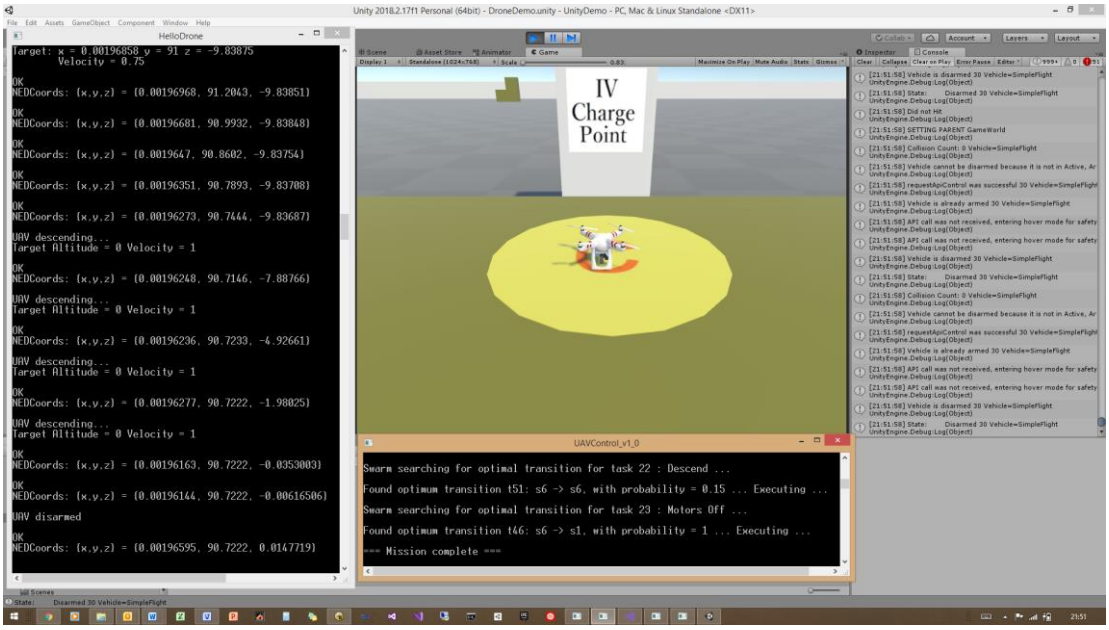

Figure 7.7 UAV reaching its destination and completing the mission.
When the UAV reached its target destination (the charging point), it descends and successfully completes the mission.

7.1.4.2 Use case 2 - results

Figure 7.8 shows the resulting state flow for use case 2, dynamically constructed by the central executive during the execution of use case 2. The start state is state s1, i.e. Motors Off.

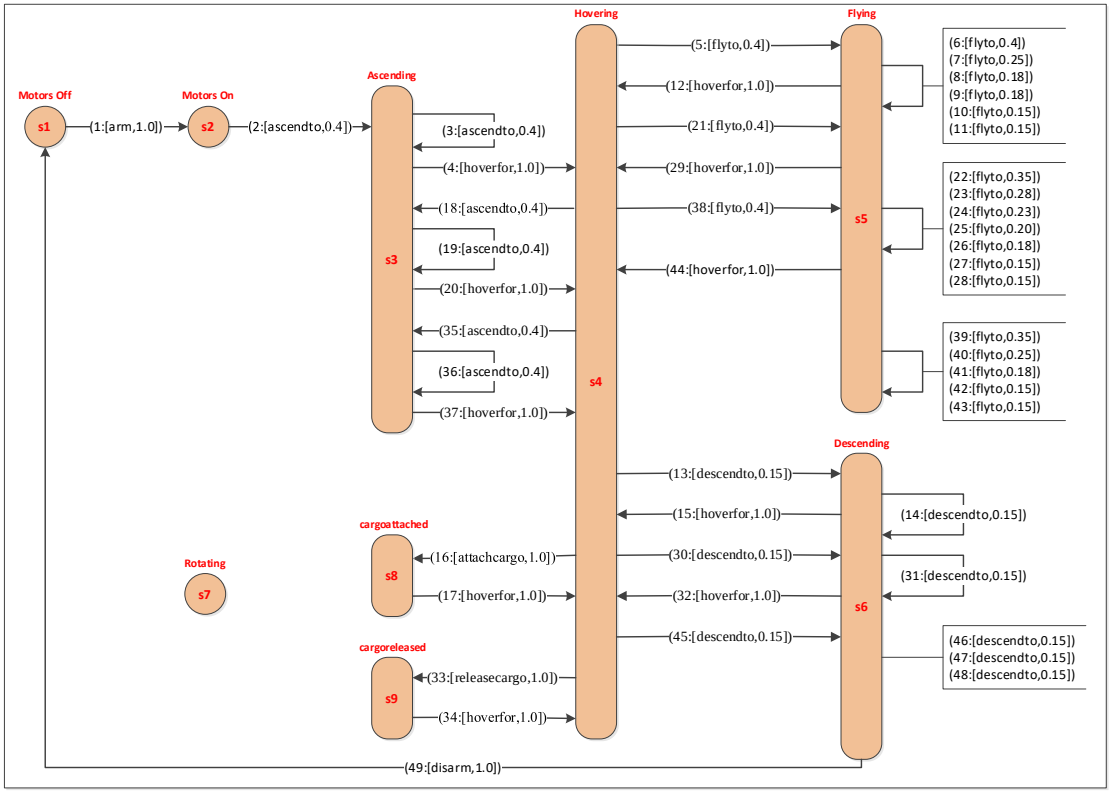

Figure 7.8 Resulting state flow constructed for use case 2.
The diagram shows the applicable states and state transitions, with each transition labelled with the task number, relevant command (action) and state transition fitness.

The resulting state flow in figure 7.9 shows the dynamic velocity control, derived from the state transition fitness. The graph shows the corresponding reduction in velocity every time the UAV near its target.

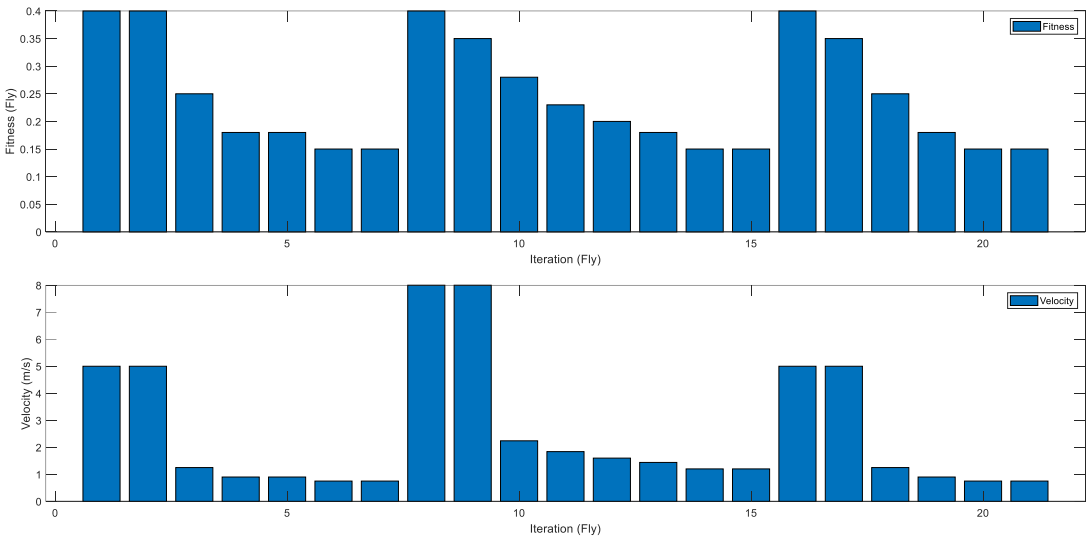

Figure 7.9 Dynamic velocity adjustment during mission 2.
The graph shows automatic adjustment of the velocity (bottom) which corresponds to the fitness (top).

On the graph, the three target destinations (collection, delivery and charging) of the missions can be seen at tasks 7, 15, and 21. The graph shows that the UAV proportionally reduces its velocity as it approaches each of the destinations (see eq. 7.1).

Figures 7.10, 7.11 and 7.12 shows some key stages in the simulation for use case 2. Figure 7.10 shows the UAV in process of collecting its cargo.

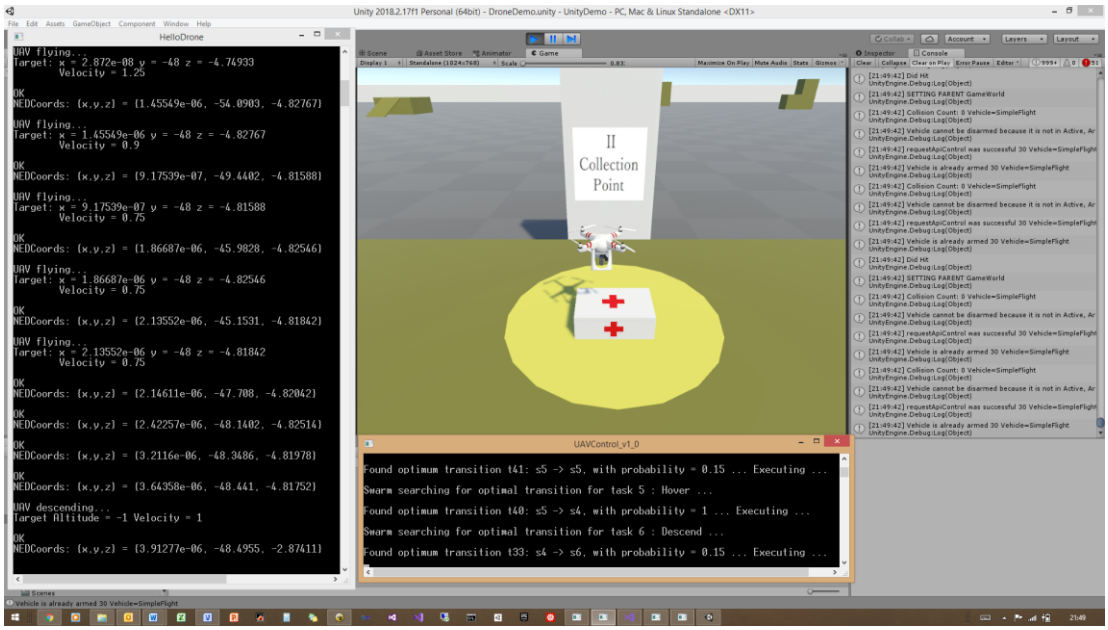

Figure 7.10 UAV collecting its cargo at the collection point.

Figure 7.11 shows the UAV adjusting its velocity in accordance with the fitness of the state transition, fly.

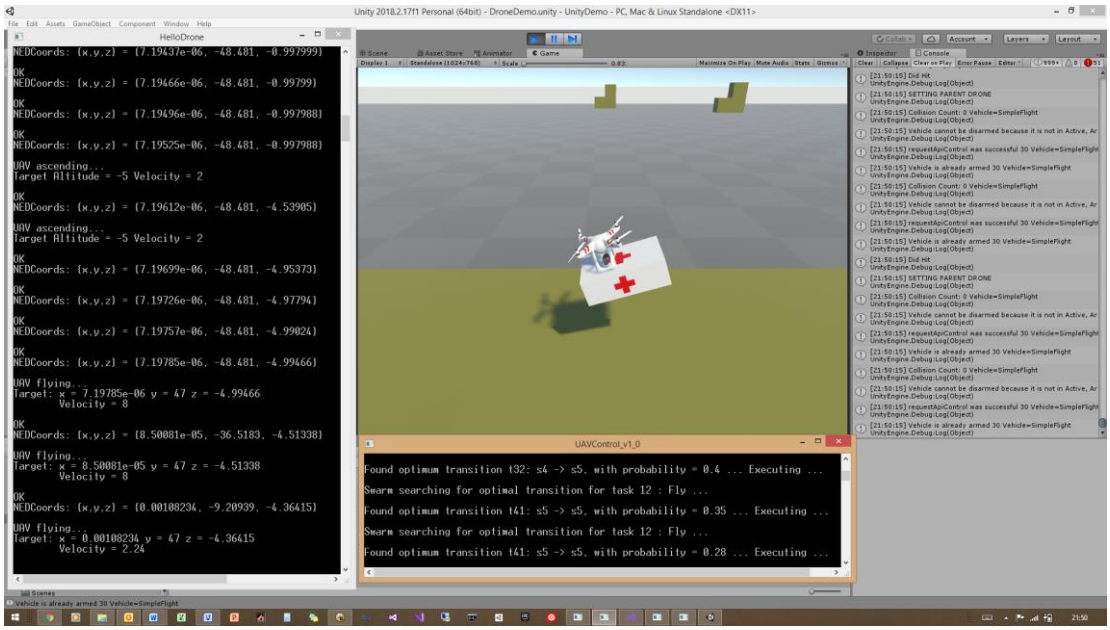

Figure 7.11 UAV reducing its velocity as it approaches its target destination.

Figure 7.12 shows the UAV successfully delivering its cargo at the specified target location.

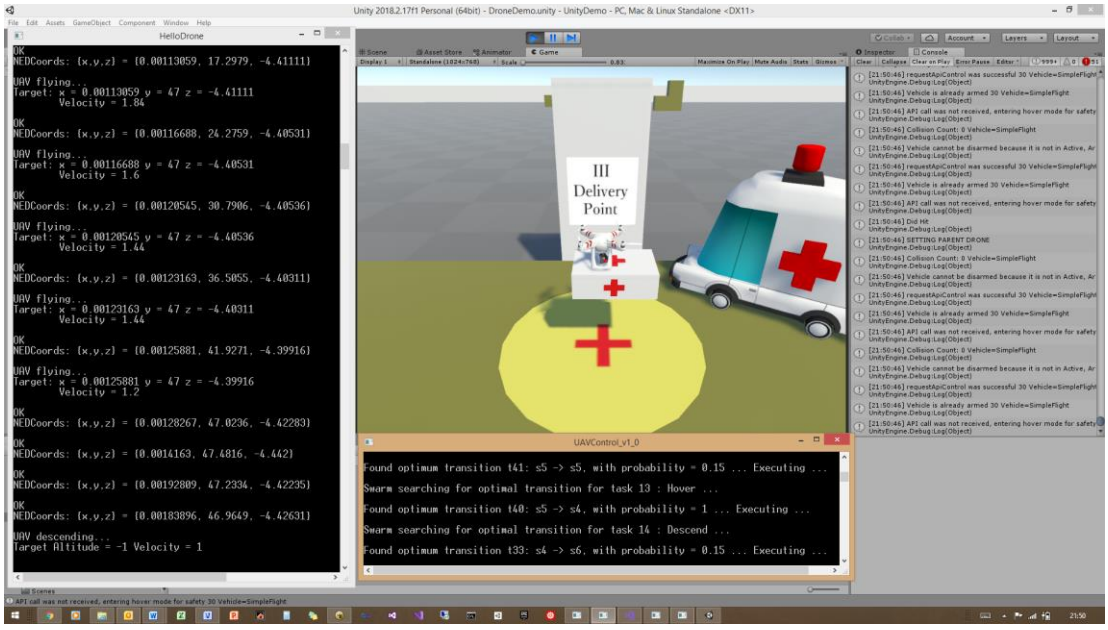

Figure 7.12 UAV successfully delivering its cargo.

7.1.5 Discussion

The domain expert submits the mission definition, the long-term memory, short-term memory and initial environmental stimuli to the central executive for execution. Each mission contains a list of actions to be performed successfully in order to successfully complete the mission. The mission, long-term memory, short-term memory and environmental stimuli are provided in extensible markup language (XML) format. After the initial loading of the definitions, modification of either the long-term memory, or the mission, means an update of a state transition in the long-term memory or an update of the mission and environmental data definitions. Changing between missions is simply a number change in the configuration of the central executive, which may be performed by the operator or autonomously, based on the central executive reasoning. This achieves the one objective of the study, i.e. simplifying the maintenance of the knowledge of a remotely deployed autonomous vehicle.

In the implementation of the central executive, a mission task was repeatedly executed, until the task objective has been reached. Due to some lagging in the communication between the AirSim simulator and Unity games engine, it was observed that, at high velocity, the UAV would overshoot its target destination in the Unity games engine. The delay in the Unity games engine to calculate the UAV's current position, causes the UAV to miss its objectives.

However, with the autonomous and dynamic velocity control, an unexpected result was observed. The UAV would autonomously correct its positioning, by repeating the task, while constantly reducing its velocity according to the fitness of the task. At low velocity, the positioning of the UAV was quite accurate and it could achieve its objectives. With the autonomous velocity control, the UAV was able to successfully reach the charging station in use case 1 and was able to successfully collect and deliver its cargo in use case 2.

The results (figures 7.4 and 7.8) show that the UAV can successfully execute its missions by optimizing the expert-provided knowledge and dynamically generating and executing its high-level controller. The behaviour of the UAV in both use cases are demonstrated in the accompanying videos [127, 128].

Although this approach is suitable for the successful execution of missions which involve a set of tasks to be executed in sequence, one-by-one, most robots can perform multiple tasks, often independent and in parallel. For example, a state flow for flight-control, a state flow for camera control and a state flow for gripper control. Memory recall, therefore, needs to optimize the memory for multiple functions. The SPSO algorithm is able to produce an optimal set of memory items, based on a single objective (function). However, multiple objectives are not possible with the SPSO, without significant changes to the set-based operators. Changing the

set-based operators will result in the SPSO architecture losing the individual and social characteristics, typical of swarm behaviour. In order to retain the swarm characteristics, while providing memory optimization for multiple objectives, a novel games-theoretic PSO algorithm, is developed in the next section.

Performance

The simulation was executed repeatedly, with consistent results. It was uncertain whether all the required actions of the missions would be selected and executed successfully or whether some actions would be missed. However, figures 7.4 and 7.5 (for use case 1) and figures 7.8 and 7.9 (for use case 2) show that the central executive successfully executed all actions of the missions submitted by the domain expert. This success was also observed during the simulation. Figures 7.5 and 7.9, for use case 1 and 2 respectively, show the successful reasoning for velocity control, using statistical reasoning. The figures show the corresponding velocity adjustment, based on the fitness (probability).

In addition, conducting the simulations also showed the following general benefits:

- The approach is less error-prone and requires less communication bandwidth to maintain because, in our approach, knowledge and missions are defined using a simple structure. The trigger memory item of state transitions is constructed as a simple conjunction of propositions, and is therefore more intuitive to the domain expert. Moreover, the knowledgebase and missions can be modified independently, reducing errors during the updating process.
- There is no re-learning of complex statistical reasoning models or networks whenever the knowledge or evidence changes because, in our approach, potential solutions are evaluated in real-time and a statistical model for reasoning is generated in real-time by the MEP.
- Autonomous behaviour can be controlled more effectively because in our approach, the probabilities and marginal probabilities provided by the AEFQ algorithm enables finer control of the statistical fitness evaluations of the state transitions.
- The high-level control provided by the CRP is more representative of human cognition, because in our approach, the open world assumption is followed. This means the action of a state transition may be less probable, but not impossible. This gives the CRP powerful reasoning capabilities.
- The fitness of a state transition is a true representation of the environment because, the MEP applied in our approach, guarantees an accurate probability assignment, based

only on the constraint averages derived from the mission constraints and environmental evidence. There are no other subjective control parameters or bias in the fitness quantification.

Time Efficiency

The objective of this study is the real-time, high-level control provided by the CRP. Therefore, the time efficiency of the CRP, i.e. the time taken by the AE-SPSO to find an optimal solution for a mission task, is evaluated. Optimization algorithms, including the PSO algorithm, are known for the extensive time it takes to converge on an optimum. This is especially true for large, multi-dimensional and real search spaces. However, in this approach, the search space is finite and discrete, allowing the AE-SPSO to find optimal solutions in acceptable and sufficient time. Moreover, the control parameters of the AE-SPSO make it easy to scale the performance of the PSO when the search space increases.

Figure 7.13 below, shows the time the CRP took to find an optimal solution for each of the tasks of each mission.

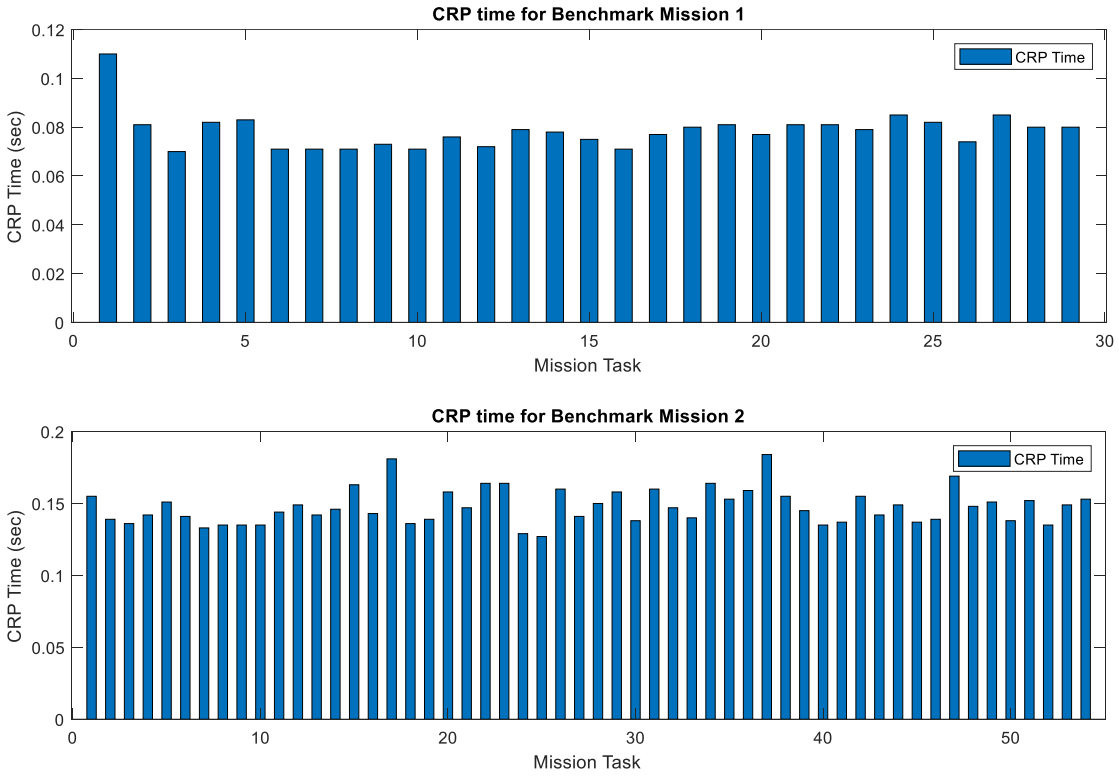

Figure 7.13 Cognitive reasoning process (CRP) time for use cases 1 and 2.
The CRP time for use case 1 (mission 1) is shown on top and the CRP time for use case 2 (mission 2), is shown below.

The average CRP time for use case 1 was 0.0785 sec and for use case 2, the average CRP time was 0.1477 sec. These times were found to be completely suitable for the high-level control of the UAV, while executing its missions.

Simulation constraints

The performance of the UAV may appear slow in the videos. This is because the complex integration architecture of the AirSim simulator and the Unity games engine is not optimal and causes a considerable time lag between the simulator and the games engine.

Similar to the use case 1 behaviour discussed in section 7.1.5, it was observed that, at high velocity, the UAV would overshoot its target destination in the Unity games engine. This resulted in the target position parameters reported by the AirSim to be inconsistent from that reported by the games engine. This caused the UAV to wrongly interpret its position and therefore miss its objectives. To improve the performance, a delay was explicitly implemented between the execution of mission tasks, in order to give the games engine and simulator time to synchronise. Assisted by the explicit delay, the UAV would autonomously correct its positioning, by repeating the task, while constantly reducing its velocity according to the fitness of the task. At low velocity, the positioning of the UAV was quite accurate and it could achieve its objectives. With the autonomous velocity control, the UAV was able to successfully reach the charging station in use case 1 and was able to successfully collect and deliver its cargo in use case 2.

It is plausible that a similar problem could occur in a real-world scenario, where a physical UAV is used. Therefore, correction control measures for positioning correction will have to be developed. These measures would use the fitness quantification, similar to the way velocity control is derived, as discussed in section 7.1.4.1 and using eq. (7.1).

7.2 Real-time Activated memory Construction for Cognitive Control of Autonomous Vehicles

Two simulations are designed for the evaluation of the robo-cognitive architecture, developed in section 6.2. The architecture provides real-time, high-level control using activated memory and focus of attention construction, based on Cowan's attentional focus theory of working memory.

7.2.1 Use Cases

The methodology is evaluated by simulation, where a UAV autonomously executes two use cases. In the simulation, a drone station is located at the Surbiton Health Centre for community support.

A domain expert defined two use cases for the UAV. The first mission is a request for a UAV to make a delivery of a medical package and is comprised of 25 tasks. The second mission is a request for aerial surveillance and is comprised of 15 tasks. During each mission, each of the tasks is presented to the cognitive process of the robo-cognitive architecture, as a cue. The UAV must successfully reason about each cue and successfully select and execute the command derived from the FOA.

Use case 3

Scenario: There is a request from healthcare personnel for medical equipment to be delivered to Surbiton train station as a result of a medical incident.

The proposed UAVs delivery approach facilitates telemedicine and makes it possible for medical professionals to interact with patients remotely, saving time by delivering urgent medication, prescription orders for medicine to the doorsteps of surgeries and care homes in the community.

It could reduce overcrowding by making this proposed approach more practical for non-urgent patients to receive care in local surgeries closer to home.

Use case 4

Scenario: A security incident was reported by a Furniture Company and the police requested aerial surveillance of the area. The proposed UAVs delivery approach facilitates on-demand surveillance support to local law-enforcement and security personnel.

7.2.2 Simulation setup

7.2.2.1 Simulation architecture

Hardware: The simulations were executed on an Intel i7 laptop computer with 2.97GHz quad-core CPU, 16Gb RAM and an Intel HD Graphics 4000 video adapter.

Software: The simulations were performed using the Drone-kit Software-in-the-loop (SITL) (Python version) quadcopter and the ArduPilot Mission Planner Ground Control Station (GCS). Two bespoke components were developed: the UAV cognitive process (UAV-CP), in .NET/C#

and a high-level UAV control server (UAV-HC), in Python. The UAV-CP and UAV-HC components are integrated using a Redis Cache middleware layer, thereby abstracting the autonomous vehicle (UAV in this case) platform from the cognitive process. The UAV-HC server listens for messages from the UAV-CP, and passes the relevant low-level control commands to the UAV, using MAVProxy protocol. All the components are deployed on a computer with the Microsoft Windows 8.1 operating system.

Figure 7.14 below illustrates the simulation's software architecture:

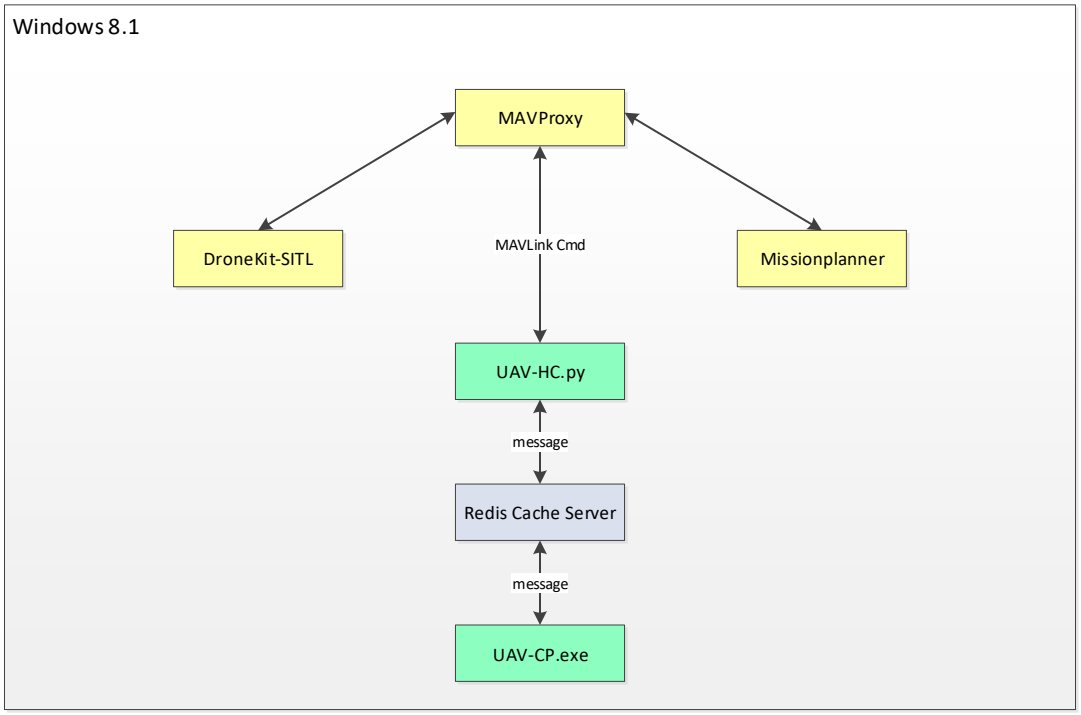

Figure 7.14 Simulation platform system architecture for uses cases 3 and 4.
The cognitive reasoning process (C#), passes a message on to the high-level control component (Python), via the Redis database middleware. The high-level control component formulates a MAVProxy command which is passes to the DroneKit simulation platform. The result, i.e. behaviour of the UAV is displayed in Missionplanner ground control station.

7.2.2.2 Simulation parameters

In order to measure the time efficiency of robo-cognitive architecture, an arbitrary maximum processing time limit (MPt) of 2 seconds is set by the designer for the use cases. Given that the evaluations are performed in simulation, a threshold of 2 seconds are deemed a sufficient response time for the cognitive processing time (CPt) of the central executive. The lapse time of the cognitive processing (memory recall and action selection) is evaluated against this time threshold and is assumed efficient if the CPt is below the MPt, for each cue processed by the central executive. In order to give the UAV time to complete a task in the simulation, explicit time delays were set in the UAV control program. Therefore, evaluating the task processing

time (TPt) against the 2 second time limit is of little use to evaluate cognitive performance. However, it is still useful, from a practical point of view, to inspect the time it takes the UAV to complete a task in the simulation.

The search space of the simulation is the long-term memory, shown in section 4.1, figure 4.3. In order to keep the introduction of the methodology clear and simple, only two UAV functions were used. However, it is clear that, without loss of generality, the proposed methodology can be scaled to a large number of functions. However, an increase in the number of functions will increase the size of the long-term memory. PSO was chosen for optimization in the cognitive process, due to its scalability. The three most important PSO runtime parameters, which may potentially impact the cognitive processing time, are:

- Swarm size (number of particles)
- Particle size (number of elements, when set-based), and;
- Iterations (for successful exploration)

The parameter sizes must be scaled according to the size of the search space. In this study, both use cases were executed repeatedly, and the PSO parameters adjusted until the CPt was consistently less than the MPt. Because the long-term memory for this simulation is relatively small, the following parameter values gave acceptable processing times which were below the MPt:

- Swarm size = 3
- Particle size = 3
- Iterations = 10

7.2.3 Evaluation criteria

Performance Measures

The performance measures for each of the use cases are:

1. Correctness – Measured by inspecting the specific states, the state transitions and transition sequence. The learned state flow is inspected for correctness against the reference state flows in figures 4.1 and 4.2.
2. Cognitive Processing time (CPt) – Measured by inspecting the lapsed time of the cognitive process. The CPt is defined as the lapsed time between the receipt of a cue and sending the appropriate command to the UAV controller. The CPt represents the cognitive processing time (or “thinking” time) of the central executive.

3. Cue processing time (TPt) - Measured by inspecting the lapsed time of cue (task) processing. The TPt is defined as the lapsed time between the receipt of a cue and the UAV completing the task. The TPt represents the complete processing time (or "thinking and doing" time) of a task.

7.2.4 Simulation Results

Overview

The simulation environment is shown in Figure 7.15. The flight behaviour of the UAV is shown in the Mission Planner CGS on the top, while the UAV-CP results are shown in the window on the bottom left of the screen. The window on the bottom right shows the results of the UAV-HC server. Figure 7.15 shows the UAV's home location (H), the Surbiton Health Centre, from where medical equipment and tests are supplied to the community.

The UAV-CP window shows the stimulus (mission task) and the central executive, composed of the active memory (AM), the focus of attention (FOA) and the action selected and sent to the UAV. The AM shows the collection of particles and the coalition of which it is a member. The relevant assets of each particle are shown as well, along with the particle's utility, quantified by the AEFQ algorithm. The FOA shows the asset receiving the focus. determined by its relevance to the stimulus. From the FOA, actions are selected for execution by the central executive and passed to the UAV-HC. All the key points of the simulation environment shown in figure 7.15.

Note that, at the start of the simulation, there are two start states, S1 and S10, for the flight-control function and the gripper control function respectively.

The results of use case 3 are presented in tables 7.1 and 7.2, including the learned state flow in figure 7.16. The results for use case 4 are presented in tables 7.3 and 7.4, including the learned state flow in figure 7.17. Tables 7.1 and 7.3 show the results of the cognitive process where columns 1 and 2 describes the cues received by the cognitive process (algorithm 6.4). Columns 3 - 5 shows the results of the AM construction (algorithms 6.2 and 6.3) and column 6 shows the FOA, derived from the coalition structure. Tables 7.2 and 7.4 present the results for the cognitive processing time and cue processing time, for the two use cases, respectively.

For each use case, an annotated video of the full mission simulation was recorded and published to YouTube:

1. Use case 3 – Cognitive Robotics - On-demand UAV delivery of COVID-19 equipment [129]
2. Use case 4 – Cognitive Robotics - On-demand UAV security surveillance support [130]

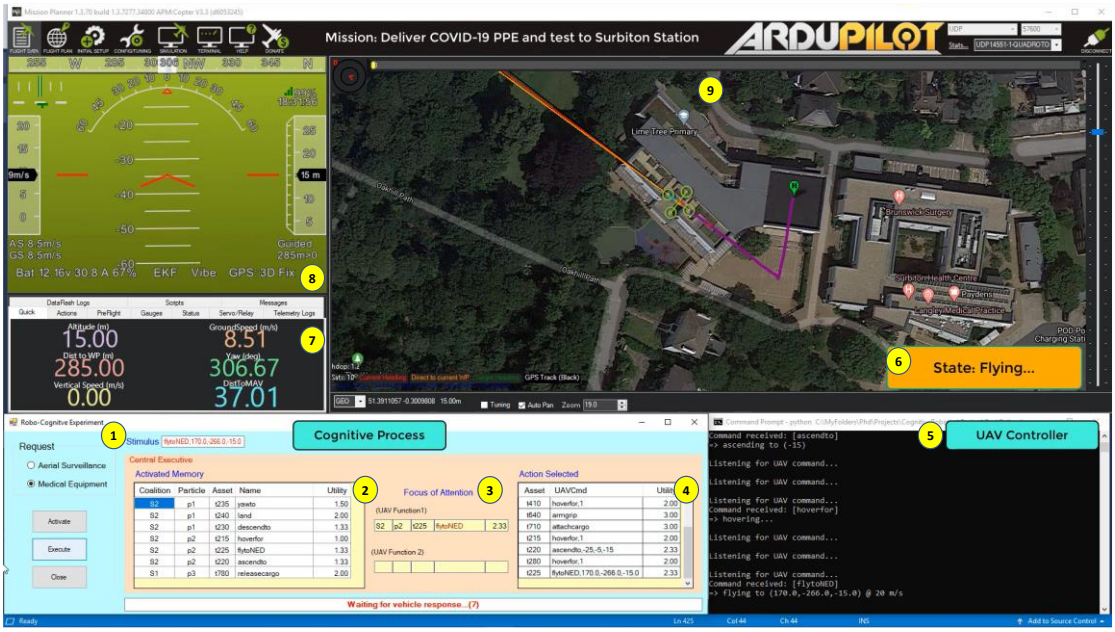

Figure 7.15 Main simulation functions for use cases 3 and 4.
The image shows the main functions of the simulation environment: 1) the cognitive process; 2) the UAV high-level controller and 3) the MissionPlanner ground control station.

The points of interest of the simulation environment, shown in figure 7.15, are:

1. A stimulus, provided to the cognitive process as a cue - Column 2,
2. The AM of the cognitive process – Column 3,
3. The FOA of the cognitive process – Column 6,
4. The action selected by the central executive – Column 7,
5. The UAV controller executing the command,
6. The state of the UAV after executing the command,
7. The attitude of the UAV, displayed in the GCS,
8. The Head-Up Display (HUD) of the UAV, displayed in the GCS,
9. The map of the environment, displayed in the GCS.

Use case 3 results (medical supplies delivery**)**

Table 7.1 shows the results of the simulation for use case 3 – delivering medical equipment to an incident. For this simulation, two functions, flight control and gripper control are applicable.

Table 7.1 shows the results for use case 3 – medical supplies delivery.

Table 7.1 Cognitive reasoning results for use case 3.
For each cue (task), the results of the cognitive process (memory recall, activated memory, focus of attention and action selection) are shown. (Cues relevant to function 1 are shown in blue and cues relevant to function 2 are shown in red. The asset and particle, relevant to the cue, is highlighted in **bold**) (Each "p" represents a specific particle and each "t" represents an asset (memory item) of the particle)

Cues		**Activated Memory**			**Focus of Attention**	**Action Selection**
No	**Cue**	**Optimized LTM (Swarm)**	**Individual Utility**	**Coalition Structure**	**Asset**	**Command**
1	arm	p1{ } p2{ **t10(2.0)** } p3{ t640(2.0) }	[p1(0.0), **p2(2.0)**, p3(2.0)]	{ {p3},{p1,**p2**},{} }	**t10** - arm	arm
2	takeoff	p1{ t640(2.0) } p2{ t70(2.0), **t80(2.0)** } p3{ }	[p1(2.0), **p2(4.0)**, p3(0.0)]	{ {p3,**p2**},{p1},{} }	**t80** - takeoff	takeoff,10.0
3	hoverfor	p1{ **t150(1.0)** } p2{ t640(2.0) } p3{ }	[**p1(1.0)**, p2(2.0), p3(0.0)]	{ {p3,**p1**},{p2},{} }	**t150** - hoverfor	hoverfor,1.0
4	flytoNED	p1{ t640(2.0) } p2{ **t225(1.33)**, t220(1.33), t215(1.0) } p3{ t235(1.5), t240(2.0), t230(1.33) }	[p1(2.0), **p2(3.7)**, p3(4.8)]	{ {p3,**p2**},{p1},{} }	**t225** - flytoNED	flytoNED,-25,-5,-15
5	hoverfor	p1{ **t345(1.0)**, t365(1.5), t355(1.08) } p2{ t640(2.0) } p3{ }	[**p1(3.6)**, p2(2.0), p3(0.0)]	{ {p3,**p1**},{p2},{} }	**t345** - hoverfor	hoverfor,1.0
6	descendto	p1{ t640(2.0) } p2{ t235(1.5), t225(1.33) } p3{ t220(1.33), **t230(1.33)**, t215(1.0), t240(2.0) }	[p1(2.0), p2(2.8), **p3(5.7)**]	{ {**p3**,p2},{p1},{} }	**t230** - descendto	descendto,-25,-5,-1
7	hoverfor	p1{ t640(2.0) } p2{ t425(1.08) } p3{ **t410(1.0)** }	[p1(2.0), p2(1.1), **p3(1.0)**]	{ {**p3**,p2},{p1},{} }	**t410** - hoverfor	hoverfor,1.0
8	armgrip	p1{ **t640(2.0)** } p2{ t220(1.08), t215(1.0), t235(1.5), t230(1.08) } p3{ t240(2.0), t225(1.08) }	[**p1(2.0)**, p2(4.7), p3(3.1)]	{ {**p1**},{p2,p3},{} }	**t640** - armgrip	armgrip

Table 7.1 Cognitive reasoning results for use case 3 - continued

Cues		Activated Memory			Focus of Attention	Action Selection
No	**Cue**	**Optimized LTM (Swarm)**	**Individual Utility**	**Coalition Structure**	**Asset**	**Command**
9	attachcargo	p1{ **t710(2.0)** } p2{ t700(2.0) } p3{ t240(2.0), t230(1.08), t215(1.0), t225(1.08), t235(1.5), t220(1.08) }	[**p1(2.0)**, p2(2.0), p3(7.7)]	{ {p3},{p2,**p1**},{} }	**t710** - attachcargo	attachcargo
10	hoverfor	p1{ t230(1.08), t225(1.08), **t215(1.0)**, t235(1.5) } p2{ t240(2.0), t220(1.08) } p3{ t780(2.0) }	[**p1(4.7)**, p2(3.1), p3(2.0)]	{ {p2,**p1**},{p3},{} }	**t215** - hoverfor	hoverfor,1.0
11	ascendto	p1{ t780(2.0) } p2{ **t220(1.33)**, t215(1) } p3{ t225(1.33), t230(1.33), t235(1.5), t240(2.0) }	[p1(2.0), **p2(2.3)**, p3(6.2)]	{ {p3,**p2**},{p1},{} }	**t220** - ascendto	ascendto,-25,-5,-15
12	hoverfor	p1{ t300(1.5), t285(1.08), **t280(1.0)** } p2{ } p3{ t780(2.0) }	[**p1(3.6)**, p2(0.0), p3(2.0)]	{ {p2,**p1**},{p3},{} }	**t280** - hoverfor	hoverfor,1.0
13	flytoNED	p1{ t780(2.0) } p2{ t215(1), t235(1.5), t230(1.33) } p3{ t240(2.0), **t225(1.33)**, t220(1.33) }	[p1(2.0), p2(3.8), **p3(4.7)**]	{ {p2,**p3**},{p1},{} }	**t225** - flytoNED	flytoNED,170.0,-266.0,-15.0
14	hoverfor	p1{ **t345(1.0)** } p2{ t780(2.0) } p3{ t365(1.5), t355(1.08) }	[**p1(1.0)**, p2(2.0), p3(2.6)]	{ {p2},{}{p3,**p1**}, }	**t345** - hoverfor	hoverfor,1.0
15	descendto	p1{ t220(1.33), t240(2.0), t225(1.33) } p2{ t780(2.0) } p3{ t215(1.0), **t230(1.33)**, t235(1.5) }	[p1(4.7), p2(2.0), **p3(3.8)**]	{ {**p3**,p1},{p2},{} }	**t230** - descendto	descendto,170.0,-266.0,-5,1

Table 7.1 Cognitive reasoning results for use case 3 - continued

Cues		Activated Memory			Focus of Attention	Action Selection
No	**Cue**	**Optimized LTM (Swarm)**	**Individual Utility**	**Coalition Structure**	**Asset**	**Command**
16	hoverfor	p1{ t780(2) } p2{ **t410(1.0)**, t425(1.08) } p3{ }	[p1(2.0), **p2(2.1)**, p3(0.0)]	{ {p3,**p2**},{p1},{} }	**t410** - hoverfor	hoverfor,1.0
17	releasecargo	p1{ **t780(2.0)** } p2{ t230(1.08), t240(2.0), t225(1.08), t220(1.08) } p3{ t235(1.5), t215(1.0) }	[**p1(2.0)**, p2(5.2), p3(2.5)]	{ {p2,p3},{**p1**},{} }	**t780** - releasecargo	releasecargo
18	disarmgrip	p1{ t230(1.08), t240(2.0) } p2{ **t830(2.0)** } p3{ t235(1.5), t215(1.0), t220(1.08), t225(1.08) }	[p1(3.1), **p2(2.0)**, p3(4.7)]	{ {p2},{}{p3,**p1**}, }	**t830** - disarmgrip	disarmgrip
19	hoverfor	p1{ t230(1.08), t225(1.08), t240(2.0), t220(1.08), **t215(1.0)**, t235(1.5) } p2{ t640(2.0) } p3{ }	[**p1(7.7)**, p2(2.0), p3(0.0)]	{ {p3,**p1**},{p2},{} }	**t215** - hoverfor	hoverfor,1.0
20	ascendto	p1{ t240(2.0), t225(1.33), t215(1.0) } p2{ t640(2) } p3{ t235(1.5), t230(1.33), **t220(1.33)** }	[p1(4.3), p2(2.0), **p3(4.2)**]	{ {**p3**,p1},{p2},{} }	**t220** - ascendto	ascendto,170.0,-266.0,-15.0
21	hoverfor	p1{ t640(2.0) } p2{ **t280**(1.0), t285(1.08) } p3{ t300(1.5) }	[p1(2.0), **p2(2.1)**, p3(1.5)]	{ {p1},{p3,**p2**},{} }	**t280** - hoverfor	hoverfor,1.0
22	flytoNED	p1{ t215(1), **t225(1.33)**, t220(1.33) } p2{ t230(1.33), t240(2.0), t235(1.5) } p3{ t640(2.0) }	[**p1(3.7)**, p2(4.8), p3(2.0)]	{ {p2,**p1**},{p3},{} }	**t225** - flytoNED	flytoNED,0,0,-15.0
23	hoverfor	p1{ t640(2.0) } p2{ **t345(1.0)**, t365(1.5), t355(1.08) } p3{ }	[p1(2.0), **p2(3.6)**, p3(0.0)]	{ {p3,**p2**},{p1},{} }	**t345** - hoverfor	hoverfor,1.0

Table 7.1 Cognitive reasoning results for use case 3 - continued						
Cues		**Activated Memory**			**Focus of Attention**	**Action Selection**
No	**Cue**	**Optimized LTM (Swarm)**	**Individual Utility**	**Coalition Structure**	**Asset**	**Command**
24	land	p1{ **t240(2.0)**, t230(1.08), t215(1.0), t235(1.5) } p2{ t640(2.0) } p3{ t220(1.08), t225(1.08) }	[**p1(5.6)**, p2(2.0), p3(2.2)]	{ {p3,**p1**},{p2},{} }	**t240** - land	land
25	disarm	p1{ t640(2.0) } p2{ **t525(2.0)** } p3{ }	[p1(2.0), **p2(2.0)**, p3(0.0)]	{ {p3,**p2**},{p1},{} }	**t525** - disarm	disarm

Table 7.2 shows the cognitive process time (CPt) and stimulus process time (TPt), for use case 3.

Table 7.2 Processing and execution time for use case 3.
(Cues relevant to function 1 are shown in blue and cues relevant to function 2 are shown in red)

Cue	CPt (Secs)	TPt (secs)	Cue	CPt (Secs)	TPt (secs)
1	0.271	05.613	14	0.617	06.137
2	0.359	17.939	15	1.245	21.766
3	0.226	06.741	16	0.432	07.121
4	1.081	21.491	17	1.221	09.165
5	0.545	06.096	18	1.258	09.125
6	1.151	21.550	19	1.216	07.995
7	0.444	07.029	20	1.204	21.833
8	1.220	09.135	21	0.599	06.122
9	1.461	08.367	22	1.252	45.963
10	1.225	08.010	23	0.620	07.330
11	1.222	22.929	24	1.269	19.543
12	0.617	08.457	25	0.281	09.044
13	1.255	48.379			
		Average:		**0.892**	**14.515**

Function 1 = Flight Control; Function 2 = Gripper Control

Figure 7.16 shows the state flow generated for use case 3. The state flow represents the set of actions, selected by the central executive, and sent as commands to the UAV controller, for execution. Each transition (edge) between the states, identifies the cue number and the resulting command, defined by the state transition (see eq. (4.3) in section 4.1.2).

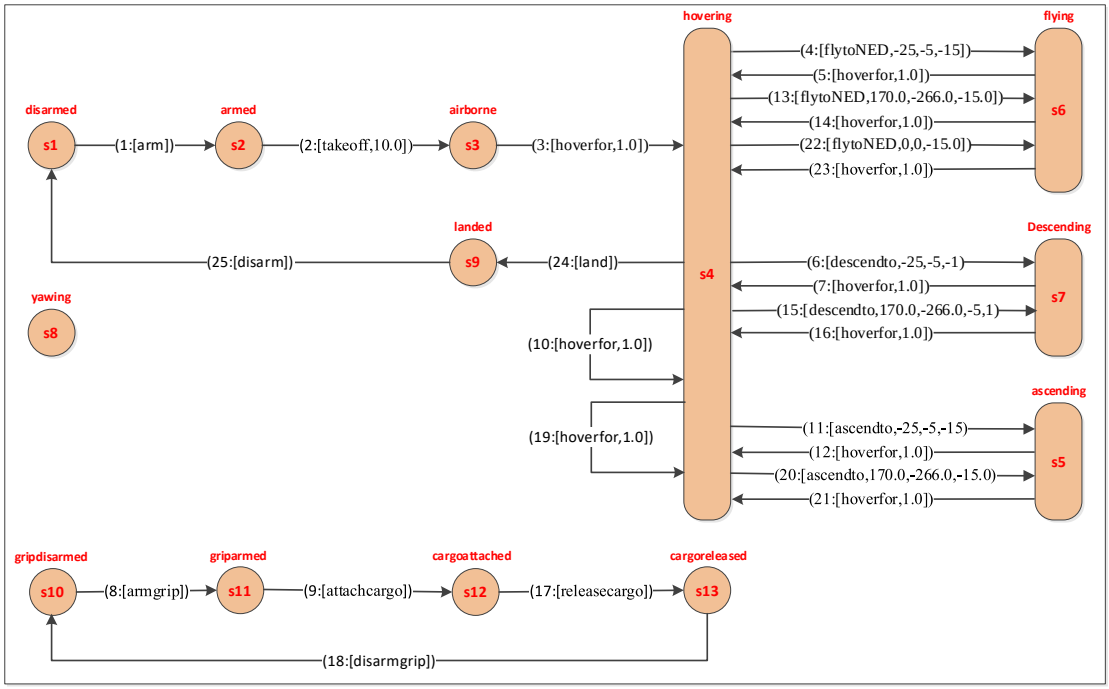

Figure 7.16 Resulting state flow constructed for use case 3.
The diagram shows the applicable states and state transitions, with each transition labelled with the task number, relevant command (action) and state transition fitness.

The two independent functions, applicable to the mission, can be seen on the state flow. The state "yawing" is not applicable to either function, and therefore (correctly) remains unconnected.

Use case 4 results (Aerial Surveillance)

Table 7.3 shows the results of the simulation for use case 4 – providing security surveillance support at an incident. For this simulation only one UAV function, flight control, is applicable.

Table 7.3 Cognitive reasoning results for use case 4.
For each cue (task), the results of the cognitive process (memory recall, activated memory, focus of attention and action selection) are shown. (Cues relevant to function 1 are shown in blue – there are no cues relevant to function 2. The asset and particle, relevant to the cue, is highlighted in **bold**) (Each "p" represents a specific particle and each "t" represents an asset (memory item) of the particle)

Cues		**Activated Memory**			**Focus of Attention**	**Action Selection**
No	**Cue**	**Optimized LTM (Swarm)**	**Individual Utility**	**Coalition Structure**	**Asset**	**Command**
1	arm	p1{ } p2{ t640(2.0) } p3{ **t10(2.0)** }	[p1(0.0), p2(2.0), **p3(2.0)**]	{ {p2}, {p1,**p3**}, {} }	**t10**-arm	arm
2	takeoff	p1{ t70(2.0), **t80(2.0)** } p2{ t640(2.0) } p3{ }	[**p1(4.0)**, p2(2.0), p3(0.0)]	{ {p3,**p1**}, {p2},{} }	**t80**-takeoff	takeoff,10.0
3	hoverfor	p1{ **t150(1.0)**} p2{ t640(2.0) } p3{ }	[**p1(1.0)**, p2(2.0), p3(0.0)]	{ {p2,p3,**p1**}, {}, {} }	**t150**-hoverfor	hoverfor,1.0
4	flytoNED	p1{ t235(1.5), t220(1.33) } p2{ t640(2.0) } p3{ **t225(1.33)**, t230(1.33), t215(1.0), t240(2.0) }	[p1(2.8), p2(2.0), **p3(5.7)**]	{ {**p3**,p1}, {p2}, {} }	**t225**-flytoNED	flytoNED,-225,190,-10
5	hoverfor	p1{ t640(2.0)} p2{ t365(1.5), t355(1.08), **t345(1.0)** } p3{ }	[p1(2.0), **p2(3.6)**, p3(0.0)]	{ {p3,**p2**}, {p1}, {} }	**t345**-hoverfor	hoverfor,1.0
6	flytoNED	p1{ t640(2.0)} p2{ t230(1.33), t215(1.0), t235(1.5) } p3{ **t225(1.33)**, t240(2.0), t220(1.33) }	[p1(2.0), p2(3.8), **p3(4.7)**]	{ {**p3**,p2}, {p1}, {} }	**t225**-flytoNED	flytoNED,-320,215,-10
7	hoverfor	p1{ **t345(1.0)**, t365(1.5), t355(1.08) } p2{ } p3{ t640(2.0) }	[**p1(3.6)**, p2(0.0), p3(2.0)]	{ {p2,**p1**}, {p3}, {} }	**t345**-hoverfor	hoverfor,1.0
8	flytoNED	p1{ t640(2.0)} p2{ **t225(1.33)**, t220(1.33) } p3{t235(1.5), t240(2.0), t215(1.0), t230(1.33) }	[p1(2.0), **p2(2.7)**, p3(5.8)]	{ {p3,**p2**}, {p1}, {} }	**t225**-flytoNED	flytoNED,-210,260,-10
9	hoverfor	p1{ t640(2.0) } p2{ **t345(1.0)** } p3{ t365(1.5), t355(1.08) }	[p1(2.0), **p2(1.0)**, p3(2.6)]	{ {p1}, {}, {p3,**p2**} }	**t345**-hoverfor	hoverfor,1.0

Table 7.3 Cognitive reasoning results for use case 4 – continued

Cues		Activated Memory			Focus of Attention	Action Selection
No	**Cue**	**Optimized LTM (Swarm)**	**Individual Utility**	**Coalition Structure**	**Asset**	**Command**
10	flytoNED	p1{ t235(1.5), **t225(1.33)**} p2{t215(1.0), t220(1.33), t240(2.0), t230(1.33)} p3{ t640(2.0) }	[**p1(2.8)**, p2(5.7), p3(2.0)]	{ {p3}, {**p1**,p2}, {} }	**t225**-flytoNED	flytoNED,-225,190,-10
11	hoverfor	p1{ t640(2.0) } p2{ **t345(1.0)**, t365(1.5), t355(1.08) } p3{ }	[p1(2.0), **p2(3.6)**, p3(0.0)]	{ {**p2**,p1,p3}, {}, {} }	**t345**-hoverfor	hoverfor,1.0
12	flytoNED	p1{ t240(2.0), **t225(1.33)**, t215(1.0) } p2{ t235(1.5), t230(1.33), t220(1.33) } p3{ t640(2.0) }	[**p1(4.3)**, p2(4.2), p3(2.0)]	{ {p3}, {}, {**p1**,p2} }	**t225**-flytoNED	flytoNED,0,0,-10,1.33
13	hoverfor	p1{ t355(1.08) } p2{ **t345(1)**, t365(1.5) } p3{ t640(2.0) }	[p1(1.1), **p2(2.5)**, p3(2.0)]	{ {p1,**p2**}, {p3}, {} }	**t345**-hoverfor	hoverfor,1.0
14	land	p1{ t640(2.0) } p2{ t215(1.0), t230(1.08), t235(1.5), t225(1.08), t220(1.08), **t240(2.0)** } p3{ }	[p1(2.0), **p2(7.7)**, p3(0.0)]	{ {p3, **p2**}, {p1}, {} }	**t240**-land	land
15	disarm	p1{ **t525(2.0)** } p2{ t640(2.0) } p3{ }	[**p1(2.0)**, p2(2.0), p3(0.0)]	{ {p2,p3,**p1**}, {}, {} }	**t525**-disarm	disarm

Table 7.4 shows the cognitive process time (CPt) and stimulus process time (TPt), for use case 4.

Table 7.4 Processing and execution time for use case 4.
(Cues relevant to function 1 are shown in blue)

No	CPt (Secs)	TPt (secs)	No	CPt (Secs)	TPt (secs)
1	0.266	5.575	9	0.640	7.308
2	0.373	16.735	10	1.268	20.659
3	0.257	7.805	11	0.641	6.212
4	1.114	47.557	12	1.210	46.919
5	0.580	7.251	13	0.619	6.124
6	1.173	22.671	14	1.217	19.469
7	0.615	7.286	15	0.278	9.118
8	1.252	24.776			
		Average:		**0.767**	**17.031**

Function 1 = Flight Control; Function 2 = Gripper Control

Figure 7.17 shows the state flow generated for use case 4. The state flow represents the set of actions, selected by the central executive, and sent as commands to the UAV controller, for execution. Each transition (edge) between the states identifies the cue number and the resulting command, as defined by the state transition (see eq. (4.3) in section 4.1.2).

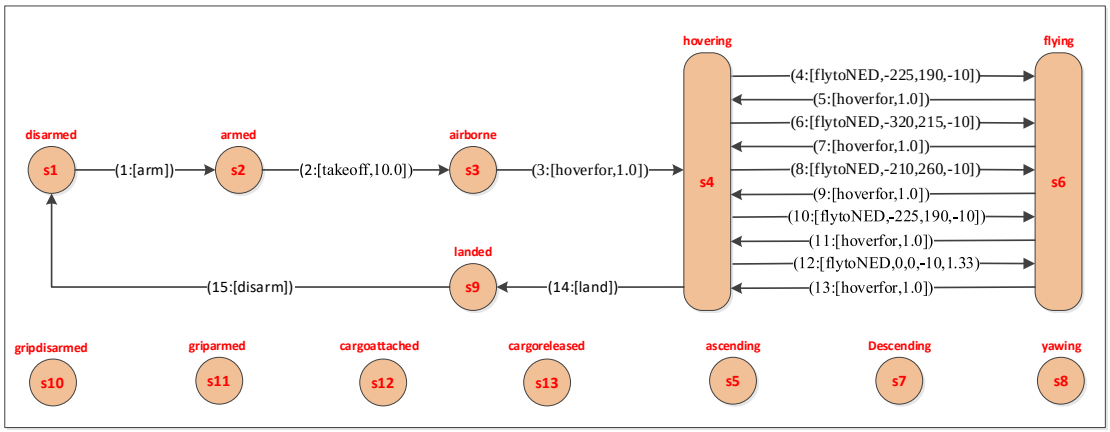

Figure 7.17 Resulting state flow constructed for use case 4.
The diagram shows the applicable states and state transitions, with each transition labelled with the task number, relevant command (action) and state transition fitness.

7.2.5 Discussion

The results of use case 3 and use case 4, are shown in tables 7.1 - 7.2 and 7.3 – 7.4, respectively. Referring to table 7.1, column 7 shows the action (command) selected and executed for each stimulus received. In column 3, the optimized long-term memory is represented by a swarm of

particles, p1, p2 and p3, including the set of quantified assets, assigned to each particle. For example, for cue 2 (takeoff), particle 2 is represented by p2{ t70(2.0), t80(2.0) }, where p2 is the particle identifier, { t70(2.0), t80(2.0) } is the set of assets, represented by t70(2.0), t80(2.0), where t70 and t80 are the asset identifiers, each quantified with a real value of (2.0). In column 4, the individual utility of each particle (in the swarm) is shown as the rounded sum of its assets. The coalition structure resulting from algorithms 6.2 and 6.3, is shown in column 5. For example, for cue 2 (takeoff), the coalition structure is represented by {{p3,p2},{p1},{}}, where {p3,p2},{p1} and {} are the coalitions 1, 2 and 3, respectively. Coalition 1 has two members, p3 and p2, coalition 2 has one member, p1 and coalition 3 is empty. The FOA, shown in column 6, is derived from the coalition structure and is the optimal asset, given the cue received. From the FOA, the central executive selects the corresponding action/s and passes it/them as a command/s (with arguments, if applicable) to the UAV controller. The commands passed to the UAV controller are shown in column 7. The results of use case 4 are represented in the same way in table 7.3.

Correctness

The state flow, shown in figure 7.16, shows that the robo-cognitive architecture correctly executed use case 1 and that the correct transition was learned for each cue received. Figure 7.16 shows that the correct states were connected for function 1 (flight control) and function 2 (gripper control), required by the mission. It is also shown that there were no incorrect or redundant transitions. The lack of any transitions between any of the states of function 1 and states for function 2, shows that independence between the functions was correctly maintained by the robo-cognitive architecture. This shows that the robo-cognitive architecture enables the UAV to perform multiple independent tasks.

(Note that, for the two simulations, the robo-cognitive architecture was executed using a sequential processing computational architecture. However, minor changes to the robo-cognitive architecture, in favour of parallel computing, will enable the parallel execution of the commands for independent functions.)

The results in table 7.1 also show that the robo-cognitive architecture reasoned correctly for each of the 25 cues of the mission. Inspecting table 7.1 shows that, for each cue in column 2, the correct corresponding asset (state transition) received the focus of attention (column 6). This allowed the central executive to select the correct action for execution (column 7).

For example, after descending, the robo-cognitive architecture receives cue number 16 (hoverfor). The asset t410, (which represent the transition between states s7 and s4), correctly

received the focus of attention which enabled the central executive to send the command "hoverfor,1.0" to the UAV controller. The robo-cognitive architecture then receives cue 17 (releasecargo) and the asset t780 ((which represent the transition between states s12 and s13), correctly received the focus of attention which enabled the central executive to send the command "releasecargo" to the UAV control program. The results in table 7.1 show that all 25 cues were processed, without repetition or redundancy and figure 7.16 shows that the 25 cues were processed in the correct order. It is therefore concluded that use case 3 executed completely and correctly.

The learned state flow (figure 7.17) shows that the robo-cognitive architecture correctly executed use case 4, and that the correct transition was learned for each cue received. Figure 7.17 shows that the correct states were connected for function 1 (flight control) and that there were no incorrect or redundant transitions. This is evident in figure 7.17, which shows no transitions were learned for any of the states for function 2 (gripper control). The states for function 2 remain unconnected.

The results in table 7.3 also show that the robo-cognitive architecture reasoned correctly for each of the 15 cues of the mission. Inspecting table 7.3 shows that, for each cue in column 2, the correct corresponding asset (state transition) received the focus of attention (column 6). This allowed the central executive to select the correct action for execution (column 7). For example, for cue number 2 (takeoff), the asset t80, (which represent the transition between states s2 and s3), correctly received the focus of attention which enabled the central executive to send the command "takeoff,10.0" to the UAV control program. The results in table 7.3 show that all 15 cues were processed, without repetition or redundancy and figure 7.17 shows that the 15 cues were processed in the correct sequence. It is therefore concluded that use case 4 also executed completely and correctly.

Time Efficiency

The time efficiency of use case 3 is evaluated by inspecting table 7.2, and presented in the graph in figure 7.18.

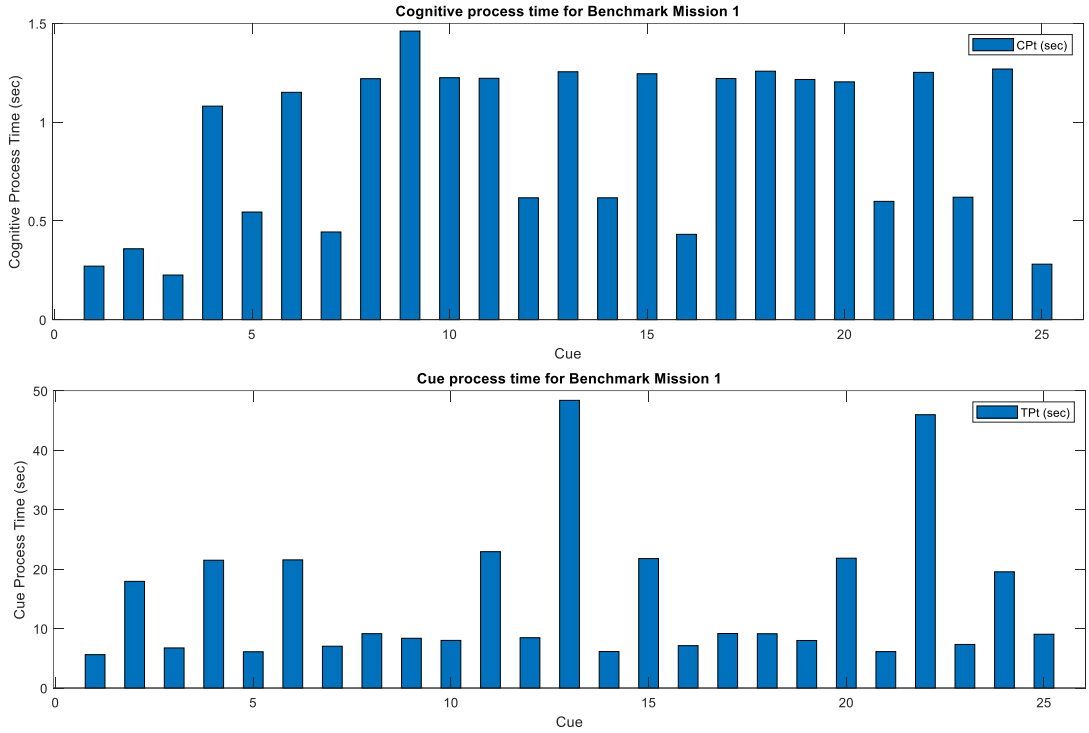

Figure 7.18 Cognitive reasoning process (CRP) time for use case 3.
The cognitive processing time (CPt) is shown on top and the cue processing time (TPt) is shown below.

Figure 7.18 shows the measured processing time for both the CPt (top) and the TPt (bottom) for each cue received and processed by the robo-cognitive architecture. Table 7.2 and the CPt graph, figure 7.18, show that the average cognitive processing time for the whole mission is 0.892s. The lowest time recorded is 0.226s (cue 3 – "hoverfor") and the highest time recorded is 1.461s (cue 9 – "attachcargo"). It is worth noting the difference in the CPt for the cognitive processing of long-term memory items with different trigger functions. For example, the CPt for cue 3 ("hoverfor") is 0.226s, while the CPt for cue 4 ("flytoNED") is 1.081s. The reason for the difference is due to the complexity of the constraint average calculation in the quantification algorithm. For example, quantifying the "hoverfor" asset involves a constraint average calculation based on a single, real parameter (i.e. hover time), while the quantification of the "flytoNED" asset involves a constraint average calculation based on spatial parameters (i.e. x, y, z). It should also be noted that the difference applies to each iteration of the CG-PSO and cumulatively impacts the overall CPt. However, it is clear that, overall, the CPt is still well below the MPt, of 2s.

The TPt results shown in table 7.2 and figure 7.18, show the total time it took to complete the whole mission was approximately 6 mins, which is deemed acceptable for the type of mission.

The shortest task took 5.613s and the longest task took 48.379s. The TPt for cues 13 and 22 show the time it took for the UAV to fly to and back from the target destination. It is concluded that use case 4 was successfully completed within an acceptable time.

The time efficiency of use case 4 is evaluated by inspecting the CPt and TPt values in table 7.4, which is represented in the graph in figure 7.19.

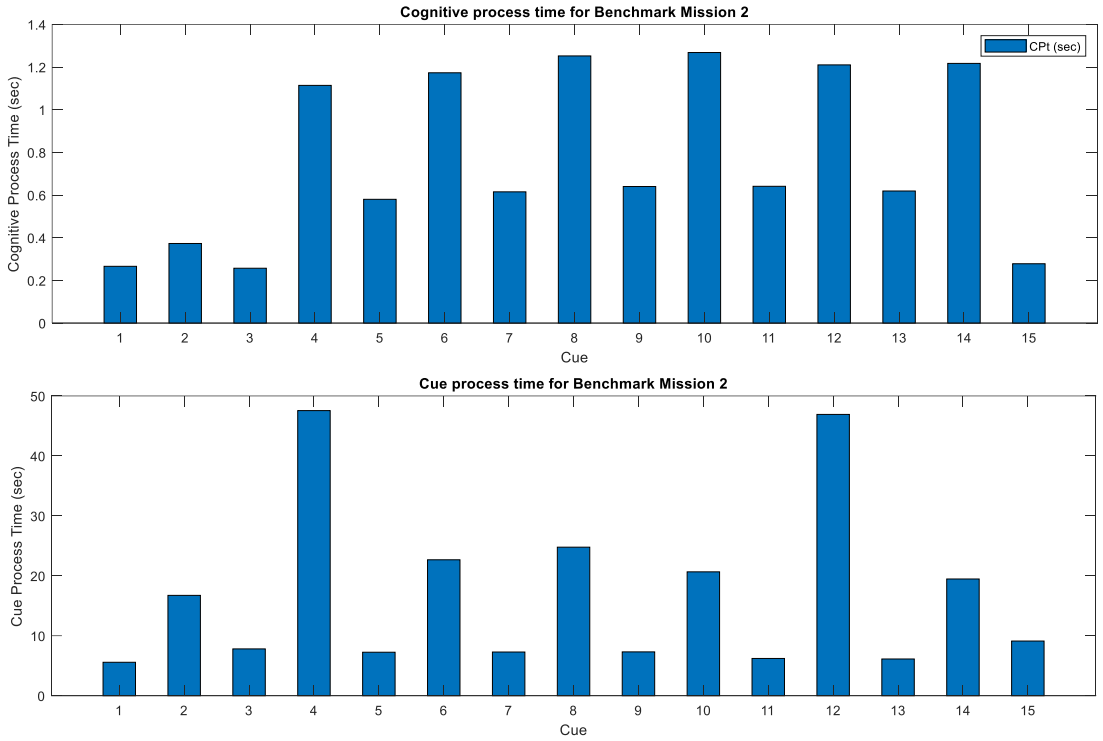

Figure 7.19 Cognitive reasoning process (CRP) time for use case 4.
The cognitive processing time (CPt) is shown on top and the cue processing time (TPt) is shown below.

Figure 7.19 shows the measured processing time for both the CPt (top) and the TPt (bottom) for each cue received and processed by the robo-cognitive architecture. Table 7.4 and the CPt graph shows that the average cognitive processing time for the whole mission is 0.767s. The lowest time recorded is 0.257s (cue 3 – "hoverfor") and the highest time recorded is 1.268s (cue 10 – "flytoNED"). Again, it is worth noting the difference in the CPt for the cognitive processing of long-term memory items with different trigger functions. For example, the CPt for cue 3 ("hoverfor") is 0.257s, while the CPt for cue 4 ("flytoNED") is 1.114s. The reason for the difference is due to the complexity of the constraint average calculation in the quantification algorithm. For example, quantifying the "hoverfor" asset involves a constraint average calculation based on a single, real parameter (i.e. hover time), while the quantification of the

“flytoNED” asset involves a constraint average calculation based on spatial parameters (i.e. x, y, z). It should also be noted that the difference applies to each iteration of the CG-PSO and cumulatively impacts the overall CPt. However, it is still clear that, overall, the CPt is well below the MPt, of 2s.

The TPt results shown in table 7.4 and figure 7.19 show the total time it took to complete the whole mission was approximately 4 mins, which is deemed acceptable for the type of mission. The shortest task took 5.575s and the longest task took 47.557s. The TPt for cues 4 and 12 show the time it took for the UAV to fly to, and back from, the target destination. It is concluded that the use case 4 was successfully completed within an acceptable time.

Note that, explicit time delays were set in the UAV control program, in order to give the UAV sufficient time to complete a task. For example, for the “flytoNED” command, a delay of 35s was set to give the UAV to get to and from its destination. Therefore, it does not make sense to evaluate the TPt against the MPt. However, from a practical point of view, the TPt still provides useful information regarding the task execution and mission execution times.

The approaches followed in the associated research, discussed in chapter 2, varies greatly in architectural design, scientific and technological approach, size and complexity. It is therefore impossible to do a fair empirical comparison between these approaches and the methodologies developed and tested in this chapter. The differences are too significant and mostly unquantifiable. Below, the benefits of the proposed robo-cognitive architecture are contrasted with the approaches reviewed in chapter 2.

1. Knowledge is represented using a simple propositional logic structure, which is more intuitive for a domain expert to maintain and therefore less error-prone than modal logic, linear temporal logic, first-order logic or Horn clause formats. While the propositional representation is not as rich as the other representations, it is sufficient for real-time, high-level control of an autonomous vehicle. The approach is discussed in detail in section 4.1.2.

2. The simplified knowledge structure simplifies modification and augmentation of the knowledge through simple propositional memory item updates. This means less computational resources are required, since there are no complex modal logic, linear temporal logic, first-order logic or Horn clause memory item resolutions required. This also means that modifications or augmentation may be applied on a proposition level, which is more intuitive and requires less communication bandwidth.

3. The cognitive reasoning process uses knowledge quantification, based on the long-term memory and real-time environmental stimuli, for inference in high-level autonomous vehicle control. No subjective probabilities, biases or parameters (such as those used in machine learning) are applied to the quantification process. This also means that the robo-cognitive architecture is more robust to changes in the environment.

4. The flexibility and adaptiveness of the knowledge representation and quantification process mean the robo-cognitive architecture enables autonomous vehicle control, based on real-time environmental stimuli. No time-consuming and expensive re-learning of models or algorithm changes are required when knowledge or environmental stimuli changes.

5. Statistical quantification of a knowledge item produces a probability distribution over all the propositions (rules) of the memory item. This allows inference to follow the open world assumption (see section 3.2.3), which allows finer-grained statistical reasoning to be applied in the inference. This has a major benefit when handling uncertainty in the inference. This approach is demonstrated in velocity control result of use cases 1 and 2 and shown in figures 7.5 and 7.9.

6. The robo-cognitive architecture has a relatively simple architecture, as it focusses on the working memory and central executive only and does not attempt to completely emulate all the neuro-cognitive processes. This greatly reduces the complexity of the architecture, but is modular enough to extend with further functionality (e.g. a learning module, see future work in chapter 8), if required.

7. The adaptiveness of the robo-cognitive architecture enables automatic identification and parallel execution of multiple functions on the autonomous vehicle, according to its mission. Because no complex and computationally expensive bespoke procedural processes are used, extending the functionality of the autonomous vehicle is simplified. No rebuilding, retesting and redeployment of components are required.

7.3 Conclusion

The methodology, developed in section 6.1, constructs optimal episodic memory, based on Baddeley's model of working memory, for the real-time, high-level control of an autonomous vehicle. The methodology was evaluated by executing two simulated missions, developed in section 7.1. The results of the simulated missions show that the robo-cognitive architecture

successfully and completely executed each mission by repeatedly finding and executing the optimal task for the mission, given the state of the unmanned aerial vehicle.

The methodology, developed in section 6.2, constructs a focus of attention from activated memory, based on Cowan's attentional focus model of working memory, for the real-time, high-level control of an autonomous vehicle. The methodology was evaluated by executing two simulated missions, developed in section 7.2. The results of the simulated missions show that the robo-cognitive architecture successfully and completely executed each mission by repeatedly finding and executing all the optimal set of tasks for the mission, given the state of the unmanned aerial vehicle.

It is concluded that the two robo-cognitive architectures, based on working memory optimization, is suitable for real-time, high-level cognitive control of a multi-functioned, autonomous vehicle.

Chapter 8

Conclusions and future work

8.1 Conclusions

In chapter 1, the problem of real-time, high-level control of robots, deployed in dynamic environments, was highlighted. A robo-cognitive architecture, providing the main cognitive processes for memory representation, memory recall, action selection and action execution, is proposed. The cognitive approach raises the following research questions:

(1) Can working memory be represented in a form which simplifies augmentation and modification by domain expert?
(2) Can working memory be statistically quantified for the evaluation of optimality, during memory recall in cognitive reasoning?
(3) Is the cognitive reasoning capable of correct action selection for both single- and multiple, independent (or parallel) tasks?

The main aim of this thesis is to address these questions by designing, developing and evaluating the main cognitive functions in a robo-cognitive architecture. While addressing these questions, the following contributions, as listed in section 1.3, were made:

1. The creation of a novel working memory representation, structured to simplify modification and augmentation.
2. The design and development of a novel adaptive entropy fitness quantification (AEFQ) algorithm for the statistical quantification of discrete memory items (knowledge).
3. The design and development of a cognitive reasoning process for memory recall, using an improved set-based particle swarm optimization algorithm (using the AEFQ quantification) for action selection for single task execution.
4. The design and development of a cognitive reasoning process for memory recall, using a novel CG-PSO algorithm (and AEFQ quantification) for multiple action selection for multiple, parallel task execution.
5. Confirmation of the suitability of the robo-cognitive architectures in the execution of four use cases in simulation.

Each research question is discussed below:

8.1.1 Research question (1)

The importance of memory representation was discussed in section 1.1 and popular knowledge definitions and structures were reviewed in sections 2.1 and 3.2. The reviews showed that knowledge may be defined by a domain expert or it may be learned programmatically. It is also shown that a popular approach is to structure the knowledge in some probabilistic graphical model or network structure for inference. An overview of some popular formats for knowledge representation, for example, first-order logic and Horn clauses, are given in section 3.2. Due to the complexity and lack of intuitiveness of these representations, a simplified memory representation structure, is developed in section 4.1.2. The novel memory representation represents the long-term memory and is used in the memory quantification function developed for memory recall. Since the memory items in the long-term memory are structured as simple conjunctive and/or disjunctive propositional logic terms, maintenance is more cost-effective in computation and bandwidth. The simplicity of the representation makes the control logic of memory items also more intuitive and therefore less error-prone. As shown in the simulations, the validity indicator in eq. (4.3) simplifies the activation of memory items. Only active state transitions are (correctly) included in the resulting state flows of the four use cases, shown in figures 7.4, 7.8, 7.16 and 7.17. The propositional logic form of the formulae of the memory items allows the domain expert to focus on the atomic facts about each memory item, while the relations between these facts are simplified through the use of conjunctions and disjunctions only. This eliminates the complexity of existential relations, such as those used in modal logic or first-order logic. Modifications and augmentations can focus on atomic facts and relations. The fully connected long-term memory, therefore, provides a memory representation which is more economical, simpler to maintain and is less error-prone. The memory representation developed and applied in the use case simulations, confirms contribution 1, described above.

8.1.2 Research question (2)

The two main functions of memory recall are memory quantification and optimization. There must be a sufficient belief in a memory item before it will be applied in the decision-making process. This degree of belief (fitness) is defined as a probability. Given the probability assigned to a memory item, it could be quantified using information entropy, discussed in section 3.3. Information entropy (also called information gain) is a means of quantifying the level of uncertainty in a memory item, given its probability. However, the probability of a memory item is not known a-priori during memory recall. Therefore, the novel AEFQ algorithm, developed in section 4.2, assigns probabilities to a memory item before the fitness evaluation of the memory item. The AEFQ algorithm uses the maximum entropy principle to derive a probability distribution over the propositional terms of the e memory item. The

maximum entropy principle is the maximization of the information entropy, given the environmental stimuli. Therefore, a memory item is quantified, not only in terms of its logic formula, but in terms of each of its facts (propositional logic terms). This means fitness evaluation can be performed on a finer level, using probability calculus, including conditional and marginal probabilities. Moreover, the probabilities assigned are "pure". This means no subjective values were introduced, instead, only real-time environmental stimuli are used in the calculation. The results of the simulations (sections 7.1 and 7.2) show that the correct memory items (state transitions) were selected for action selection and execution. This means only memory items with positive fitness were selected. Each edge of the state flows in figures 7.4 and 7.8 shows the correct memory item and corresponding fitness value. Figures 7.5 and figure 7.9 shows the fine-grained control of vehicle velocity, controlled by the fitness quantification. The figures show the decreasing fitness, as the unmanned aerial vehicle approaches its destination. Moreover, figures 7.13, 7.18 and 7.19 show that all use cases were executed within the specified time window. Tables 7.1 and 7.3 shows the activated memory, along with fitness (utility) for each memory item. In addition, figures 7.16 and 7.17 show the correct and complete state flows for use cases 3 and 4. Each edge of the state flow also shows the correct action selected. All four use cases were successfully, timeously and completely executed, as demonstrated in the videos. The successful completion of the use cases confirms that memory quantification and optimization, in memory recall, were successfully performed. The success of the AEFQ algorithm, used in the set-based PSO and coalitional game-theoretic PSO algorithms during memory recall, confirms contribution 2, as described above.

8.1.3 Research question (3)

In chapter 6, two methodologies were developed for the real-time, high-level cognitive control of robots:

- Real-time episodic memory construction for single-task cognitive control in robotics (section 6.1)
- A cognitive architecture using coalitional games-theoretic particle swarm optimization for real-time, multi-task control in cognitive robotics (section 6.2)

In section 6.1, an SPSO and AEFQ methodologies was developed for the real-time construction of episodic memory. The central executive selected and executed an action, after selecting an optimal memory item from episodic memory. The approach is based on the Baddeley working memory model (section 3.1.2), which represents working memory as episodic memory. The results of the simulations (section 7.1) show that the correct memory items (state transitions) were selected for action selection and execution. Figure 7.4 and figure 7.8 shows the correct and complete state flows for the simulations. Each edge of the state flow also shows the correct

fitness. The simulations show that each task of the mission was correctly selected, one by one, and executed from episodic memory. Both use cases were successfully, timeously and completely executed, as demonstrated in the videos.

In section 6.2, a CG-PSO methodology was developed for the real-time construction of activated memory and focus of attention, from which the central executive selects the appropriate action for execution. The approach is based on Cowan's attentional focus working memory model (section 3.1.2), which represent working memory as activated memory with a focus of attention. The results of the simulations (section 7.2) show that the correct memory items (state transitions) were selected for action selection and execution. Tables 7.1 and 7.3 shows the activated memory, along with fitness (utility) for each memory item. Figure 7.16 and figure 7.17 show the correct and complete state flows of the simulations, including the correct action selected from the focus of attention. Figure 7.16 shows two correct, independent state flows, one for flight-control and one for gripper control, while figure 7.17 correctly shows only the one state flow, for flight-control. Moreover, figure 7.18 and figure 7.19 shows that the use cases were executed within the specified time window. Both use cases were successfully, timeously and completely executed, as demonstrated in the videos.

The success of the set-based particle swarm optimization and coalitional game-theoretic particle swarm optimization algorithms during memory recall, confirms contributions 3 and 4.

The successful completion of all four use cases shows that the cognitive reasoning of the cognitive architecture, proposed in this research study, can successfully select and execute single tasks sequentially. The results also show that multiple tasks are correctly selected and executed (in parallel, if the architecture allows). The successful completion of all four use cases confirm contribution 5.

8.2 Future work

In this research study, the problem of real-time, high-level control of robots, in remote and dynamic environments, were investigated. The focus was on effective cognitive processing of working memory pertaining to memory representation, memory recall, action selection and action execution. This study assumes knowledge is provided and maintained by a domain expert, while the cognitive reasoning is left to the autonomous vehicle. During the literature review and the development and testing of the robo-cognitive architecture, two specific areas were identified for further research. These areas are described below.

8.2.1 Reasoning

The robo-cognitive architecture developed in this research study quantifies a memory item, representing a state transition, by calculating a probability distribution over the propositional terms of the trigger formula of the state transition. To simplify the introduction of the methodology, only one of the probabilities were used for fitness evaluation. However, this approach lays, without loss of generality, the foundation for more extensive reasoning in the robo-cognitive architecture. An adaptive, cognitive reasoner paradigm, using, the probability quantification, will extend the applicability and power of the robo-cognitive architecture significantly. The results of the adaptive cognitive reasoner will support decision-making, and enable tractability and explanation of the reasoning and decision-making. Explanation of inference is an important requirement in artificial intelligence solutions. The adaptiveness of the paradigm will allow the reasoning to take new knowledge acquired into account. This approach will improve the real-time reactiveness to changing knowledge or a changing environment. An adaptive cognitive reasoner, used in conjunction with a learning paradigm (section 8.2.2), will also improve the autonomy of an autonomous vehicle. For example, the autonomous vehicle could decide if and when to report its findings, instead of being requested or instructed. An example of such an application is the exploration of the Mars autonomous vehicle, Curiosity.

8.2.2 Learning

The approach in this research study assumes knowledge (memory) is initially defined and provided by a domain expert. The robo-cognitive architecture focusses on the simplification of the representation of the memory, as well as the cognitive processing of this memory. An unsupervised learning paradigm, will extend the autonomy of an autonomous vehicle using the robo-cognitive architecture, significantly. However, the argument is maintained that knowledge cannot be learned from nothing, especially in remotely deployed robots. However, it is argued that the simplified memory representation presented in this research study lays the foundation for an effective unsupervised learning paradigm. This learning paradigm could build on and extend the initial knowledge provided by the domain expert. Problems may be solved autonomously, within the functional capabilities of the autonomous vehicle, by employing the adaptive, cognitive reasoner (recommended in section 8.2.1) and augmenting the long-term memory and the short-term memory. The quantification method of the AEFQ algorithm may be extended to take the biases and weights, resulting from the learning paradigm, into account during the reasoning process. This extension will drive the reasoning, and therefore, the decision-making process. This approach will also assist the domain expert in the definition of

initial knowledge, and the efficacy of the initial knowledge may be tested and evaluated using experimentation and simulation in a laboratory.

8.2.3 Collaboration

The robo-cognitive architecture developed in this research study, provides a cognitive approach towards real-time, high-level control of a multi-function autonomous vehicle. Some tasks, however, require more than one autonomous vehicle. For example, in the simulations in section 7.1 and 7.2, the unmanned aerial vehicle delivers a medical package to a target destination. There may be situations where the package requires two (or more) vehicles to work together in order to transport the cargo successfully. For autonomous vehicles to successfully cooperate in a group, there needs to be robust, real-time collaboration between them. By extending the robo-cognitive architecture with a collaboration function, the autonomous vehicle will be able to operate in a group of autonomous vehicles. Memory recall, in the robo-cognitive architecture, involves memory quantification, by the AEFQ algorithm, developed in section 4.2. The algorithm uses information from the environmental stimuli and short-term memory, to find the optimal memory in the long-term memory. By including collaboration information in the environmental stimuli and short-term memory, memory quantification and memory optimization will take the collaboration information into account when finding appropriate memory items from the long-term memory. The collaboration function has to update the environmental stimuli and short-term memory in real-time, and this will enable the central executive to select and execute actions, appropriate to the state of all autonomous vehicles in the group. There are many technical solutions available for sharing of information between processes. However, since the high-level control affects the behaviour of the autonomous vehicle in real-time, a time-efficient communication protocol must be applied. By extending the robo-cognitive architecture, with a collaboration component, autonomous vehicles may be deployed in team (multi-agent) or swarm configurations.

Bibliography

[1] Y. Ma, Z. Wang, H. Yang, and L. Yang, "Artificial intelligence applications in the development of autonomous vehicles: a survey," *IEEE/CAA Journal of Automatica Sinica*, vol. 7, no. 2, pp. 315-329, 2020, doi: 10.1109/JAS.2020.1003021.

[2] A. Chamuah and R. Singh, "Securing sustainability in Indian agriculture through civilian UAV: a responsible innovation perspective," *SN Applied Sciences*, vol. 2, no. 1, p. 106, 2019/12/18 2019, doi: 10.1007/s42452-019-1901-6.

[3] S. Avasker *et al.*, "A method for stabilization of drone flight controlled by autopilot with time delay," *SN Applied Sciences*, vol. 2, no. 2, p. 225, 2020/01/17 2020, doi: 10.1007/s42452-020-1962-6.

[4] A. Martinez Alvarez and C. A. Lozano Espinosa, "Nonlinear control for collision-free navigation of UAV fleet," *SN Applied Sciences*, vol. 1, no. 12, p. 1577, 2019/11/08 2019, doi: 10.1007/s42452-019-1606-x.

[5] R. Almadhoun, T. Taha, L. Seneviratne, and Y. Zweiri, "A survey on multi-robot coverage path planning for model reconstruction and mapping," *SN Applied Sciences*, vol. 1, no. 8, p. 847, 2019/07/10 2019, doi: 10.1007/s42452-019-0872-y.

[6] M. Biba, *Integrating Logic and Statistics - Novel Algorithms in Markov Logic Networks*. VDM Verlag Dr Muller Aktiengesellschaft & Co. KG, 2009.

[7] B. Wilcox *et al.*, "Robotic vehicles for planetary exploration," in *Proceedings 1992 IEEE International Conference on Robotics and Automation*, 12-14 May 1992 1992, pp. 175-180 vol.1, doi: 10.1109/robot.1992.220266.

[8] R. Francis *et al.*, "AEGIS autonomous targeting for the Curiosity rover's ChemCam instrument," in *2015 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*, 13-15 Oct. 2015 2015, pp. 1-5, doi: 10.1109/aipr.2015.7444544.

[9] D. Vernon, "Two Ways (Not) To Design a Cognitive Architecture," in *Cognitive Robot Architectures*, Vienna, Austria, December 8-9, 2016 2016, no. Proceedings of EUCognition 2016.

[10] P. Singh, "Contrasting Embodied Cognition with Standard Cognitive Science: A Perspective on Mental Representation," *Journal of Indian Council of Philosophical Research*, journal article vol. 36, no. 1, pp. 125-149, January 01 2019, doi: 10.1007/s40961-018-0159-5.

[11] S. Lewandowsky and S. Farrell, *Computational Modeling in Cognition: Principles and Practice*. Sage Publications Inc., 2011, p. 357.

[12] J. R. Anderson, *The Architecture of Cognition*. Harvard University Press, 1983, p. 345.

[13] J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, and Y. Qin, "An integrated theory of the mind," *Psychological Review*, vol. 111, 4, pp. 1036-1060, 2004.

[14] J. E. Laird, *The SOAR Cognitive Architecture*. The MIT Press, 2012.

[15] C. Eliasmith, *How to Build a Brain* (Oxford Series on Cognitive Models and Architecture). United States: Oxford University Press, 2013, p. 456.

[16] A. V. Samsonovich, "Socially emotional brain-inspired cognitive architecture framework for artificial intelligence," *Cognitive Systems Research*, vol. 60, pp. 57-76, 2020/05/01/ 2020, doi: https://doi.org/10.1016/j.cogsys.2019.12.002.

[17] R. R. Pieters, M Veronese,A Kyrki,V, "Human-Aware Interaction: A Memory-inspired Artificial Cognitive Architecture," in *Cognitive Robot Architectures*, Vienna, Austria, December 8-9, 2016 2016, no. Proceedings of EUCognition 2016.

[18] B. J. G. Baars, Nicole M., *Fundamentals of Cognitive Neuroscience - A Beginner's Guide*. Academic Press - Elsevier, 2012.

[19] E. Tulving, "How many memory systems are there," *American Psychologist*, vol. 40, pp. 385-398, 1985.

[20] G. A. Radvansky, *Human Memory*, Third ed. Routledge, 2017.

[21] F. L. Van Harmelen, V; Porter, B, *Handbook of Knowledge Representation*, First ed. (no. Volume 1). Elsevier, 2008.

[22] Y. Chen, R. Yu, Y. Zhang, and C. Liu, "Circular formation flight control for unmanned aerial vehicles with directed network and external disturbance," *IEEE/CAA Journal of Automatica Sinica*, vol. 7, no. 2, pp. 505-516, 2020, doi: 10.1109/JAS.2019.1911669.

[23] Z. Gao and G. Guo, "Fixed-time sliding mode formation control of AUVs based on a disturbance observer," *IEEE/CAA Journal of Automatica Sinica*, vol. 7, no. 2, pp. 539-545, 2020, doi: 10.1109/JAS.2020.1003057.

[24] L. Huang, M. Zhou, K. Hao, and E. Hou, "A survey of multi-robot regular and adversarial patrolling," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 4, pp. 894-903, 2019, doi: 10.1109/JAS.2019.1911537.

[25] D. J. Blower, *Information Processing - The Maximum Entropy Principle*. CreateSpace Independent Publishing Platform, 2013.

[26] A. Francalanza, L. Aceto, and A. Ingolfsdottir, "Monitorability for the Hennessy–Milner logic with recursion," *Formal Methods in System Design*, vol. 51, no. 1, pp. 87-116, 2017/08/01 2017, doi: 10.1007/s10703-017-0273-z.

[27] M. Jasper, M. Schlüter, and B. Steffen, "Characteristic invariants in Hennessy–Milner logic," *Acta Informatica*, vol. 57, no. 3, pp. 671-687, 2020/10/01 2020, doi: 10.1007/s00236-020-00376-5.

[28] J. J. Sarbo and R. Cozijn, "Belief in reasoning," *Cognitive Systems Research*, vol. 55, pp. 245-256, 2019/06/01/ 2019, doi: https://doi.org/10.1016/j.cogsys.2019.01.004.

[29] S. Schiffer, "Integrating Qualitative Reasoning and Human-Robot Interaction in Domestic Service Robotics," *KI - Künstliche Intelligenz*, vol. 30, no. 3, pp. 257-265, 2016/10/01 2016, doi: 10.1007/s13218-016-0436-x.

[30] A. Hong, O. Igharoro, Y. Liu, F. Niroui, G. Nejat, and B. Benhabib, "Investigating Human-Robot Teams for Learning-Based Semi-autonomous Control in Urban Search and Rescue Environments," *Journal of Intelligent & Robotic Systems*, journal article August 09 2018, doi: 10.1007/s10846-018-0899-0.

[31] S. Lemaignan, M. Warnier, E. A. Sisbot, A. Clodic, and R. Alami, "Artificial cognition for social human–robot interaction: An implementation," *Artificial Intelligence*, vol. 247, pp. 45-69, 2017/06/01/ 2017, doi: https://doi.org/10.1016/j.artint.2016.07.002.

[32] D. Zhang, Aziguli, and H. Chen, "Research on Cognitive Induction Based Knowledge Acquisition," in *Computer Science-Technology and Applications, 2009. IFCSTA '09. International Forum on*, 25-27 Dec. 2009 2009, vol. 1, pp. 227-234, doi: 10.1109/ifcsta.2009.62.

[33] D. H. Perico *et al.*, "Humanoid Robot Framework for Research on Cognitive Robotics," *Journal of Control, Automation and Electrical Systems*, vol. 29, no. 4, pp. 470-479, 2018/08/01 2018, doi: 10.1007/s40313-018-0390-y.

[34] J. Pearl, *Causality: Models, Reasoning and Inference*, Second ed. Cambridge University Press, 2009.

[35] A. Darwiche, *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009, p. 548.

[36] M. Richardson and P. Domingos, "Markov logic networks," *Mach. Learn.*, vol. 62, no. 1-2, pp. 107-136, 2006, doi: 10.1007/s10994-006-5833-1.

[37] S. K. M. X. Wong, Y., "Construction of a Markov Network from Data for Probabilistic Inference," in *In Proc. 3rd Inter. Workshop on Rough Sets and Soft Computing*, 1994, pp. 562--569.

[38] D. Lowd and J. Davis, "Improving Markov network structure learning using decision trees," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 501-532, 2014.

[39] R. D. Alba, "A graph-theoretic definition of a sociometric clique," *The Journal of Mathematical Sociology,* vol. 3, no. 1, pp. 113-126, 1973/07/01 1973, doi: 10.1080/0022250x.1973.9989826.

[40] P. Domingos and D. Lowd, *Markov Logic - An Interface Layer for Artificial Intelligence*. Morgan & Claypool Publishers, 2009.

[41] D. Lowd and P. Domingos, "Efficient Weight Learning for Markov Logic Networks," in *Knowledge Discovery in Databases: PKDD 2007*, vol. 4702, J. Kok, J. Koronacki, R. Lopez de Mantaras, S. Matwin, D. Mladenič, and A. Skowron Eds., (Lecture Notes in Computer Science: Springer Berlin Heidelberg, 2007, ch. 21, pp. 200-211.

[42] D. Jain, B. Kirchlechner, and M. Beetz, "Extending Markov Logic to Model Probability Distributions in Relational Domains," in *KI 2007: Advances in Artificial Intelligence*, vol. 4667, J. Hertzberg, M. Beetz, and R. Englert Eds., (Lecture Notes in Computer Science: Springer Berlin Heidelberg, 2007, ch. 12, pp. 129-143.

[43] J. Shavlik and S. Natarajan, "Speeding up inference in Markov logic networks by preprocessing to reduce the size of the resulting grounded network," presented at the Proceedings of the 21st international jont conference on Artifical intelligence, Pasadena, California, USA, 2009.

[44] L. Getoor and J. Grant, "PRL: A probabilistic relational language," (in English), *Machine Learning*, vol. 62, no. 1-2, pp. 7-31, 2006/02/01 2006, doi: 10.1007/s10994-006-5831-3.

[45] S. D. R. Muggleton, L., "Inductive Logic Programming: Theory and Methods," *Journal of Logic Programming*, vol. 19, no. 20, pp. 629--679, 1994.

[46] W. D. Van Laer, L; De Raedt, L, "Applications of a logical discovery engine," in *In Proceedings of the AAAI Workshop on Knowledge Discovery in Databases*, 1994, pp. 263--274.

[47] H. Karimi and A. Kamandi, "A learning-based ontology alignment approach using inductive logic programming," *Expert Systems with Applications*, vol. 125, pp. 412-424, 2019/07/01/ 2019, doi: https://doi.org/10.1016/j.eswa.2019.02.014.

[48] M. P. Wellman, J. S. Breese, and R. P. Goldman, "From knowledge bases to decision models," *The Knowledge Engineering Review*, vol. 7, no. 01, pp. 35-53, 1992, doi: doi:10.1017/S0269888900006147.

[49] D. Z. Wang, Y. Chen, S. Goldberg, C. Grant, and K. Li, "Automatic knowledge base construction using probabilistic extraction, deductive reasoning, and human feedback," presented at the Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction, Montreal, Canada, 2012.

[50] J. R. Quinlan;, "Learning logical definitions from relations," *Machine Learning*, vol. 5, pp. 239--266, 1990.

[51] Y. Sato, K. Izui, T. Yamada, and S. Nishiwaki, "Data mining based on clustering and association rule analysis for knowledge discovery in multiobjective topology optimization," *Expert Systems with Applications*, vol. 119, pp. 247-261, 2019/04/01/ 2019, doi: https://doi.org/10.1016/j.eswa.2018.10.047.

[52] M. V. J. Luis, G. A. Holguin, and H. L. Mauricio, "A Methodology for Movement Planning in Autonomous Systems with Multiple Agents," in *2018 IEEE 2nd Colombian Conference on Robotics and Automation (CCRA)*, 1-3 Nov. 2018 2018, pp. 1-6, doi: 10.1109/ccra.2018.8588140.

[53] Y. Shoukry *et al.*, "Linear temporal logic motion planning for teams of underactuated robots using satisfiability modulo convex programming," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, 12-15 Dec. 2017 2017, pp. 1132-1137, doi: 10.1109/cdc.2017.8263808.

[54] F. Kamil, T. S. Hong, W. Khaksar, M. Y. Moghrabiah, N. Zulkifli, and S. A. Ahmad, "New robot navigation algorithm for arbitrary unknown dynamic environments based on future prediction and priority behavior," *Expert Systems with Applications*, vol. 86, pp. 274-291, 2017/11/15/ 2017, doi: https://doi.org/10.1016/j.eswa.2017.05.059.

[55] A. C. Tenorio-González and E. F. Morales, "Automatic discovery of concepts and actions," *Expert Systems with Applications*, vol. 92, pp. 192-205, 2018/02/01/ 2018, doi: https://doi.org/10.1016/j.eswa.2017.09.023.

[56] P. Meyer and D. V. Dimarogonas, "Hierarchical Decomposition of LTL Synthesis Problem for Nonlinear Control Systems," *IEEE Transactions on Automatic Control*, vol. 64, no. 11, pp. 1-1, 2019, doi: 10.1109/tac.2019.2902643.

[57] R. Bellman, "A Markovian Decision Process," *Indiana University Mathematics Journal*, vol. 6, no. 4, pp. 679--684, 1957.

[58] A. Rodriguez-Ramos, C. Sampedro, H. Bavle, P. de la Puente, and P. Campoy, "A Deep Reinforcement Learning Strategy for UAV Autonomous Landing on a Moving Platform," *Journal of Intelligent & Robotic Systems*, vol. 93, no. 1, pp. 351-366, 2019/02/01 2019, doi: 10.1007/s10846-018-0891-8.

[59] E. S. Low, P. Ong, and K. C. Cheah, "Solving the optimal path planning of a mobile robot using improved Q-learning," *Robotics and Autonomous Systems*, vol. 115, pp. 143-161, 2019/05/01/ 2019, doi: https://doi.org/10.1016/j.robot.2019.02.013.

[60] P. Rosenbloom, A. Demski, and V. Ustun, "The Sigma Cognitive Architecture and System: Towards Functionally Elegant Grand Unification," *Journal of Artificial General Intelligence*, vol. 7, 01/15 2016, doi: 10.1515/jagi-2016-0001.

[61] S. Franklin *et al.*, "A LIDA cognitive model tutorial," *Biologically Inspired Cognitive Architectures*, vol. 16, pp. 105-130, 2016/04/01/ 2016, doi: https://doi.org/10.1016/j.bica.2016.04.003.

[62] R. Sun, "The CLARION Cognitive Architecture: Extending Cognitive Modeling to Social Simulation," in *Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Simulation*, R. Sun Ed. Cambridge: Cambridge University Press, 2005, pp. 79-100.

[63] P. Ye, T. Wang, and F. Wang, "A Survey of Cognitive Architectures in the Past 20 Years," *IEEE Transactions on Cybernetics*, vol. 48, no. 12, pp. 3280-3290, 2018, doi: 10.1109/TCYB.2018.2857704.

[64] D. F. Lucentini and R. R. Gudwin, "A Comparison Among Cognitive Architectures: A Theoretical Analysis," *Procedia Computer Science*, vol. 71, pp. 56-61, 2015/01/01/ 2015, doi: https://doi.org/10.1016/j.procs.2015.12.198.

[65] G. Metta *et al.*, "The iCub humanoid robot: An open-systems platform for research in cognitive development," *Neural Networks*, vol. 23, no. 8, pp. 1125-1134, 2010/10/01/ 2010, doi: https://doi.org/10.1016/j.neunet.2010.08.010.

[66] R. A. Brooks, C. Breazeal, M. Marjanović, B. Scassellati, and M. M. Williamson, "The Cog Project: Building a Humanoid Robot," in *Computation for Metaphors, Analogy, and Agents*, Berlin, Heidelberg, C. L. Nehaniv, Ed., 1999// 1999: Springer Berlin Heidelberg, pp. 52-87.

[67] P. J. Antsaklis and A. Rahnama, "Control and Machine Intelligence for System Autonomy," *Journal of Intelligent & Robotic Systems*, vol. 91, no. 1, pp. 23-34, 2018/07/01 2018, doi: 10.1007/s10846-018-0832-6.

[68] D. Hernández García, C. A. Monje, and C. Balaguer, "Task Oriented Control of a Humanoid Robot Through the Implementation of a Cognitive Architecture," *Journal of Intelligent & Robotic Systems*, vol. 85, no. 1, pp. 3-25, 2017/01/01 2017, doi: 10.1007/s10846-016-0383-7.

[69] D. Drenjanac, S. D. K. Tomic, and E. Kuhn, "A Semantic Framework for Modeling Adaptive Autonomy in Task Allocation in Robotic Fleets," in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2015 IEEE 24th International Conference on*, 15-17 June 2015 2015, pp. 15-20, doi: 10.1109/wetice.2015.29.

[70] A. Martínez-Tenor, J. A. Fernández-Madrigal, A. Cruz-Martín, and J. González-Jiménez, "Towards a common implementation of reinforcement learning for multiple robotic tasks," *Expert Systems with Applications*, vol. 100, pp. 246-259, 2018/06/15/ 2018, doi: https://doi.org/10.1016/j.eswa.2017.11.011.

[71] C. Sampedro, A. Rodriguez-Ramos, H. Bavle, A. Carrio, P. de la Puente, and P. Campoy, "A Fully-Autonomous Aerial Robot for Search and Rescue Applications in Indoor Environments using Learning-Based Techniques," *Journal of Intelligent & Robotic Systems*, 2018/07/03 2018, doi: 10.1007/s10846-018-0898-1.

[72] G. Rishwaraj and S. G. Ponnambalam, "Integrated trust based control system for multirobot systems: Development and experimentation in real environment," *Expert Systems with Applications*, vol. 86, pp. 177-189, 2017/11/15/ 2017, doi: https://doi.org/10.1016/j.eswa.2017.05.074.

[73] T. Wareham and A. Vardy, "Putting it together: the computational complexity of designing robot controllers and environments for distributed construction," *Swarm Intelligence*, journal article vol. 12, no. 2, pp. 111-128, June 01 2018, doi: 10.1007/s11721-017-0152-7.

[74] U. Kurup and C. Lebiere, "What can cognitive architectures do for robotics?," *Biologically Inspired Cognitive Architectures*, vol. 2, pp. 88-99, 2012/10/01/ 2012, doi: https://doi.org/10.1016/j.bica.2012.07.004.

[75] R. Gudwin *et al.*, "The TROCA Project: An autonomous transportation robot controlled by a cognitive architecture," *Cognitive Systems Research*, vol. 59, pp. 179-197, 2020/01/01/ 2020, doi: https://doi.org/10.1016/j.cogsys.2019.09.011.

[76] P. Bustos, L. J. Manso, A. J. Bandera, J. P. Bandera, I. García-Varea, and J. Martínez-Gómez, "The CORTEX cognitive robotics architecture: Use cases," *Cognitive Systems Research*, vol. 55, pp. 107-123, 2019/06/01/ 2019, doi: https://doi.org/10.1016/j.cogsys.2019.01.003.

[77] K. Miyazawa, T. Horii, T. Aoki, and T. Nagai, "Integrated Cognitive Architecture for Robot Learning of Action and Language," (in English), *Frontiers in Robotics and AI*,

Original Research vol. 6, no. 131, 2019-November-29 2019, doi: 10.3389/frobt.2019.00131.

[78] J. R. Cole and D. Reitter, "The role of working memory in syntactic sentence realization: A modeling & simulation approach," *Cognitive Systems Research*, vol. 55, pp. 95-106, 2019/06/01/ 2019, doi: https://doi.org/10.1016/j.cogsys.2019.01.001.

[79] D. Vanderelst and A. Winfield, "An architecture for ethical robots inspired by the simulation theory of cognition," *Cognitive Systems Research*, vol. 48, pp. 56-66, 2018/05/01/ 2018, doi: https://doi.org/10.1016/j.cogsys.2017.04.002.

[80] L. H. Favela, "Editor's introduction: Innovative dynamical approaches to cognitive systems," *Cognitive Systems Research*, vol. 58, pp. 156-159, 2019/12/01/ 2019, doi: https://doi.org/10.1016/j.cogsys.2019.06.001.

[81] M. Demir, N. Cooke, and P. Amazeen, "A Conceptual Model of Team Dynamical Behaviors and Performance in Human-Autonomy Teaming," *Cognitive Systems Research*, vol. 52, 08/01 2018, doi: 10.1016/j.cogsys.2018.07.029.

[82] M. Demir, "The Impact of Coordination Quality on Coordination Dynamics and Team Performance: When Humans Team with Autonomy," Doctor of Philosophy, Arizona State University, 2017, 2017.

[83] N. J. McNeese, M. Demir, N. J. Cooke, and C. Myers, "Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming," *Human Factors*, vol. 60, no. 2, pp. 262-273, 2018/03/01 2017, doi: 10.1177/0018720817743223.

[84] E. L. John, L. Christian, and S. R. Paul, "A Standard Model of the Mind: Toward a Common Computational Framework across Artificial Intelligence, Cognitive Science, Neuroscience, and Robotics," *AI Magazine*, vol. 38, no. 4, 12/28 2017, doi: 10.1609/aimag.v38i4.2744.

[85] A. Chella and A. Pipitone, "A cognitive architecture for inner speech," *Cognitive Systems Research*, vol. 59, pp. 287-292, 2020/01/01/ 2020, doi: https://doi.org/10.1016/j.cogsys.2019.09.010.

[86] G. Baghdadi, F. Towhidkhah, and R. Rostami, "A mathematical model of the interaction between bottom-up and top-down attention controllers in response to a target and a distractor in human beings," *Cognitive Systems Research*, vol. 58, pp. 234-252, 2019/12/01/ 2019, doi: https://doi.org/10.1016/j.cogsys.2019.07.007.

[87] L. H. Favela and M. M. J. W. van Rooij, "Reasoning across continuous landscapes: A nonlinear dynamical systems theory approach to reasoning," *Cognitive Systems Research*, vol. 54, pp. 189-198, 2019/05/01/ 2019, doi: https://doi.org/10.1016/j.cogsys.2018.12.013.

[88] G. Van Orden, G. Hollis, and S. Wallot, "The Blue-Collar Brain," (in English), *Frontiers in Physiology*, Hypothesis and Theory vol. 3, no. 207, 2012-June-18 2012, doi: 10.3389/fphys.2012.00207.

[89] D. Wang, Y. Hu, and T. Ma, "Mobile robot navigation with the combination of supervised learning in cerebellum and reward-based learning in basal ganglia," *Cognitive Systems Research*, vol. 59, pp. 1-14, 2020/01/01/ 2020, doi: https://doi.org/10.1016/j.cogsys.2019.09.006.

[90] Y. Jian and Y. Li, "Research on intelligent cognitive function enhancement of intelligent robot based on ant colony algorithm," *Cognitive Systems Research*, vol. 56, pp. 203-212, 2019/08/01/ 2019, doi: https://doi.org/10.1016/j.cogsys.2018.12.014.

[91] B. J. G. Baars, Nicole M., *Fundamentals of Cognitive Neuroscience - A Beginner's Guide*, Second ed. Academic Press - Elsevier, 2018.

[92] M. Persiani, A. M. Franchi, and G. Gini, "A working memory model improves cognitive control in agents and robots," *Cognitive Systems Research*, vol. 51, pp. 1-13, 2018/10/01/ 2018, doi: https://doi.org/10.1016/j.cogsys.2018.04.014.

[93] H. J. Levesque, *The Logic of Knowledge Bases*. MIT Press, 2000.

[94] S. Brass and U. W. Lipeck, "Specifying closed world assumptions for logic databases," in *MFDBS 89: 2nd Symposium on Mathematical Fundamentals of Database Systems Visegrád, Hungary, June 26–30, 1989 Proceedings*, J. Demetrovics and B. Thalheim Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, pp. 68-84.

[95] G. F. Luger, *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*, Fifth Edition ed. Pearson Education Ltd, 2005, p. 903.

[96] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal, The*, vol. 27, no. 4, pp. 623-656, 1948, doi: 10.1002/j.1538-7305.1948.tb00917.x.

[97] J. Pitman, *Probability*. Springer-Verlag, 1993, p. 559.

[98] J. H. Joseph K. Blitzstein, *Introduction to Probability* (Texts in Statistical Science). CRC Press, 2015.

[99] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Micro Machine and Human Science, 1995. MHS '95., Proceedings of the Sixth International Symposium on*, 4-6 Oct 1995 1995, pp. 39-43, doi: 10.1109/mhs.1995.494215.

[100] P. K. Das, B. M. Sahoo, H. S. Behera, and S. Vashisht, "An improved particle swarm optimization for multi-robot path planning," in *2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH)*, 3-5 Feb. 2016 2016, pp. 97-106, doi: 10.1109/iciccs.2016.7542324.

[101] Q. Cai, T. Long, Z. Wang, Y. Wen, and J. Kou, "Multiple paths planning for UAVs using particle swarm optimization with sequential niche technique," in *2016 Chinese Control and Decision Conference (CCDC)*, 28-30 May 2016 2016, pp. 4730-4734, doi: 10.1109/ccdc.2016.7531839.

[102] C. Walha, H. Bezine, and A. M. Alimi, "A Multi-Objective Particle Swarm Optimization approach to robotic grasping," in *Individual and Collective Behaviors in Robotics (ICBR), 2013 International Conference on*, 15-17 Dec. 2013 2013, pp. 120-125, doi: 10.1109/icbr.2013.6729267.

[103] S. Yuhui and R. Eberhart, "A modified particle swarm optimizer," in *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, 4-9 May 1998 1998, pp. 69-73, doi: 10.1109/icec.1998.699146.

[104] C. Wei-Neng, Z. Jun, H. S. H. Chung, Z. Wen-Liang, W. Wei-gang, and S. Yu-Hui, "A Novel Set-Based Particle Swarm Optimization Method for Discrete Optimization Problems," *Evolutionary Computation, IEEE Transactions on*, vol. 14, no. 2, pp. 278-300, 2010, doi: 10.1109/tevc.2009.2030331.

[105] X. Yu *et al.*, "Set-Based Discrete Particle Swarm Optimization Based on Decomposition for Permutation-Based Multiobjective Combinatorial Optimization Problems," *IEEE Transactions on Cybernetics*, vol. PP, no. 99, pp. 1-15, 2018, doi: 10.1109/tcyb.2017.2728120.

[106] W.-N. Chen and D.-Z. Tan, "Set-based discrete particle swarm optimization and its applications: a survey," *Frontiers of Computer Science*, vol. 12, no. 2, pp. 203-216, 2018/04/01 2018, doi: 10.1007/s11704-018-7155-4.

[107] Y. Weng, W. Chen, A. Song, and J. Zhang, "Set-Based Comprehensive Learning Particle Swarm optimization for Virtual Machine Placement Problem," in *2018 Ninth*

International Conference on Intelligent Control and Information Processing (ICICIP), 9-11 Nov. 2018 2018, pp. 243-250, doi: 10.1109/icicip.2018.8606676.

[108] Y. Zhong, J. Lin, L. Wang, and H. Zhang, "Discrete comprehensive learning particle swarm optimization algorithm with Metropolis acceptance criterion for traveling salesman problem," *Swarm and Evolutionary Computation*, 2018/03/20/ 2018, doi: https://doi.org/10.1016/j.swevo.2018.02.017.

[109] A. ŞİMŞEk and R. Kara, "Using swarm intelligence algorithms to detect influential individuals for influence maximization in social networks," *Expert Systems with Applications*, vol. 114, pp. 224-236, 2018/12/30/ 2018, doi: https://doi.org/10.1016/j.eswa.2018.07.038.

[110] K. V. Shihabudheen, M. Mahesh, and G. N. Pillai, "Particle swarm optimization based extreme learning neuro-fuzzy system for regression and classification," *Expert Systems with Applications*, vol. 92, pp. 474-484, 2018/02/01/ 2018, doi: https://doi.org/10.1016/j.eswa.2017.09.037.

[111] M. Mahmood, S. Mathavan, and M. Rahman, "A parameter-free discrete particle swarm algorithm and its application to multi-objective pavement maintenance schemes," *Swarm and Evolutionary Computation*, 2018/03/31/ 2018, doi: https://doi.org/10.1016/j.swevo.2018.03.013.

[112] L. Cao, L. Xu, and E. D. Goodman, "A collaboration-based particle swarm optimizer with history-guided estimation for optimization in dynamic environments," *Expert Systems with Applications*, vol. 120, pp. 1-13, 2019/04/15/ 2019, doi: https://doi.org/10.1016/j.eswa.2018.11.020.

[113] C. W. Cleghorn and A. P. Engelbrecht, "Particle swarm stability: a theoretical extension using the non-stagnate distribution assumption," *Swarm Intelligence*, journal article vol. 12, no. 1, pp. 1-22, March 01 2018, doi: 10.1007/s11721-017-0141-x.

[114] M. Chih, "Three pseudo-utility ratio-inspired particle swarm optimization with local search for multidimensional knapsack problem," *Swarm and Evolutionary Computation*, vol. 39, pp. 279-296, 2018/04/01/ 2018, doi: https://doi.org/10.1016/j.swevo.2017.10.008.

[115] Y. Chen, L. Li, H. Peng, J. Xiao, and Q. Wu, "Dynamic multi-swarm differential learning particle swarm optimizer," *Swarm and Evolutionary Computation*, vol. 39, pp. 209-221, 2018/04/01/ 2018, doi: https://doi.org/10.1016/j.swevo.2017.10.004.

[116] M. Z. Shirazi, T. Pamulapati, R. Mallipeddi, and K. C. Veluvolu, "Particle Swarm Optimization with Ensemble of Inertia Weight Strategies," Cham, 2017: Springer International Publishing, in Advances in Swarm Intelligence, pp. 140-147.

[117] J.-J. Wang and G.-Y. Liu, "Saturated control design of a quadrotor with heterogeneous comprehensive learning particle swarm optimization," *Swarm and Evolutionary Computation*, vol. 46, pp. 84-96, 2019/05/01/ 2019, doi: https://doi.org/10.1016/j.swevo.2019.02.008.

[118] M. Brady, V. K. Mamanduru, and M. K. Tiwari, "An evolutionary algorithmic approach to determine the Nash equilibrium in a duopoly with nonlinearities and constraints," *Expert Systems with Applications*, vol. 74, pp. 29-40, 2017/05/15/ 2017, doi: https://doi.org/10.1016/j.eswa.2016.12.037.

[119] M. Maschler, E. Solan, and S. Zamir, *Game Theory*. Cambridge University Press, 2013.

[120] S. Tadelis, *Game Theory - An Introduction*. Princeton University Press, 2013.

[121] S. Schoenmackers, O. Etzioni, D. S. Weld, and J. Davis, "Learning first-order Horn clauses from web text," presented at the Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, Cambridge, Massachusetts, 2010.

[122] A. P. Engelbrecht, *Fundamentals of Computational Swarm Intelligence*. John Wiley & Sons Ltd., 2005, p. 672.

[123] D. Simon, *Evolutionary Optimization Algorithms: Biologically Inspired and Population-Based Approaches to Computer Intelligence*, First Edition ed. John Wiley & Sons, Inc, 2013, p. 742.

[124] J. L. Sun, Choi-Hong; Wu, Xiao-Jun, *Particle Swarm Optimization: Classical and Quantum Perspectives* (Analysis and Scientific Computing Series). Chapman & Hall/CRC, 2012, p. 419.

[125] G. W. F. Corder, D.I., *Nonparametric Statistics - A Step-by-Step Approach*, Second ed. John Wiley & Sons, Ltd, 2014, p. 282.

[126] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates, 1988, p. 400.

[127] D. De Jager. "Cognitive Robotics - Autonomous UAV recharging." https://youtu.be/OiA6Tip923U.

[128] D. De Jager. "Cognitive Robotics - Autonomous medical supplies delivery." https://youtu.be/ZPCfCGsr7eY.

[129] D. De Jager. "Cognitive Robotics - On-demand UAV delivery of COVID-19 equipment." https://youtu.be/94Y_moDr43s.

[130] D. De Jager. "Cognitive Robotics - On-demand UAV security surveillance support." https://youtu.be/LIRCporJmxM.