An Affective Computing Multilayer Cognitive Architecture for Evolutionary Cognitive Robotics

by

Zheming Zhang

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Engineering.

Victoria University of Wellington
2020

Abstract

Robots are entering our daily lives from self-driving cars to health-care robots. Historically, pre-programmed robots were vulnerable to changing conditions in daily-life, primarily because of a lack of ability to generate novel, non-preset flexible solutions. Thus there is a need for robotics to incorporate adaptation, which is a trait of higher-order natural species. This adaptation allows higher-order natural species to change their behaviours and internal mechanisms based on experience with often dynamic environment. The ability to adapt emerged through evolutionary processes.

Evolutionary Robotics is an approach to create autonomous robots that are capable of automatically generating artificial behaviors and morphologies to achieve adaptation. Evolutionary robotics has the potential to automatically synthesize controllers for real autonomous robots and generate solutions to complete tasks in the uncertain real-world. Compared to the inflexibility of pre-programmed robots, evolutionary robots are able to learn flexible solutions to given tasks through evolutionary methods.

Cognitive robotics, a branch of artificial cognitive systems research, is such an attempt to create autonomous robots by applying bio-inspired methods. As the robot interacts with environment, an underlying cognitive system can learn its own solutions toward task completion. This learning-solution-from-interaction approach, also termed as a Reinforcement Learning (RL) approach, is widely applied in cognitive robotics to learn the solutions automatically. Ideally, the solutions can emerge in the cognitive system through the trial-and-error process of the RL approach without introducing human bias.

This thesis aims to develop an evolutionary cognitive architecture (system) for a robot that can learn adaptive solutions to complete tasks. Inspired by emotion theories, this work proposes *Affective Computing Multilayer Cognitive Architecture* (*ACMCA*), a universal cognitive architecture, which is able to learn diverse solutions. Extending from previous work, ACMCA has a five-layer structure, where each layer aims to achieve different components of the solutions. The position of this thesis is that introducing a novel emotion inspired multilayer architecture that produces task solutions through

subsumption operations and underlying appropriate machine learning algorithms will allow a robot to complete admissible tasks.

ACMCA's five layers are: *primary reinforcer* layer, *secondary reinforcer* layer, *core affect state* layer, *strategy* layer, and *behaviour* layer. This five-layer decomposition also meets the traditional decomposition of a mobile control system into functional modules (e.g. perception, modelling, planning, task execution, and motor control). Each layer contains computing nodes as functional modules that process various Stimuli, Actions, and their consequential Outcomes of the cognitive system. In this work, 17 computing nodes and their connections in ACMCA represent the solutions that a mobile robot has learned to complete navigation tasks in complex scenarios.

Inspired by the Constructive Theory [1] and the robotic subsumption system, this work proposes a contingency-based subsumption approach to construct ACMCA. This contingency is termed *Stimuli-Action-Outcome Contingency* (*SAOC*), which is extended from the Action-Outcome (AO) contingency of Construction Theory. SAOCs are represented by "if-then" rules, termed SAOC rules, which encapsulate Stimuli, Actions, and their consequential Outcomes, providing clear symbolic interpretations. That is, the symbolic meaning of a SAOC rule can be interpreted as: if the input stimulus is perceived, the output action will be advocated as a cognitive response, expecting the outcome of the action with an estimation of relevance. As low-level computing nodes encapsulate Stimulus, Actions, and Outcomes, high-level computing nodes can subsume these low-level ones through the form of SAOC rules. Therefore, the proposed ACMCA can be constructed by subsumption layers of Stimuli-Action-Outcome Contingency (SAOC) rules.

This work applies machine learning techniques to facilitate ACMCA's real-world robotic implementation. This work selects Accuracy-based Learning Classifier Systems (*XCS*) algorithms as the underlying machine learning techniques that are deployed at computing nodes for the contingency-based subsumption operations. The *mitosis approach* of XCS and the *XCS with a Combined Reward method* (*XCSCR*) are two novel variants of XCS algorithm. They are proposed to amend two challenges that occur when the standard XCS approaches are applied for robotic applications. The mitosis approach introduces an accuracy pressure into the algorithm's evolutionary process, improving the algorithms' performance in robotic applications where noisy interferences exist. The

[1]Constructive Theory is a majority emotion theory. Three majority emotion theories are Constructive Theory, Appraisal Theory, and Basic Emotion Theory.

XCSCR enables the policy to emerge earlier and more frequently than the existing benchmark approaches in multistep problems. Therefore, a robot with the XCSCR can handle a multistep scenario more effectively than those with the benchmarked algorithms.

This work conducts five experiments to test the capability of ACMCA and its underlying algorithms in learning solutions for robotic navigation tasks. The five experiments are conducted as follows: *reflex-learning*, *IR-tuning*, *deliberation-establishing*, *emotion model*, *and combined reward assignment*. As the results of the experiments, three different affective patterns have emerged in the first three experiments, an emotion model has emerged in the fourth experiments, and the fifth experiment explores ACMCA's potential implementation in the life-long learning scenario.

These results demonstrate that ACMCA, a novel emotion inspired multilayer architecture, can produce task solutions through contingency-based subsumption operations and underlying appropriate machine learning algorithms, allowing a robot to complete admissible tasks through evolutionary processes. The contingency-based subsumption operations can establish three contingencies and one emotion model between the subsumed components by multiple RL agents which deploy the proposed mitosis approach of XCS algorithms. These three emotion patterns and emotion model can consistently improve the robot's navigation performance with interpretable explanations. These two variants of XCS algorithms can amend shortfalls of the standard XCS approach in real-world robotic implementations. It has been demonstrated that the diverse solutions learned by ACMCA improve the navigation performance of the robot in terms of higher flexibility, reduction in continuous collisions and shorter navigation time consumption.

Acknowledgments

I would first like to express my deepest gratitude to my supervisors, Dale Carnegie and Will Browne, for their continuous support of my PhD study. I greatly appreciate the amount of time and resource you have put into helping me with academic training, including my research and English. You convincingly guided me to do the research professionally, especially when I became stubborn on details and thus missed the big picture. You continuously encouraged and supported me to improve my English; especially, you spent enormous effort on my writing, providing insightful comments that inspired my research. Otherwise, the presence of this thesis would become an impossible mission. You are the best supervisors that I have ever met during my time in academia.

I am grateful to Victoria University of Wellington (VUW) and Xiamen University of Technology (XMUT) for their agreement that offered me the opportunity to pursue a PhD at VUW. Without the tuition fee supported by VUW, I would not have been able to conduct this research.

I wish to thank our Evolutionary Computation Research Group (ECRG). ECRG provides weekly meetings, discussions, seminars, presentations, etc. These academic practices are highly valuable and I receive lots of help from the group.

I would like to thank staffs of the School of Engineering and Computer Science (ECS). Thanks to Diana Siwiak for providing oral practice to discuss my project. This helped me explain my ideas with greater clarity when I was struggling to write the thesis. Thanks to Mark Davies and Tim Exley for their technical supports. Thanks to Peter ("Pondy") Andreae, Patricia Stein and Sarah Dillon for their help and kindness.

I want to thank my friends and colleges in my office: Raghavender Deshagoni, Abhi Chatterjee, Trung Nguyen, Roshni Babu, and Yi Liu. I am glad that you all look good and prosper as usual, because I was always worried that your sense of humour might become weird if you were suffering from my terrible jokes for years.

I would like to pay my special regards to Kirsten Reid, a senior learning adviser of the Student Learning Center of VUW. I truly appreciate advice on writing that you all provided during my study. Your advice on my writing, including this thesis, is extremely valuable. For these years, I learnt to improve my English writing through your feedback.

Last but not least, I give my whole-hearted thanks to my parents for supporting me during my PhD. You still offered me your support and understanding although my separation from family was extremely hard for you. I cannot finish this thesis without your supports.

Contents

List of Figures

List of Tables

Glossary

ACMCA Affective Computing Multilayer Cognitive Architecture.

IR Inflation Radius.

LCSs Learning Classifier Systems.

RL Reinforcement Learning.

ROS Robot Operating System.

SAOC Stimulus-Action-Outcome Contingency.

VUW Victoria University of Wellington.

XCS Accuracy-based Learning Classifier Systems.

XCSCR XCS with a combined reward method.

Chapter 1

Introduction

1.1 Motivation

Robots are being integrated into human society from self-driving cars to health-care robots (i.e. robots that are helping to fight the coronavirus pandemic [1]). “Hard-wired" robots are vulnerable in daily-life as it contains changing conditions that can confuse these pre-programmed robots. Thus, there is a need for intelligent robots to incorporate adaptation to improve their performances in daily scenarios. As a trait of higher-order species, adaptation allows natural agents to adjust their mental processes and control of their bodies in often dynamic environments. Similarly, robots can achieve adaptation by evolving internal mechanisms and behaviours through interactions with environments.

Evolutionary robotics is an approach to create artificial “brains” and morphologies of autonomous robots [2, 3]. In evolutionary robotics, evolutionary methods are proposed to search for solutions and avoid the bias introduced by human designers [4]. The bias is rooted in the designer’s limitations of interpretation on the true pattern of the environment. Thus the bias can decrease the robot’s performance when the interpretation cannot reflect the true pattern in the environment or the environment changes over time so that the interpreted pattern does not cover the ideal behaviours. Evolutionary robotics avoids bias by adapting behaviours based on the worth of behaviours/actions as direct feedback from the environment. Compared with the inflexible solutions that are often interpreted and perceived by human beings, evolutionary robots can learn flexible solutions that are better adapted to environmental changes [4, 5]. For example,

evolutionary algorithms can be applied to achieve adaptive behaviours through their underlying modules. In these cases, evolutionary algorithms are considered as optimising approaches for these modules [4, 6]. Therefore, evolutionary robotics has the potential to automatically synthesize controllers for real autonomous robots and generate solutions to complete tasks in the uncertain real-world without human intervention [7].

Evolutionary cognitive robotics embodies cognition within evolutionary robotics. Cognition refers to the mental processes of perception, memory, reasoning, and learning that an agent applied to inhabit and adapt to the environment [8, 9, 10]. In order to create cognition in an autonomous robots, an abstract cognitive architecture is needed. That is, the design of the architecture of a robot can be inspired by natural cognitive systems that provide frameworks for the acquisition, representation, and use of knowledge. Although there is still much debate in cognitive science on how animals adapt to the environment, enough theories exist to act as inspirations to construct a robotic cognitive architecture.

Evolutionary cognitive robotics can integrate emotion theory into control systems to create *affective* robots. Psychologically, emotional intelligence relates to "the ability to recognize emotions in others, using emotions to support thinking and actions, understand emotions, and regulating emotions" [6]. In robotics, affective robots mainly focuses on two topics: Human-Robot Interaction (HRI) and emotion-inspired decision-making and responses.

The first topics focuses on developing robots that can "detect common human communication cues for more natural interactions" [11]. Therefore, affective robot in HRI contains three computational tasks: affect detection, affect interpretation, and affective inference [12]. For example, Park et al. [13] proposed a robotic system that increases children's engagement in their learning processes based on verbal and nonverbal affective cues. As this work does not require interactions with humans by accurately interpreting and appropriately responding to natural human communication cues, this work will not focus on this topic.

The second topic is the concern of this work. The current research emphasizes emotions as internal states that influence behaviours. Emotions are elicited as a summary of perception through well-defined models. For example, Butz et al. introduce "curiosity", "wealth", "progress", and "health" as four basic motivations for Mario in its Super Mario world [14]. The motivation system will trigger different goal events and learning processes. Mario (the agent) can make a probabilistic choice among different

goals based on the current motivation, leading an emergence of experience-dependent behaviours. Jacobs et al. [15] predefines the emotions (i.e joy, fear, hope) and demonstrates their dynamic during reinforcement learning agents were trained in a simulated maze environment. Through well-defined emotional models, the dynamic intensity of emotion states can be interpreted with the properties of emotion dynamics in humans. Moerland et al. [16] specify models of hope and fear based on the robot's performance in specific Pacman scenarios. Emotion models of hope and fear are specified by the best and worst forward traces, leading the eliciting of these emotion states (e.g. hope and fear) in the agent with interpretable explanations. In these work, emotions that are elicited in the robotic architecture does not frequently engage within the robot's learning loop for adaptive behaviours.

However, robotic cognitive architecture can adopt a fully learning approaches of co-evolving of emotion states and robot's adaptive behaviours. Previously, Williams [17, 18] proposed an emotion inspired cognitive architecture (system), which can learn solutions for robotic navigation tasks through an RL and evolutionary approach. Inspired by Constructive Theory [1], this cognitive architecture can automatically construct an emotion model, which, for the first time, can generate affective solutions for adaptive path-planning of robotic navigation. The emotion model is constructed by three layers of different computing nodes for perception, emotion, and execution respectively. These computing nodes are termed reinforcers, emotions, and modifiers. The mappings between reinforcers, emotions, and modifiers represent the adaptive solutions, which evolve through the underlying RL agent of the architecture. After multiple RL iterations, an experienced emotion model emerges and this emotion model can provide solutions that improve the adaptation of the path-planning of the robot. This work proposed the novel integration approach of emerging emotions and adaptive behaviours that researchers "should start focussing on" [19].

However, in this existing work it is difficult to generate heterogeneous solutions. Although this solution-emerging process requires no human interference, the solutions (reinforcer-emotion-modifier mappings) lack diversity. Additional emotions in the emotion layer only bring redundant reinforcer-emotion-modifier mappings rather than diverse solutions. That is, this architecture cannot achieve heterogeneous solutions in the experimental environment by merely increasing computing nodes (e.g. emotions). This

[1] Constructive Theory is an emotion theory (see details in Section 2.3)

suggests a limitation in the scalability of the architecture.

Hence, the aim of this thesis is to develop an evolutionary cognitive architecture (system) for a mobile robot that can learn adaptive solutions to complete tasks. Inspired by emotion theories, this work proposes Affective Computing Multilayer Cognitive Architecture (ACMCA), a universal cognitive architecture, to construct diverse solutions for robotic tasks. Extending from the previous work, this ACMCA has a five-layer structure, where each layer aims to achieve different components of the solution. The position of this thesis is that introducing a novel, emotion inspired multilayer architecture that produces task solutions with underlying appropriate learning algorithms will allow a robot to complete admissible tasks [2].

These five layers are ranked into three levels of the hierarchy of ACMCA. Components in the higher-level layers of the architecture can subsume components in the lower-layers by their symbolic explanations. The subsumption operations bring two advantages. Firstly, the operations can establish interpretable contingencies between the subsumed components. The contingencies that are established in the middle-level layer can be interpreted as emerged affective patterns, and those in the highest-level can be considered as the evolved emotion model. Secondly, the subsumption operations facilitate the emerging and evolving processes of the contingencies. This is because the diverse components can innately and easily create an almost infinite search space for the solutions. The subsumption operations on the contingencies can exponentially reduce the entire search space, allowing optimal diverse solutions to emerge in the architecture.

1.2 Objectives

This work presents and examines a novel Affective Computing Multiplayer Cognitive Architecture (ACMCA) that can evolve diverse solutions of admissible robot tasks. The proposed objectives of the five-layer architecture are as follows:

(1) To create a multilayer cognitive architecture, which can improve a robot's capability to complete admissible tasks. The multilayer cognitive architecture will

[2] An admissible task for a robot is a task that the robot can make every effort to complete through its physical mechanism. For example, a task that requires a wheeled mobile robot to fly is not an admissible task for the robot.

contain diverse computing nodes distributed among different layers of the architecture. Each computing node focuses on a specific topic of the perception, the localization, the mapping, the planning or the execution of a navigation task. Thus a computing node can provide a unique solution component of a complete solution to the given task. Through the cooperation of computing nodes, the multilayer cognitive architecture is to generate solutions that allow a robot to perform flexible behaviours to achieve a given task, demonstrating its adaptation to the given environment. This multilayer cognitive architecture will be a novel cognitive architecture that contributes to evolutionary robotics.

(2) To develop the multilayer architecture (1) into a novel emotion-inspired, affective computing multilayer cognitive architecture, which can generate affective responses for a robot. Inspirations from different emotion theories (e.g. Constructive Theory, Appraisal Theory, and Basic Emotion Theory) will be applied to construct affective computing processes of the cognitive system. In affective computing processes, diverse *stimuli* [3] will be generated as a robot perceives the environment. These stimuli will elicit non-preset *emotion states* [4] that summarise the robot's perception and action tendency. These elicited emotion states will automatically map to *actions* [5], allowing the robot to respond flexibly to achieve anticipated *outcomes* in the given task. Through a robotic implementation of inspirations from emotion theories, this work can provide insights for psychological theories, cognitive systems and Artificial Intelligence.

(3) To introduce a novel contingency-based subsumption operation to construct the proposed Affective Computing Multiplayer Cognitive Architecture. Compared to the traditional behaviour-based subsumption robotic system, the proposed architecture (ACMCA) will be constructed by layers of Stimuli-Action-Outcome Contingency (SAOC) rules, which describe contingencies between stimuli, actions, and their consequential outcomes. As stimuli, actions, and outcomes are encapsulated in low-level computing nodes, high-level computing nodes can establish interpretable SAOC rules by subsuming these low-level nodes through

[3]The stimuli are categoried into *primary reinforcer* and *secondary reinforcer* in this work.

[4]The emotion states are termed *core affect states* in this work.

[5]The actions are categoried into *strategies* and *behaviours* in this work.

evolutionary processes. When a robot interacts with the environment, the SAOC rules become accurate by underlying machine learning agents that are deployed for the evolutionary processes. As SAOC rules establish mappings between computing nodes, the proposed ACMCA will construct through contingency-based subsumption operations. Therefore, ACMCA will be a novel contingency-based subsumption system that contributes to robotic control systems.

(4) To develop variants of XCS algorithms for real-world robotic implementations. The mitosis approach and XCSCR (XCS with Combined Reward Method) are two variants of XCS algorithms that can be deployed for real-world robotic implementations. They are proposed for the underlying machine learning agents that will be deployed at computing nodes of ACMCA. The mitosis approach introduces an accuracy pressure to amend the overgeneralized tendency of the benchmark XCS algorithm. The mitosis approach excludes inaccurate niche-coverage of overgeneralized classifiers and passes on only the accurate niche-coverage to "children" classifiers. In this approach, accurate classifiers can be made robust against interference from noise. The XCSCR method is proposed for the multistep reward-assignment problem for robotic applications. By combined efforts of different reward-assignment mechanisms, the XCSCR method can bootstrap the emerging of global optimal policies in early learning iterations. These two variants of XCS algorithms are to be deployed for the learning agents in ACMCA, aiming to amend the shortfalls of the standard XCS approach in the implementation of ACMCA. Therefore, these two underlying novel XCS algorithms will extend the usage of XCS algorithms for real-world robotic implementations.

(5) To test the proposed approach in realistic indoor environments (i.e. office domain). These environments will have irregular features to test the robustness of the approach in navigation tasks. The domain will also have dynamic components to test the range of known capabilities. A simulated robotic platform will be used for efficiency and a real-world robot will be used for authenticity. This will also test the transferability of the proposed approach between these two platforms.

1.3 Thesis Outline

Chapter Two is an overview of robotic cognitive systems, covering cognitive systems, emotion theories, robotic subsumption systems, machine learning algorithms, a previous work of emotion inspired cognitive system, and the robotic platform utilised in the task. Firstly, a review from the field of cognitive architecture provides the general background of this work, providing a review of the various existing architectures for cognitive systems. Secondly, three emotion theories, which provide inspirations of the information processing progress in this work, are introduced. Third, behaviour-based subsumption operations that construct robotic control systems are introduced as the background of our contingency-based subsumption approach. Fourth, machine learning algorithms are presented as the underlying approach for learning knowledge in this robotic application. Fifth, a previous work of emotion inspired cognitive system, which this work is extended from, is reviewed. Finally, a mobile robot for conducting the experiments is illustrated with its hardware and software operating system.

Chapter Three describes the methodology of ACMCA, which learns diverse affective solutions for a robotic task. The chapter starts with a discussion of the previous work of emotion inspired cognitive system, which the proposed five-layer cognitive architecture is extended from. In the design of the architecture, an overview of the novel affective solution, that is encapsulated in the proposed five-layer cognitive system, is introduced. That is, the proposed cognitive architecture targets to achieve the first objective from this part. In the description of the affective solution, physiological inspirations are introduced to design affective computing processes that are executed among the five layers of the proposed cognitive system. Specifically, the functions of the five layers are based on the inspirations from three majority emotion theories: Constructive Theory, Appraisal Theory, and Basic Emotion Theory. That is, the proposed cognitive architecture is developed into an emotion-inspired one. This is the second objective of this work. In the design of computing nodes in the layers, the function of each computing node is introduced by the sequence of the five layers. Computing nodes in high-level can subsume low-level ones according to various Stimuli-Action-Outcome Contingency (SAOC), eventually leading to the construction of

the proposed cognitive architecture (ACMCA). That is, the third objective is achieved as the contingency-based subsumption operations constructed ACMCA. The first and second objectives are also achieved along with the achievement of the third objective at the end of this chapter.

Chapter Four describes two novel machine learning algorithms proposed for robotic applications. The *mitosis approach* of XCS and the *XCS with a Combined Reward method* (*XCSCR*) are proposed to amend two challenges that occur when the standard XCS approaches are applied for robotic applications. The two proposed algorithms will improve the performance of the standard XCS algorithm for robotic applications. Therefore, the fourth objective is achieved in this chapter.

Chapter Five presents the results that demonstrate the utility of ACMCA in learning solutions in various indoor scenarios. Both simulated and real-world robotic platforms are engaged to test the robustness of ACMCA in navigation tasks. The learnt solutions and their solution components are also presented with analyses of the proposed algorithms. Therefore, the fifth objective is achieved in this chapter.

Chapter Six concludes with a summary of the contributions, the publications of this work, and a discussion of future work.

Chapter 2

Literature Review

2.1 Introduction

This work aims to realise a cognitive architecture inspired by studies of cognitive science for a mobile robot. With this novel cognitive architecture, a robot should act as an intelligent agent. This mobile robot should be able to perform cognitive behaviours, such as perceiving the environment, storing memory, learning from past experience, and making a decision between various options. To achieve these goals, the novel cognitive architecture should have certain knowledge processing abilities, such as the acquisition, representation and use of knowledge. Inspired by emotion theories, a cognitive architecture will lead to human-like knowledge, including emotion mechanisms and emotional responses. Actually, such an affective computing cognitive architecture is a universal approach that has been developed by thousands of years of evolution among humans, mammals, and other creatures. Although the nature of the category of the evolving emotional mechanism still requires exploring, this affective computing cognitive approach is a promising methodology to perform adaptively to the environment [20] as its universal adaptation demonstrated by living creatures.

This work seeks to investigate this cognitive approach that combinates the fields of cognitive architecture, emotion theory, robotic control architecture (Figure 2.1). Therefore, chapter 2 will provide background reviews of these fields and related topics. Section 2.2 will review four cognitive systems in the field of cognitive science. These cognitive systems feature their unique architectures, producing various approaches to process

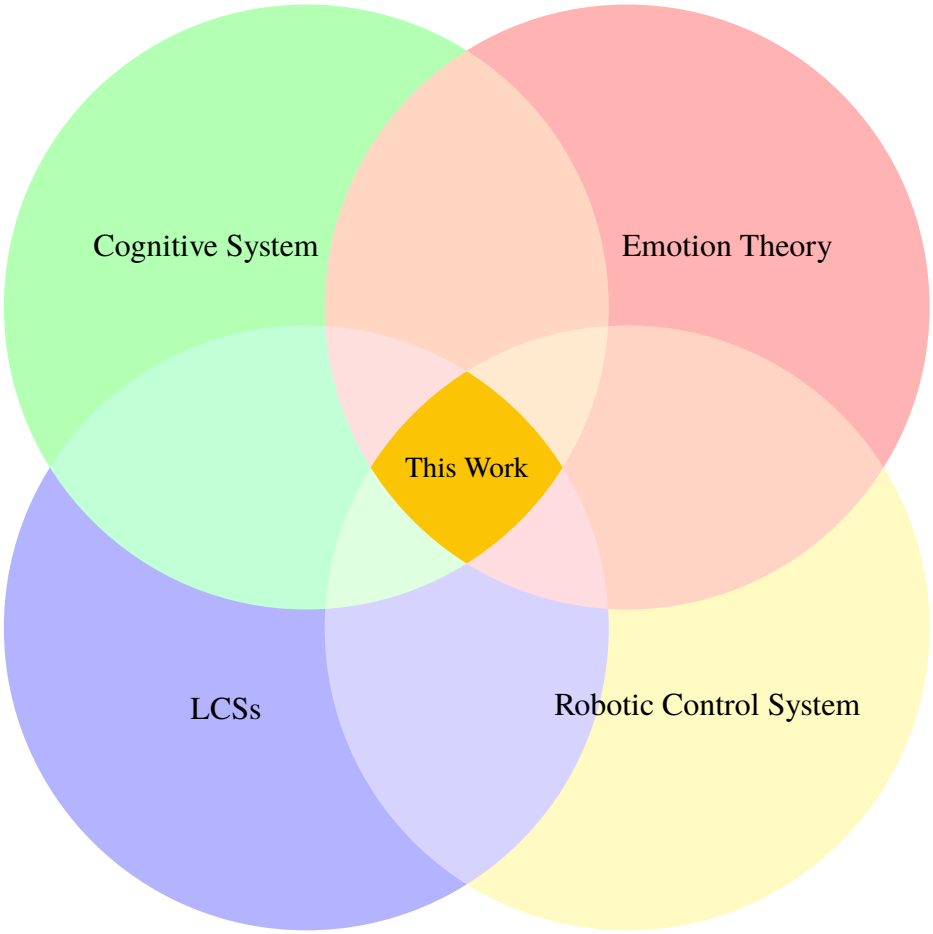


Figure 2.1: Venn diagram of this work

information. These features are inspired by the architecture design of the proposed cognitive system. Section 2.3 will discuss emotion theories, providing principles to relate the emotion and the cognitive processes together. Emotion theories attempt to explain the role of emotion and emotion mechanisms during the cognitive information process in the brain's architecture. The emotion mechanism discussed will be applied to cognitive processes proposed in this work. Section 2.4 will discuss the subsumption architecture for robotic control systems. The classical subsumption architecture composes behaviours by its multilayer architecture to achieve tasks. The subsumption approach will be applied in this work to compose solutions through the proposed architecture for the achievement of complex tasks. Section 2.5 will introduce a symbolic machine learning technique, Learning Classifier Systems (LCSs) algorithm, which is an

evolutionary algorithm originated from the field of cognitive science. This algorithm will be applied as the underlying learning technique in the proposed cognitive system. A brief description of LCSs algorithm and its iteration loop are illustrated in this section, providing the reviews for two novel variants that will be proposed for real-world robotic applications. Section 2.6 will describe the previous work, an emotion inspired cognitive architecture, which can learn an emotion model for adaptive path-planning of robotic navigation tasks. For the first time, this work evolves an emergent emotion model that can provide appropriate, non-preset affective responses during navigations. The architecture of the previous work and its underlying emotional inspirations are discussed. The previous work provides a benchmark, from which this work develops. Section 2.7 will introduce a mobile robot, the Pioneer, where the proposed contingency-based subsumption architecture will be tested. Hardware capabilities and the software operating system of the Pioneer are described.

2.2 Cognitive Systems

Studies of cognitive systems try to provide explanations for the nature of intelligence. “Cognitive science is the scientific study of the mind, the brain, and intelligent behaviour, whether in humans, animals, machines, or the abstract" according to the definition from University of California, San Diego (UCSD). Although there are many definitions of cognitive system (see appendix table 1), these definitions are common in terms of cognitive processes and related disciplines (see Table 2.1). The common topics of cognitive processes include perception, memory, reasoning, and learning. The common disciplines related to the cognitive system include cognitive psychology, neuroscience, philosophy, linguistics, and computer science. Kotseruba [21] researches at least 195 cognitive systems featured in 17 sources, and they argue that "There is no exhaustive list of cognitive architectures, their exact number is unknown, but it is estimated to be around three hundred, out of which at least one-third of the projects are currently active". Each cognitive system originates from a different set of assumptions and disciplines, represents the world by its knowledge, drives its memory system, and operates by its methodology. Therefore, diverse cognitive systems were proposed by their cognitive assumptions [21, 22, 23].

Institution	Topics of Cognitive Process	Related Disciplines	Knowledge Representation and other characterizations
University of California, San Diego (UCSD)		anthropology, computer science, psychology, neuroscience, linguistics, sociology and philosophy.	Cognitive science is the scientific study of the mind, the brain, and intelligent behaviour, whether in humans, animals, machines, or the abstract.
Princeton University	natural language, memory, problem solving, learning, vision, and reasoning	anthropology, philosophy, psychology, operations research, computer science, linguistics, and history of the science	acquisition, representation and use of human knowledge
University of Toronto, Scarborough Campus (UTSC)	perception, memory, and communication.	philosophy, psychology, computer science, and linguistics, neuroscience and anthropology	Cognitive system tackles the relations between mechanical computation and human knowing and problem solving.

Table 2.1: Comparison of different definitions of cognitive science [8, 9, 10]

General speaking, cognitive systems require an innate architecture to represent knowledge and process information. When information flows through a cognitive system, the architecture of this cognitive system provides a framework for the acquisition, representation, and use of the knowledge. This framework decomposes the knowledge processing of a cognitive system into underlying cognitive topics, such as perception, reasoning, learning, and memory. Following the decomposition, cognitive systems are

constructed based on different types of knowledge representation.

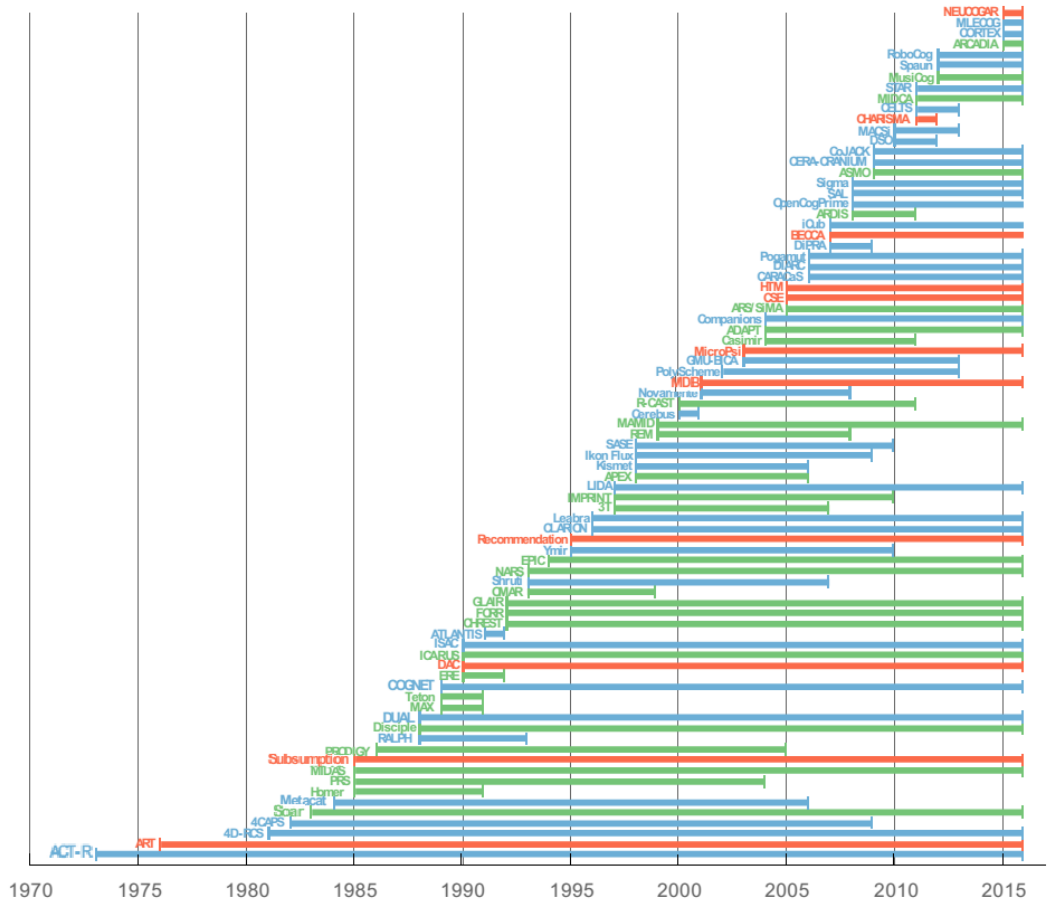


Figure 2.2: A timeline of 86 cognitive architectures [21].
Colors correspond to different types of architectures: symbolic (green), emergent (red) and hybrid (blue).

Based on the knowledge representation, cognitive systems can be categorized by three approaches: symbolic, emergent (sub-symbolic), and hybrid approaches (Figure 2.2) [21]. The first approach, the symbolic approach, constructs cognitive systems through "language-like" models. In a symbolic cognitive system, knowledge is treated as "discrete units of information that are encoded in a language-like format" [24]. This approach is characterised by the natural, intuitive way of knowledge representation. A popular symbolic model, the SOAR architecture (Section 2.2.1), demonstrates the desirable characteristic of the interpretation of a rule-based cognitive system. The "if-

then" pairs that are encoded in rule-based modules of the cognitive system is easy of interpretation [24, 25].

The second approach to design a cognitive architecture is the emergent model (sub-symbolic model). The emergent model adopts a massive parallel model where knowledge is represented by links of nodes from different layers. Compared to the rule-based symbolic system where knowledge is represented by rules, the emergent model represents knowledge through mappings among the cognitive architecture. Two examples of the emergent model will be reviewed in this section. An example is the Artificial Neural Networks (ANNs) approach. The knowledge exists in a network, which is a distributed pattern of a set of computing elements and their connections [24]. An example of the emergent model is the Multilevel Darwinist Brain (MDB) architecture (Section 2.2.2), where knowledge is distributed in a multilevel architecture. The multilevel structure of the MDB represents a hierarchy of solutions, which have the potential to achieve complex tasks, including robotic tasks. Therefore, the review Section 2.2.2 will focus on how the distributed solution components compose solution in the hierarchy.

The third approach is the hybrid model that attempts to combine the symbolic model and the emergent model. To separate the uses of these two models, a hybrid model categories knowledge into two types: declarative knowledge and procedural knowledge. Declarative knowledge is the type of knowledge stored for recognition, telling what the factor or information it is in the memory. Procedural knowledge is the type of knowledge exercised in the performance of a task. Procedural knowledge basically describes how you know to do something. A hybrid model can represent declarative knowledge by a symbolic model and can represent procedural knowledge by an emergent model [22]. An example of the emergent model is the ACT-R architecture (Section 2.2.3), which processes declarative knowledge and procedural knowledge through heterogeneous modules for various functions. The ACT-R provides a benchmark for this work in terms of applying various modules to achieve different processes of declarative knowledge and procedural knowledge.

This work will take the hybrid approach to construct our cognitive system because this approach combines the advantages of these the symbolic and the emergent approach. Because the position of this work takes the hypothesis that emotions play a critical role in the construction of a cognitive system, the following literature review of the hybrid cognitive system thus focuses on emotion inspired hybrid cognitive systems. An example

is Cognitive-Affective Personality System (CAPS, (Section 2.2.4)). Compared to the ACT-R, which applies heterogeneous modules, the CAPS applies homogeneous modules to process knowledge. In the CAPS, a module is a network that combines homogeneous nodes and various mappings between these nodes. The CAPS hypothesises that after training, various networks can emerge as different modules that can achieve cognitive processes.

2.2.1 SOAR

SOAR [1] is a cognitive architecture that has been under continuous development since the early 1980s [23]. It dynamically combines available knowledge for decision-making, and can dynamically create subgoals whenever the knowledge for a decision is incomplete or inconsistent [26]. SOAR can also compile the problem-solving in subgoals into rules, using a process called chunking, so that over time, problem-solving in subgoals is replaced by rule-driven decision-making [26]. Chunking allows SOAR to explore various learning methods, strategy acquisition and many other methods by storing away problem-solving information in a subgoal. Therefore, the decision-making process in SOAR becomes extremely versatile.

The traditional SOAR architecture contains three parts: working memory, recognition memory and a chunk (Figure 2.3) [27]. The working memory is a short-term memory, which attempts to achieve goals, whereas the recognition memory and the chunk are long-term memory. The recognition memory keeps the declarative knowledge and a chunk contains productions, which are also known as condition-action associations. The working memory consists of a problem space, which is represented by the three triangles (Figure 2.3). The problem space consists of states and operators. The state, such as an initial state or a goal state, is represented by the small circle. The operator is represented by the arrow. A set of operators apply to states and produce new states.

A step in the problem space is taken in each decision cycle (bottom of Figure 2.3). The decision cycle consists of two phases: the elaboration phase and the quiescence phase. During the elaboration phase, SOAR's learning mechanism, chunking, builds new associations in the recognition memory that are compared to the retrieving knowledge from relevant contexts. After all the associations have been executed, the system reaches

[1] SoarâĂŹs name is derived from this basic cycle of State, Operator, And Result

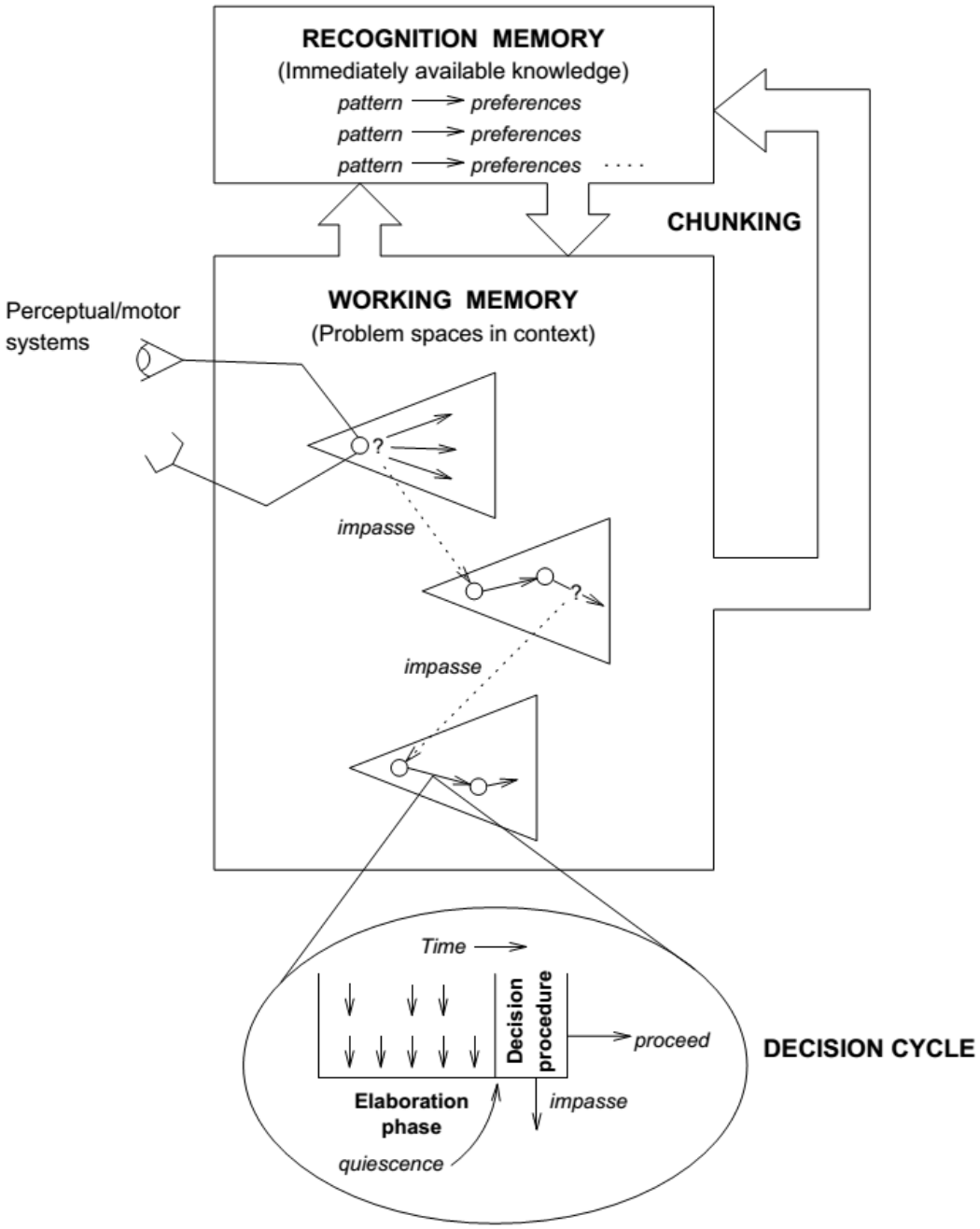


Figure 2.3: SOAR Architecture [27]

the quiescence phase, in which the retrieved preferences are interpreted by the decision procedure. The decision procedure implements the semantics of a fixed preference

language in the recognition memory and decides the next problem states. However, when knowledge about operator selection is insufficient to determine the next operator to apply or when an abstract operator cannot be implemented, an impasse occurs. In response, SOAR creates a new goal to determine which operator it should select or how it should implement the abstract operator. As a result, all tasks in SOAR are formulated as attempts to achieve goals. [27, 23].

At the learning algorithm level, the most recent SOAR module (SOAR 9) updates its architecture by incorporating new modules (Figure 2.3). These additional modules contain various machine learning algorithms to form different types of knowledge. SOAR 9 provides a general platform in which each machine learning algorithm, such as supervised learning, unsupervised learning, or reinforcement learning can be an option to choose for generating knowledge. For example, Nason integrated Reinforcement Learning with SOAR (SOAR-RL), which was implemented on a Pacman-like agent for searching food in puzzles [26]. The agent can learn preference rules of moving towards food.

As a rule-based cognitive system, SOAR demonstrates a framework of applying rules in different cognitive processes. However, the SOAR faces two challenges to be applied in real-world robotic applications. Firstly, the SOAR does not emphasise on discovering the useful rules. Instead of encouraging rule-discovery from an architecture level, the SOAR relies on the underlying algorithm's ability to discover useful rules. In contrast, this work argues that learning ability, including rule-discovery, is critical for a cognitive system who cannot ignore. Secondly, as a complex robotic application requires cooperations of different modules/rules, the SOAR framework is overgeneral and lacks specific modules to handle the required cooperations. A robotic cognitive system should consider the cooperations between multiple modules.

2.2.2 Multilevel Darwinist Brain

The Multilevel Darwinist Brain (MDB) is a cognitive architecture that employs an evolutionary approach, which provides autonomous robots with lifelong adaptation through its interaction with the environment [28, 29]. This approach adopts neuroevolution, which is an intrinsic part of the cognitive system that allows a robot to be able to learn different tasks and objectives. The overview of MDB Cognitive Architectures is described in

Figure 2.4. The MDB Cognitive Architecture consists of three models, namely, the world model (W), the internal model (I) and the satisfaction (S) model.

The world model (W) describes the external perception *e(t)*, which can be expressed as a function of the last action performed by the agent *A(t-1)*, the sensory perception it had of the external world in the previous time instant *e(t-1)* and a description of the events occurring in the environment that are not due to its actions, *Xe(t-1)* (Equation 2.1).

$$e(t) = W[e(t-1), A(t-1), Xe(t-1)] \tag{2.1}$$

$$i(t) = I[i(t-1), A(t-1), Xi(t-1)] \tag{2.2}$$

The internal perception *i(t)* of an agent is made up of the sensory information provided by its internal sensors. The internal model *(I)* describes the internal perception *i(t)* in terms of the last action performed by the agent, the sensory perception it had from the internal sensors in the previous time instant *i(t-1)* and other internal events not caused by the agent's actions *Xi(t-1)* (Equation 2.2). The satisfaction *s(t)* of the agent can be defined as a magnitude or vector that represents the degree of fulfilment of the motivation or motivations of the agent and it can be related to its internal and external perceptions through Equation 2.3.

$$s(t) = S[e(t), i(t)] = S[W[e(t-1), A(t-1)], I[i(t-1), A(t-1)]] \tag{2.3}$$

The main objective of the cognitive architecture is the satisfaction of the motivation of the agent, which, without any loss of generality, may be expressed as the maximisation of the satisfaction *s(t)* in each instant of time (Equation 2.4).

$$MAX(s(t)) = MAX(S[W[e(t-1), A(t-1)], I[i(t-1), A(t-1)]]) \tag{2.4}$$

The Multilevel Darwinist Brain applies Artificial Neural Networks (ANNs) to modify models' representations (Figure 2.4). The MDB is structured utilising two different time scales, one devoted to the execution of the actions in the environment (reactive part) and the other dealing with the learning of the models and behaviours (deliberative part).

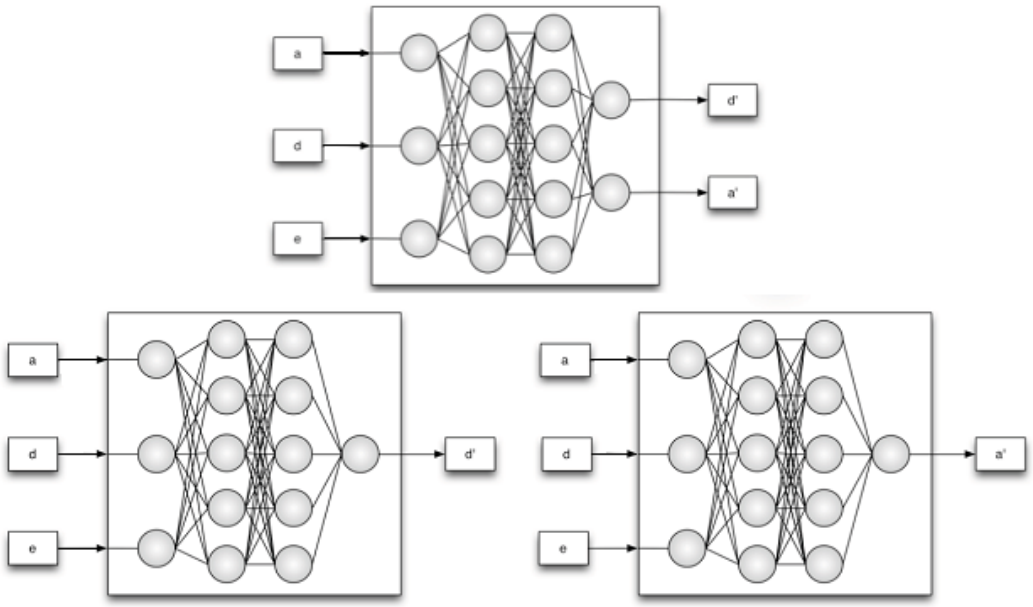


Figure 2.4: Multilevel Darwinist Brain's Architecture [29]

This separation shows ANNs can distinguish the relevance between two models. This ability can also be applied to simplify the complex network.

However, the Multilevel Darwinist Brain takes the Connectionist approach (Section 2.5) to evolve the cognitive architecture. The results are purely weights and connections. This type of result requires large extra effects to interpret their meanings before the result can apply to the real-world application without safety concerns.

2.2.3 Adaptive Control of Thought-Rational (ACT-R)

Adaptive Control of Thought-Rational (ACT-R) is a cognitive architecture, which simulates the human cognitive system as a set of independent modules [30, 31, 32] (Figure 2.5). ACT-R can be considered as a hybrid cognitive architecture for its symbolic and sub-symbolic representation [33]. In ACT-R, the symbolic representation is the declarative knowledge, and the sub-symbolic representation is the procedural knowledge [33]. The knowledge that is able to verbally describe or declare is considered declarative knowledge, whereas knowledge that can only be inferred from an individual's behaviour is considered as procedural knowledge [34]. Intuitively, the relationship between declar-

ative knowledge and procedural knowledge is similar to that of the data and the program. A chunk is the basic unit of knowledge in the declarative memory, and the production is that of knowledge in the procedural memory. The chunk tries to specify what it is, while the production provides an appropriate method to processes it. ACT-R is built on the declarative-procedural distinction, whereas other cognitive systems, such as SOAR, blurs this distinction.

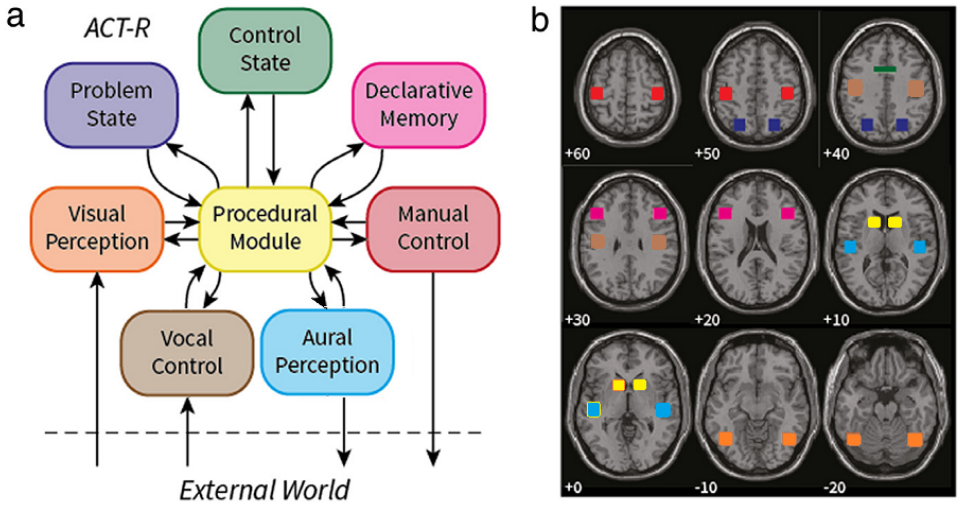


Figure 2.5: The ACT-R cognitive architecture
(a) Its apping to brain regions (b). The colours of the modules correspond to the coloured squares in the brain [30].

ACT-R processes information through a set of modules, each of which processes a different type of information. Each module has an associated buffer, a chunk, which holds a relational declarative structure. Productions respond to the chunk by applying a rule whose condition part matches the chunk and whose action part initiates what modification should be done in that buffer. The modification may involve commands such as retrieving a chunk from long-term declarative memory or executing a motor command.

Schumacher et al. reported an experiment in which participants performed a visual-manual task and an auralâĂŞvocal task (Figure 2.6) [32]. The following steps demonstrate the process of how the ACT-R architecture responds to the visual input with a figure movement command. 1. When the visual module detects a visual stimulus, it updates

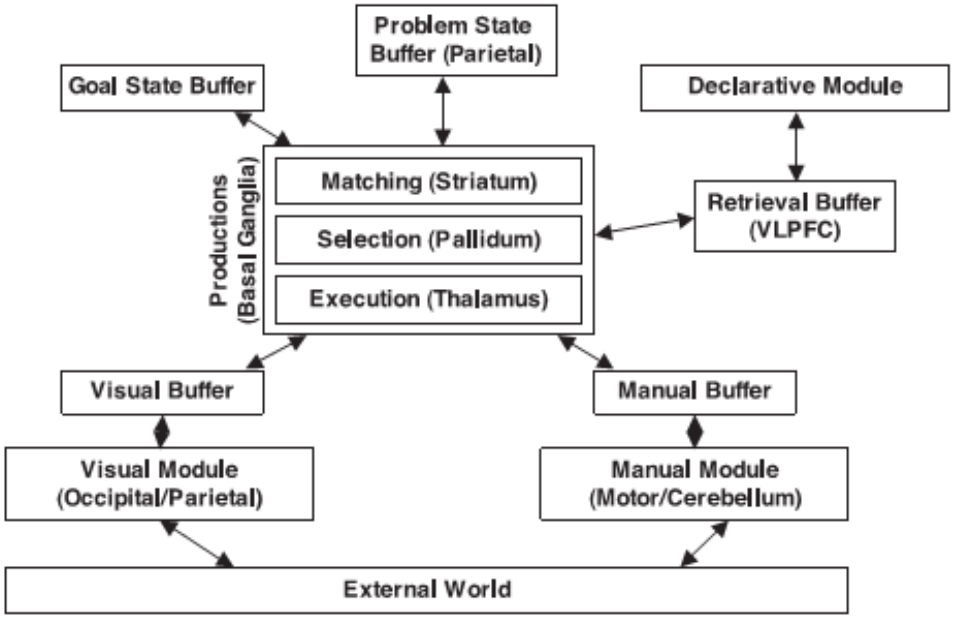


Figure 2.6: Overview of the ACT-R architecture [35]

the visual buffer. 2. Productions notice this update, and a production rule (compiled-attention-rule) puts a request to indicate this visual stimulus in the visual buffer. 3. The visual module finds the pattern to this visual stimulus and places it in the visual buffer. 4. This pattern in the visual buffer will elicit another production rule (extract-index-finger) that will put a request to press the index finger in the manual buffer. 5. When the manual module is done, the rule "Ready" terminates the trial. This experiment demonstrated the ACT-R architecture's potential to be applied to a real-world robot.

The ACT-R community has used its architecture to model a variety of phenomena from the experimental psychology literature, including aspects of memory, attention, reasoning, problem-solving, and language processing. However, the ACT-R architecture relies on the preset modules that are activated in a fixed sequence. The ACT-R architecture may be more adaptive if it can improve its performance in two fields: the knowledge acquiring skill and the decision process. The heterogeneous modules of the ACT-R architecture limit its ability to acquire knowledge. The ACT-R architecture is difficult to automatically generate new modules based on the existing heterogeneous modules. It cannot also generate new behaviours based on the current decision-making processes.

2.2.4 Cognitive-Affective Personality System (CAPS)

Three cognitive architectures have been discussed in previous sections. These cognitive architecture don't emphasise on the role of emotion in their cognitive process. However, emotion might gravely engage in cognitive process consciously and unconsciously [36]. In the field of emotion theory, an affective cognitive hypothesis advocates that an affective cognitive system can achieve solutions by its capability of reconceptualizing situations, dispositions, dynamics, and invariance in personality structure [37]. In a task scenario, relevant capabilities toward task solutions are diverse in terms of encodings, expectancies and beliefs, affects, goals and values, competencies and self-regulatory plans (see an example in Figure 2.7). Generally, these capabilities are upheld by computing unites in the affective cognitive system. Therefore, each computing unite represents a component of the task solutions, and the network of the total unites represents the completed solution. At a given moment, these units are activated or deactivated to execute a solution for the task. As diverse computing nodes are able to represent the components of the task solutions, an affective cognitive system has the potential to handle every admissible task.

An affective cognitive system also has the potential to learn "appropriate" components of the task solutions. Ideally, instead of presetting the task solutions and their components, the computing unites of an affective cognitive system can apply learning agents (i.e, RL agents) to learn these components through the RL approach. That is, the learning agent of each component aims to search the appropriate context of this component. For example, a learning agent can be deployed to search the optimal value of hyperparameter of a path-planning model. In this case, the hyperparameter is the targeted component, and the learning agent aims to search the optimal value from a space that contains all the admissible values. Therefore, the potential of learning diverse components of a task solution exists if all the learning agents can learn effectively.

However, all the learning agents cannot learn effectively if they face a large searching space of the solutions. The searching space of the solutions is constituted from the searching spaces (sub-spaces) of the components of these solutions. Mathematically, the operation of the constitution of sub-spaces can generate a combinatorial explosion of the constituted space. As components diversify, the searching space of the solutions approaches to almost infinite exponentially. As a result of an enormous searching space,

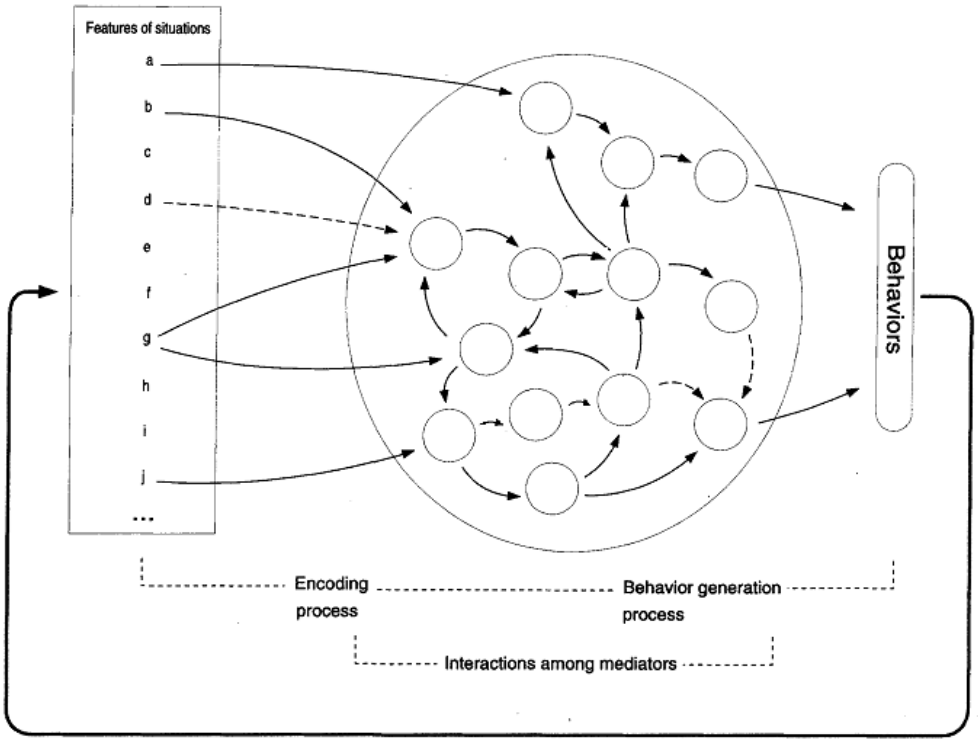


Figure 2.7: Cognitive-affective mediating processes [37]

learning agents incline to be failed in learning components.

This work argues that an affective cognitive system should require an architecture that innately decomposes the searching space of the solution. A hierarchy structure is a promising architecture because it can introduce innate subsuming operations to the system. Similarly to the subsumption system [38], the subsuming operations allow components in the higher-layer of the architecture to subsume components in the lower-layer by their symbolic meanings. The subsuming operations can diminish the entire searching space of the solutions into sub-spaces, facilitating the embedded learning agents. Therefore, subsumption between components of different layers should be necessary for an affective cognitive system to learn solutions with diverse components.

2.2.5 Summary

Before designing our affective cognitive system for robotic applications, this section reviewed benchmarked cognitive system in the field of psychology. Four cognitive

systems were introduced in this section: SOAR, Multilevel Darwinist Brain, Adaptive Control of Thought-Rational, and Cognitive-Affective Personality System. These four cognitive systems cover all three types of knowledge representation: symbolic, emergent, and hybrid. Compared to the other three cognitive systems, the hypothesis of Cognitive-Affective Personality System are more related to emotion theory, which plays a critical role in the hypothesis of this work. Thus the next section will introduce three mainstream emotion theories.

The number of emotion inspired cognitive systems that are applied for robotic navigation tasks is rare, although hundreds of cognitive systems [21] have been proposed. For the first time, the previous work (Section 2.7) proposed an emotion inspired robotic cognitive system, which can evolve an emergent emotion model and generate appropriate, non-preset affective responses during navigations [19]. Because of the shortfalls of the previous work, it is necessary to propose a novel cognitive system, which is inspired by the insights from the fields of cognitive systems, emotion theories and robotic control systems.

2.3 Emotion Theories and Brain's Cognitive Architecture

2.3.1 Introduction to the Nature of Emotion

Emotions can be thought of as states elicited by rewards or punishers [39]. Generally speaking, "a reward is anything for which an animal will work, whereas a punisher is anything that an animal will work to escape or avoid, or that will suppress actions on which it is contingent" [40]. By approaching rewards and avoiding punishments, emotions are elicited in cognitive processes (i.e. decision-making), leading the execution of actions for goals. In neuroscience, various human brain areas (such as orbitofrontal cortex (OFC), amygdala, and the anterior cingulate cortex) fire during these cognitive processes, suggesting a complex brain mechanism existing in emotion affect cognitive processing [40, 41, 42]. Therefore, investigation of the architecture that underlies affective cognitive processes has the potential to create an autonomous robot that can pursue rewards and avoiding punishments due to elicited emotion states.

2.3.2 Emotion States and Cognitive Architecture

Cognition is closely related to emotion. Every type of cognitive operations may be involved in processes which determine the goal for an action. In cognitive operations, a certain event, stimulus, and memory will be involved to evaluate the agent related event in terms of rewards and punishments. Thus, emotions as states will be provoked in the cognitive process [40]. Under this circumstance, emotion can also be used as the "umbrella" concept that includes affection, cognition, behaviour, expression, and a host of physiological changes [20]. The hierarchy of mind, which include emotion within a cognitive architecture, is worth exploring because it can inspire the methodology for this work (Section 3.2). At the molecular level, the brain's emotional-affective hierarchy has three levels, namely, the primary process level, the secondary process level and the tertiary level (Figure 2.8) [43, 40]. Investing emotion machines in these three levels will provide reasonable presumptions for the proposed methodology.

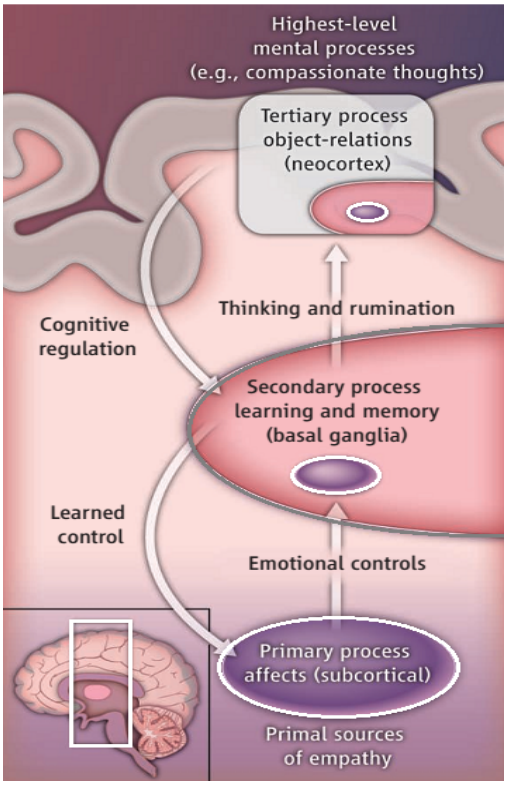


Figure 2.8: Nested brain-mind hierarchies [36]

2.3.2.1 Primary Process Level

The primary process level takes place in the sub-neocortical region of the brain. This level contains an unconscious perception system and an autonomic endocrine system [2]. The perception system can perceive the external stimulus, transfer them into perceived fact, then pass this declarative knowledge onto the next brain level and activate the endocrine system at the same time. The endocrine system will prepare the body for action by releasing chemicals. For example, the endocrine system will release adrenaline when a snake-phobia patient sees a python [40]. The perceived fact can be considered as the "cold" declarative knowledge, which will be stored in the short-term memory. Conclusively, the primary process level aims to perceive all the facts, which will be transferred to the secondary process level for further processing.

2.3.2.2 Secondary Process Level

The secondary process level receives facts from the primary-processes level, and processes the information in the basal ganglia and cingulate cortex of the brain to obtain rewards and punishments unconsciously or implicitly.

(1) Stimulus-response behaviour. The basal ganglion deals with facts, which are the results of perceptions of external stimuli in the primary process level, to learn the stimulus-response behaviour or activate such behaviour that "Nature" built into our brains. The behaviour triggered by the basal ganglion will directly respond to certain stimuli unconsciously. This process does not require an intervening state such as a goal to be reached [40]. For example, the hand will retract immediately once it touches a hot surface. Only minimal explicit consciousness is involved in stimulus-response behaviours.

(2) Stimulus-reinforcer behaviour. Another process on this level is the cingulate cortex process that supports the stimulus-reinforcer association behaviour, which will connect the personal relevance in this process and thus provoke emotions. With the rewards and punishments, the cingulate cortex process transfers the

[2]The endocrine system is the collection of glands that produce hormones that regulate metabolism, growth and development, tissue function, sexual function, reproduction, sleep, and mood, among other things.

> "cold" fact into the "hot" declarative knowledge, which is also known as appraisal. The hot declarative knowledge appraisal will define the personal significance of an encounter for well-being [44]. As a result, the appraisal is a proximal variable which directly influences whether an emotion will be generated, and if so, its type and intensity.
>
> In the stimulus-reinforcer learning process, the fact is evaluated by what happens beneficially or harmfully to the agent itself. The fact with such anticipated knowledge is transferred into appraisal knowledge by the stimulus-reinforcer learning process. As a result, each positive emotion reflects a particular kind of appraised benefit, and each negative emotion reflects a particular kind of appraised harm [44].

Classical conditioning might be generated by this stimulus-reinforcer learning process. The classical conditioning, which is also known as Pavlovian conditioning, originates from the legendary experiments conducted by Ivan Pavlov (1849-1936) [39]. In one of Pavlov's experiments, also called Pavlov's dog, the sound of a bell would connect with dog's prediction that food is coming. After training, the dog will salivate when the bell rings. The food is considered to be the unconditioned stimuli (US) that are the true rewards, whereas the bell is the conditioned stimuli (CS). In the stimulus-reinforcer learning process, the fact, which is the sound of the bell in this case, has been evaluated as beneficial by the reward that is food. With the evaluation, the fact has transferred into appraisal knowledge. This appraisal will implicitly provoke emotion and thus activate the responsive behaviour. In other words, the ringing bell (CS) is associated with the dog's positive emotion and triggers its slobbering in this case. During the stimulus-reinforcer process, the long-term memory stores the declarative knowledge including fact and appraisal in the secondary-processes level.

However, if the bell keeps ringing without food being provided, the dog will establish a new stimulus-reinforcer association to replace the old one, and stop salivating. In this new stimulus-reinforcer learning process, the new appraisal knowledge will be learned. In other words, the new appraisal knowledge will be established, while the old appraisal knowledge of the bell will be inhibited but is still in the memory.

Another example that shows the old appraisal knowledge will be inhibited by a new one is the new PTSD (Post-Traumatic Stress Disorder) therapy. Originally coming

from the military, PTSD often happens to people who have experienced life and death scenes. Those people who over-learn emotional memory (the appraisal knowledge) might develop PTSD. Those who suffer from PTSD might have flashbacks, nightmares or severe psychological trauma, which might cause them unable to function normally in daily life. The new therapy is to provide a drug to overcome the overlearning of the old emotional memory by increasing the learning of new memory. The new drug increases the activity of the receptor, which is similar to the gate for neurons and cells. Therefore, the drug can increase the learning of new memory so that the good new memory will inhibit the old one that provokes too much fear. This new PTSD therapy provides more insights for the stimulus-reinforcer behaviour and the stimulus-reinforcer learning process through exploring the interaction between memory and appraisal [45, 46].

Inspired by the stimulus-reinforcer learning process, this work will introduce secondary reinforcers (Section 3.2.2) into the proposed cognitive system. A secondary reinforcer attempts to achieve stimulus-reinforcer learning, which connects primary reinforcers (Section 3.2.1) with optional actions and their consequential outcomes. Different stimulus-reinforcer learnings will be achieved through subsumption operations on secondary reinforcers (Section 3.3.2).

2.3.2.3 Tertiary Process Level

The tertiary process level consciously processes four topics in the Neo-cortical area of the brain: knowledge, emotions, goals, and goal-directing behaviours. The process on this level is conscious; the agent is aware of these four topics. This is different from the secondary process level which "has no control over whether the unconditioned stimulus is delivered" [40]. In terms of knowledge, which is a topic at this level, some processes related to knowledge are clear: the declarative knowledge will be processed in the tertiary process level. The processing progress of declarative knowledge includes generating both fact and appraisal, storing them in both short term memory and long term memory, and retrieving them from long term memory.

However, the interactions between these four topics are still not clear. There is also no agreement of how conscious emotion would be elicited by or affect the other three topics [47]. Traditionally, there are three emotion theories to explore this question: Basic Emotion Theory, Appraisal Theory, and Constructive Theory. These three emotion

theories will be discussed below, following the framework of Constructive Theory. This framework may not strictly match paradigms of Basic Emotion Theory and Appraisal Theory: Basic Emotion Theory does not have a clear definition about the unconscious emotion; and Appraisal Theory does not distinguish the relationship between behaviour and conscious or unconscious emotions. The principal ideas of these three theories are as follows:

(1) Basic Emotion Theory. Basic Emotion Theory, which is also known as discrete emotions theory, is inspired by Tomkins's (1962) rediscovery of Darwin's (1872/1998) work on the expression of emotion. "The fundamental assumption is that a specific type of event triggers a specific affect programme corresponding to one of the basic emotions and producing characteristic expression patterns and physiological response configurations" [48]. According to the theory, this basic emotions are "joy", "fear", "anger", "disgust", "sadness" and "interest" [49]. Panksepp follows this discrete emotions theory. His emotion model is slightly modified from these seven basic emotions, and advocates the seven basic affective systems according to each basic emotion [50, 51]. These are SEEKING for enthusiastic, RAGE for pissed-off, Fear for anxious, LUST for horny, CARE for tender and love, PANIC for lonely and sad, PLAY for joyous. These basic affective systems are executed in a certain part of the brain to activate the behaviours. For example, the SEEKING system is associated with anticipatory-appetitive behaviours and is driven by activation of several neurological structures, including the ventral tegmental area (VTA) and lateral hypothalamus (LH) [33, 43]. Generally speaking, this approach treats emotions as fixed action patterns elicited as motor responses to stimuli [40].

(2) Appraisal Theory. Appraisal Theory generally treats emotions through the assessment from four aspects: relevance, implications, coping potential, and normative significance [40]. They advocate adaptive response by "defining the adaptive functions in terms of the efferent results of individual appraisal checks that add up cumulatively to prepare appropriate action tendencies" [48]. This theory treats emotion as a reaction to significant events [40]. As a result, the agent might have actions' readiness [3] and select an optimised action to different types of alternatives.

[3] Actions' readiness is "a state of preparedness for action that is elicited as part of an emotional response

(3) Constructive Theory. Constructive Theory considers emotions as states, which are critical in brain design. The emotion mechanism suggests a way from stimuli to states, "in which a goal must be held in mind as the target for an instrumental action to direct behaviour" [40]. The emotional states are represented by the valence/arousal affect space. This core affect space provides a theoretical foundation to quantify every emotion state in a reasonable way [47]. The goal-directed actions are realised by the instrumental learning, in which there is a contingency between the behaviour and the reinforcing outcome. The behaviour may be said to be goal-directed if it depends on both (1) the instrumental contingency between the action and a particular outcome (Action-Outcome (AO) contingency), and (2) a presentation of the outcome as a goal [40, 47].

If an agent has AO contingency, it can learn the instrumental contingency between an action and its consequence. Whereas emotions are directly linked to models in Basic Emotion Theory or to functions in Appraisal Theory, they are linked to "labels" in Constructive Theory. This theory mentions behaviours as the declarative knowledge rather than the procedural knowledge. In other words, it does not define how to carry out behaviours in AO contingency. The benefit of symbolic behaviour is that it might increase the flexibility of agents in the actual actions that are produced [53]. Yet, it requires another effort to link the behaviours' declarative knowledge to its procedural knowledge.

Constructive Theory pays much more attention to the learning process than the other two theories because the agent acquires the instrumental learning process to establish the AO contingency [53].

The instrumental learning process introduces an incentive value to demonstrate the linking strength between stimuli and behaviours. The incentive value is directly linked to motivation. The decreasing incentive value will weaken the motivation, and thus affect the subsequent behaviour [53].

While the incentive value tells how strong a desire is for the agent, the hedonic assessment demonstrates the agent's attitudes toward different stimuli. For example, one can rate a monkey's acceptation to food by measuring whether the monkey will open its

and associated with such physiological indicators as changes in heart rate, respiratory rate, and muscle tension" [52].

mouth ready to accept the food, whether he will swallow food being placed in the mouth, or whether he will use his hand to push away the food [53].

Discriminative stimuli are introduced to justify the conditional relationship of the stimuli or the behaviour. The agent must discriminate between different stimuli to respond to them in different ways. For example, a S+ is a discriminative stimulus which indicates an agent that reinforcement is available, while a S- is a discriminative stimulus which tells an animal reinforcement is not available. Then the agent will learn to approach a S+ and avoid a S- [54]. Discriminative stimuli have been frequently used to study brain mechanisms involved in processing reward-related visual and olfactory stimuli [53]. This suggests that instead of being bias to positive rewards and ignoring the negative reward, both positive and negative rewards can be applied in the cognitive process.

2.3.3 Summary

To conclude, this section has provided a brief definition of emotion and an introduction to a three-level cognitive structure where emotion states are elicited.

The primary process level focuses on the perception of the external stimuli for the future cognitive process. The input of the environment for the cognitive agent is transferred into fact, which will be stored in the short-term memory as one type of declarative knowledge. This level also controls the endocrine system for the body's behaviour preparation.

The secondary process level advocates two types of learning behaviours, the stimulus-response behaviour and the stimulus-reinforce behaviour. The stimulus-response behaviour provides a quick, automatically responding mechanism for the cognitive agent to react in an emergency. After the link between fact (stimuli) and behaviour model is established, the agent directly reacts to the fact without explicit consciousness. In contrast, the stimulus-reinforce behaviour learning process will label fact with emotion according to the personal significance. This process will transfer fact into the appraisal. The stimulus-reinforce behaviour learning process might be the fundamental mechanism of Pavlovian conditioning. This learning process not only creates the appraisal but also interfaces its intensity. This idea is demonstrated by the principle of new PTSD therapy. The stimulus-reinforce behaviour also provides insights about how emotion interacts

with declarative knowledge.

Three traditional emotion theories are introduced in the tertiary process level to explore the interaction between knowledge, conscious emotion and behaviours. Basic Emotion Theory explores the interaction by defining a prototypical reaction, which includes a specific action tendency, physiological response pattern, motor expression and feeling state. Appraisal Theory indicates that emotional states can be elicited by stimuli. Constructive Theory considers emotions as estimations of primary stimuli, thus transfers these primary stimuli into secondary stimuli by attaching the estimations.

This section has reviewed the study in neuroscience and psychology regarding emotion and the brain's cognitive architecture. The brain's mechanism provides insights to construct an affective computing cognitive system for an agent. Firstly, an emotion mechanism indicates an agent's evolutionary result. An emotion mechanism is a dynamic mechanism, which is generated through the agent's interactions with an environment. Secondly, emotion mechanism may contain a certain hierarchy in which each layer may focus on a specific topic of a cognitive process. Therefore, the emotion mechanism may contain various specific mechanisms in each layer to solve a cognitive topic, such as decision-making. Thirdly, generated emotion mechanisms are diverse and universal. The diversity in environments and agents leads to diversity in emotion mechanism. Yet, the universal hierarchy in cognitive process indicates universal underlying principles (mechanisms) of diverse evolutionary results.

These reasons also indicate the necessity of introducing an evolutionary approach to construct an emotion model in a cognitive architecture. Section 2.4 will review robotic subsumption architectures, which can be applied to automatically construct an emotion model with underlying machine learning techniques (Section 2.5).

2.4 Subsumption Architecture

A completely autonomous robot would include complex functional modules for task completions in uncertain, dynamic real-world environments [55]. Traditionally, a control system for an autonomous mobile robot is decomposed into functional modules by the sense-plan-act (SPA) topics: proception, modelling, planning, task execution, and motor control. Under this functional decomposition, each functional module attempts to achieve individual topic, and all modules combined together as a control system for

tasks (see Figure 2.9). As shown in the figure, functional modules activate by a sequence of topics to complete tasks.

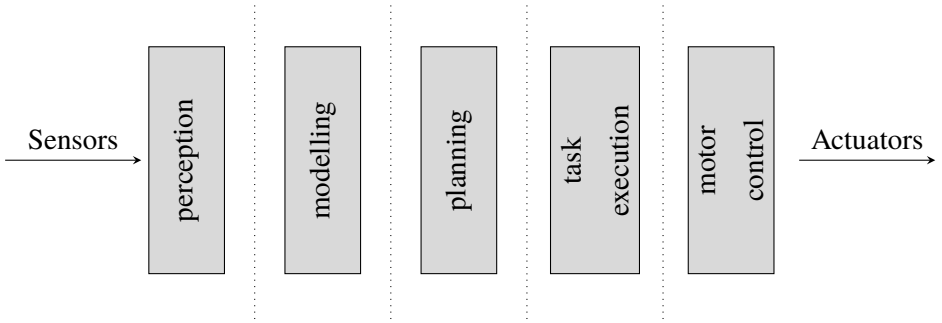


Figure 2.9: A traditional decomposition of a mobile robot control system into functional modules [55]

However, this functional decomposition might introduce human bias thus causing problems. Firstly, the functional decomposition ignores the potential innate interactions and impacts between these modules. For example, an motor execution without involving sensors could be dangerous in a dynamic environment. The knowledge that is learned by a functional module is hard to be transferred to improve other modules. Secondly, the system's overall performance is constrained by the sequential firing chain of functional modules. As one systemic output requires a sequential firing chain of functional modules, the system's robustness is much inferior to each functional module's robustness. One bug in a module might cost the performance of the entire system.

The subsumption architecture proposed an alternative control system for a robot. Compared to the approach of the functional decomposition of a control system, a subsumption system decomposes itself by layers of behaviours (see Figure 2.10). A layer of behaviours is a behaviour-based Finite State Machine, which is a control loop that connects perceptions to action behaviours (see Figure 2.11). As high-level behaviours can override low-level behaviours, hierarchical control loops compose a subsumption system. As behaviours in the lowest layer directly connect sensors and actuators, a subsumption system can optimise its behaviours through a hierarchy of Finite State Machines to interact with the environment. Compared to a functional decomposition system, this feature increases the robot's flexibility in its behaviours for task completions. Besides, the subsumption system can react to a dynamic environment more quickly than

the alternative system, because the subsumption system constantly perceives the environment. The subsumption architecture is developed with hierarchical components to

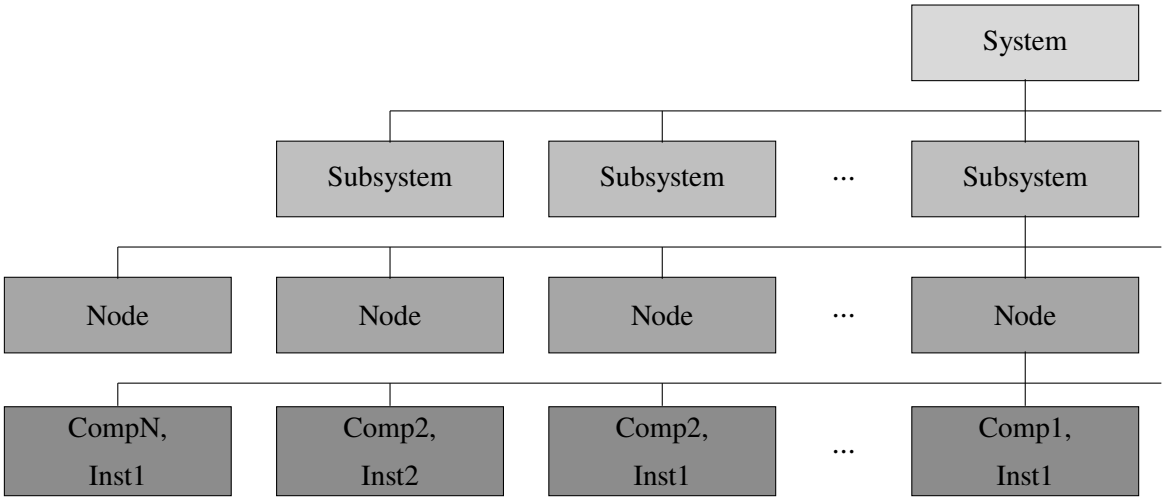


Figure 2.10: A layer of a subsumption system based on task achieving behaviours [55]

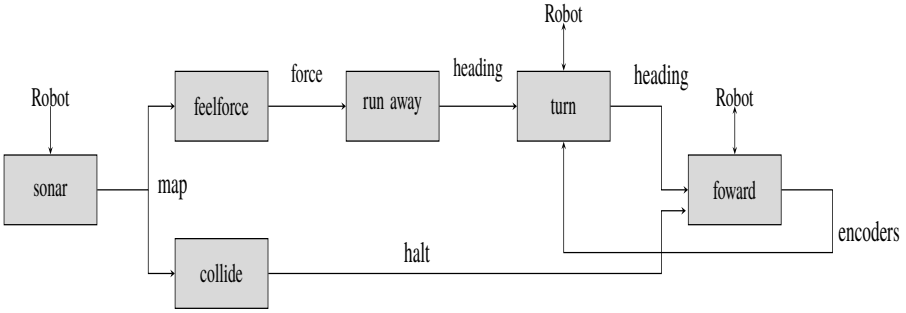


Figure 2.11: Example layer of subsumption system [55]

achieve goal-directed behaviours. These hierarchical components attempt to optimize a robot's planning toward goal achievement. Hierarchical components, such as planning components and learning components, are thus introduced into the classical subsumption system. Autonomous Robot Architecture (AuRA) [56] is an example of a combination of hierarchical components with Finite State Machines [4] (see Figure 2.12). The hierarchical components, including mission planner and spatial reasoner, aim to achieve symbolic reasoning in the high-level of the architecture to generate goal-directed behaviours in

[4]The Finite State Machines in the AuRA are called finite-state acceptors (FSAs).

the low-level. Through hierarchical components, a subsumption system is able to perform goal-directed behaviours rather than initiate reactive behaviours. Machine learning

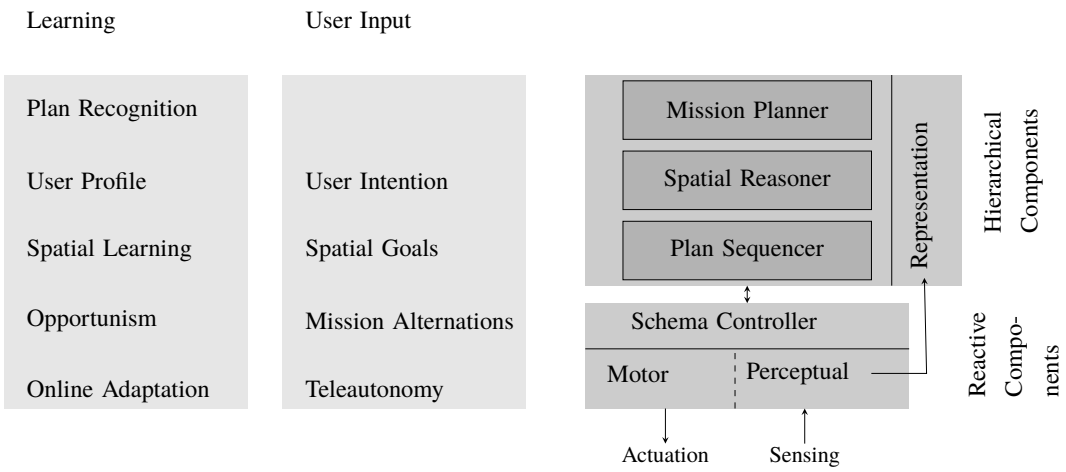


Figure 2.12: High-level AuRA schematic [56]

techniques are introduced to the subsumption system to optimize the robot's behaviours. Traditionally, Finite State Machines (FSMs) establish relatively fixed innate connections from sensors to actuators in the classical behaviour-based subsumption system. These fixed connections make the classical system almost impossible to optimize its behaviours [57]. Therefore, hierarchical components are introduced to improve the system's capability in behaviour optimization.

For example, Arruda introduces Dynamic Fuzzy Cognitive Maps (DFCMs) with Reinforcement Learning into the classical behaviour-based subsumption system, termed DFCMs-based subsumption system [58]. In the DFCMs-based subsumption system, DFCMs replace the FSMs, becoming hierarchical components that affect innate connections of the system. A learning system applies the Reinforcement Learning technique to train these connections through interactions with the environment (see Figure 2.13). As a system that is inspired by cognitive models of marine life [57], the DFCMs-based subsumption system also introduces an explicit internal state system, which aims to simulate motivation and homeostasis system for physiological processes. Compared to the behaviour-based subsumption system, the DFCMs-based subsumption system has shown the ability to learn and adapt its behaviours for the task in a simulation environment.

Although subsumption systems are based on the hypothesis that achieving behaviours

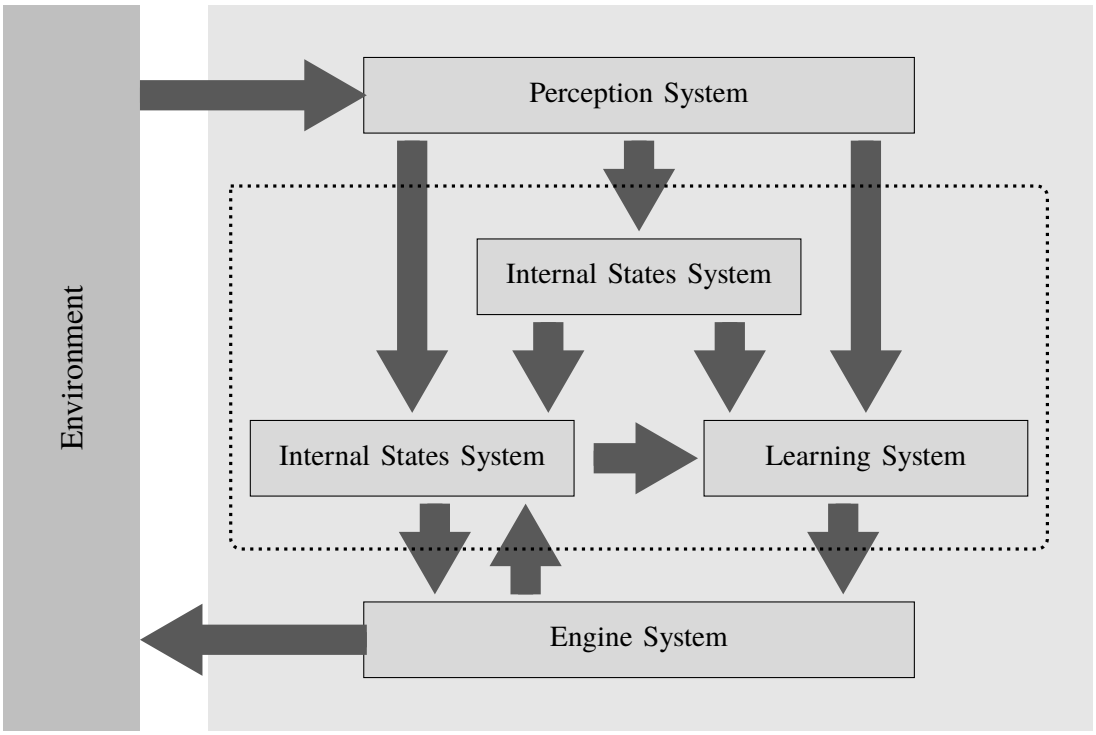


Figure 2.13: Architecture of DFCMs-based Subsumption System [57]

can compose a robot's control system through subsumption, it proved very difficult to compose behaviours to achieve long-range tasks [57]. One potential reason is that the FSMs approach to behaviour-based subsumption essentially preset a finite behaviour space, which has no guarantee of the existence of achieving behaviours. As a result, "no emergent or unexpected behaviours" would be composed by the system [59], which indicates the limitation of the FSMs approach. Introducing a huge behaviour space (even an almost infinite behaviour space) would increase the flexibility of behaviours. But the huge search space brings challenges for the underlying machine learning techniques to subsume behaviours effectively.

Another reason is that planning and reasoning are separated from the composition of behaviours. As the system's interactions with the environment fail to directly impact the behaviours' composition in each layer, the behaviours' compositions have no reason to move toward task achievement. In a huge behaviour space, the machine learning algorithm should guide these behaviours' compositions evolving toward task achievement.

Additionally, the architecture with a high hierarchy structure might create a huge

search space for the behaviours' composition. A total search space explodes exponentially as layers pile up in the architecture. Even the search space contains successful behaviour combinations, machine learning techniques still require significant computing resource for this search, bringing practical challenges for real-world robotic applications.

2.5 Learning Classifier Systems

Existing machine learning techniques can be categorised into five tribes: Symbolist, Connectionist, Evolutionaries, Bayesians, and Analogizers [60]. Each tribe has a major advantage over others. Symbolist tends to compose knowledge into semantic rules that can be easily interpreted. Connectionist technology is good at reward backpropagation, so is adaptable to Reinforcement Learning tasks. Thus Connectionist approaches are popular among robotic paradigms for learning from the past. Evolutionary technologies, such as Genetic Algorithm, contain optimization in an evolutionary process. Bayesians focus on handling uncertainty using probabilistic approaches such as the Bayesian approach. The Analogizers approach focuses on techniques to match bits of data to each other. For the evolutionary cognitive robot, evolutionary technologies/algorithms are selected for the application. This is because the evolutionary capability would allow the robot to autonomously evolve solutions for the task with the symbolic interpretation of the final solution, which is desirable in the field of robotics.

Evolutionary Algorithms are search algorithms inspired by the Darwin's theory of natural selection [61, 62, 63]. This theory assumes that natural competition allows the transmission of hereditary benefits to pass through generations. Therefore, evolutionary algorithms need many iterations to allow individuals of a population to evolve over a number of generations. During each iteration, feedback from successful behaviours is used to select appropriate "parent" individuals to generate "children" individuals. In this process, genetic operators, such as crossover and mutation, are applied to evolve children from parents [6]. The crossover operator generates one or more child individuals from the combinations of several parents, and the mutation operator alters an individual independently of parents. Therefore, evolutionary algorithms allow the evolving population to search for solutions to complex problems.

Machine learning techniques have three different learning paradigms: Supervised Learning, Unsupervised Learning, and Reinforcement Learning. Supervised Learning

trains a model with a labelled dataset, which creates a learning scenario where a "teacher" is available. The output of the model is a prediction guided by the labelled data. In contrast, Unsupervised Learning trains its model based on unlabeled data, which creates a learning scenario without any intervention from a "teacher". The output of the model is based on the dataset's pattern that is discovered in the training process. Reinforcement Learning has an agent that can interact with its environment by performing its actions. Thus the agent learns "how to perform behaviours" from errors or rewards. This is similar to a learning scenario where the agent can obtain a score from an environment according to criteria from an "unavailable teacher". As the agent can exploit or explore the environment, the model can be trained with no predefined data. This learning by trial-and-error approach makes Reinforcement Learning popular in the field of robotics [64].

Learning Classifier Systems (LCSs) are rules-based evolutionary algorithms that cover many characteristics of machine learning techniques [65]. An LCS agent contains a population of classifiers, where a classifier is a "condition-action" rule with statistics (i.e, predicted-reward, fitness). The "condition-action" representation of the rules facilitates their symbolic interpretation, becoming the primary characteristic of this algorithm. The evolutionary capability that drives both global and local search for improved rules is the second characteristic of LCSs algorithms. Evolutionary technologies, such as Genetic Algorithms, are applied in the methods that generate classifiers, enabling the entire population of classifiers to move from initial population towards an optimal one. Expansibility is the third characteristic of LCSs algorithms. LCSs algorithms provide an evolutionary framework to generate optimal rules without strict restrictions on its application. For example, various data-niche representations can be introduced into the LCSs algorithms to meet the requirements of different applications. For encoding the condition of a problem instance, the tree structure can be introduced in the algorithm's representation (i.e, XCSCF [66]), which leads to the induction capability to represent the environment in a structured way. By converting Multi-Layered Perceptron (MLP) neural networks [67] into the representation, Accuracy-based Neuro and Neuro-Fuzzy Classifier Systems (X-NCS) [68] can apply the connectionist technology to learn the function approximation from the input to the output. Thus, the characteristic of expansibility will facilitate this proposal of novel variants of LCSs algorithms for robotic applications. Therefore, this work selects LCSs as the major underlying machine learning technique

for this research application.

Generally, LCSs can be categorised into Accuracy-based Learning Classifier Systems and Strength-based Learning Classifier Systems, according to the method used to calculate *fitness*. Fitness is a statistic that estimates the quality of a rule through its *predicted reward*. Accuracy-based Learning Classifier Systems (e.g. *XCS* [69]) calculate the fitness based on the accuracy of the prediction of reward, and Strength-based Learning Classifier Systems calculate the fitness based on the strength of the received reward [70]. XCS rules attempt to move towards the most general, accurate rules because of the pressure that is generated by the accuracy-based fitness coupled with subsumption and deletion pressure.

In this work, we choose XCS as the benchmark algorithm to apply to the proposed cognitive architecture for three reasons. The first reason is that a robot is preferred to be consistent in its performance compared to the opposite case of performing excellently in many trials, but crashing in others. This argues for an Accuracy-based Learning Classifier System. Secondly, solutions that a robot learns from a less frequent event will not be overwhelmed by solutions learned from a much more frequent event, which argues against Strength-based Learning Classifier Systems. The third reason is that “correct" actions are generally unknown in a robot’s training process. So a “teacher” is unavailable. In the learning process, XCS adopts the reinforcement learning approach, which will allow the robot to take actions in the environment and learn solutions to improve its performance [71, 72]. The robot’s training process makes XCS more suitable for this robotic application than UCS (sUpervised Classifier System) [73, 74], which is designed for supervised problems where the “correct" actions are available in the training process [75].

2.5.1 Standard XCS Learning Iteration

A standard XCS agent estimates the fitness of rules through iterations of interactions with the environment. Based on how the agent perceives the environment in each iteration, rules in the XCS agent will advocate an action as the agent’s effect on the environment (see Figure 4.3.a). All rules in the XCS rules’ population [P] that match the current perception will form a Match Set [M] through the Match filter. Next, [M] selects an action by the selection filter through either the exploration or the exploitation

method. In exploration, the selection filter selects an action randomly from all options. In exploitation, the selection filter selects the most promising action with the maximum worth in [*M*] (see Equation 4.6). The selected action will be executed by the agent, and rules in [*M*] who vote for the executed action will form an Action Set [*A*]. After the action execution, the agent will immediately receive a step reward r_a from the reward filter as feedback from the environment in single step problems. The worth (fitness) and other statistics of the rules in [*A*] are updated according to the step reward r_a. As iterations progress, fitness better reflects the utility of a rule to guide evolutionary search to better solutions (policies of actions).

The reward filter updates each rule in [*A*] by updating the statistics of a rule, including predicted reward r_p, prediction error ϵ, and fitness F based on its r_a. Predicted reward r_p is updated by a learning rate β (see Equation 4.1).

$$r_p = r_p + \beta * (r_p - r_a) \tag{2.5}$$

Prediction error ϵ is also updated in a similar way (see Equation 4.2).

$$\epsilon = \epsilon + \beta * (|r_p - r_a| - \epsilon) \tag{2.6}$$

Finally, fitness fit, the worth of the rule, is updated through calculations of absolute accuracy κ and relative accuracy κ' (see Equations 4.3, 4.4, 4.5).

$$\kappa = \begin{cases} 1, & if\ \epsilon \leq \epsilon_0 \\ (\epsilon/\epsilon_0)^{\nu}, & otherwise. \end{cases} \tag{2.7}$$

$$\kappa' = \kappa / (\textstyle\sum_{[A]} \kappa) \tag{2.8}$$

$$fit = fit + \beta * (fit - \kappa') \tag{2.9}$$

$$worth_{a_j} = \frac{\sum_{cl_k \in [M] | a_j} r_{p_k} * fit_k}{\sum_{cl_k \in [M]} fit_k},$$
where j,k $\in \mathbb{N}$, *cl* is a classifier, a_j is the action of *cl*,
fit is the fitness of *cl*, r_p is the predicted reward of *cl*,
$worth_{a_j}$ is applied to each action in a rule set [*M*]
to select the most promising action in the exploit mode. (2.10)

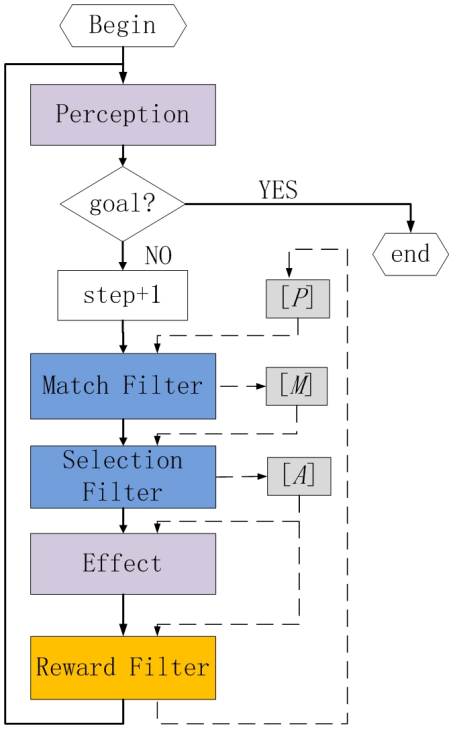


Figure 2.14: Iteration Loop of Standard XCS

Blue blocks are the standard XCS sub-processes, grey blocks are XCS rule sets, orange blocks are sub-processes of reward methods, and purple blocks indicate the agent's interactions with a maze environment. Arrows indicate the agent's working flow, and dotted arrows indicate classifiers/rules working flow.

2.5.2 Multistep Problem and Robotics Adaption

XCS algorithms have been applied to solve multistep problems [76], such as maze problems introduced by Wilson [69]. In maze problems, the agent has to execute actions in sequential iterations to complete a navigation task. Rewards are provided only in certain states in the multistep problems. A long-term reward r_l represents the completion of the task. r_l equals an arbitrary value of 1000 if the task is completed. Otherwise, r_l equals an arbitrary value of 0 in non-goal states or if the task fails (e.g. the agent has taken more steps than the step-threshold in an epoch of a trial). A short-term reward (immediate reward) r_s traditionally represents the agent's execution effects during each iteration. r_s equals an arbitrary value of -1 as a cost of the agent moving

a step forward. r_s equals an arbitrary value of -50 if a collision occurs. At the final state, the sum of all short-term rewards and the long-term reward contributes to the final reward as the measurement of the performance of the agent.

Traditionally, a reward propagation is required by a reward method for applying the standard XCS iteration loop in the multistep problems. The reward method is internally responsible for assigning external rewards, including short-term rewards and any long-term reward, to step rewards for each iteration. The step reward r_a is the reward which the agent assigns to recent active rules in $[A]_i$ for the step i, as mentioned in the single step problem above. The standard XCS reward method encapsulates long-term rewards into short-term rewards through a Q-learning like algorithm [69, 76]. The maximum potential reward available is propagated to previous actions $[A]_{i-1}$, even if that action is not taken in $[A]_i$. Although Q-learning can mathematically guarantee the approximation of estimated r_a, it takes a long time before the effect of a r_l can propagate to early states. The propagation speed of the standard reward method might be not fast enough for robotic applications in the real-world. A mobile robot might take minutes to finish a propagation by interacting with a real-world environment, but millions of iterations for training the robot is impractical.

Bull et al. [77, 78, 79] proposed an approach to weight the influence of time on the reward propagation in TCS (Temporal Classifier System). Instead of the propagation of the reward to every micro-states that a robot responds to, in TCS, the reward is propagated to previous macro-states, which subsume these micro-states through the generalization ability of the traditional LCS. Therefore, the number of iterations in the reward propagation can be significantly reduced, thus increasing the efficiency of the reward propagation, when micro-states can be subsumed based on spatial continuous features (i.e. the brightness in a square that contains a single light [77, 80, 81]) in the environment. That is, the subsumption of micro-states essentially increases the effectiveness of reward propagation. However, TCS will face a similar reward propagation problem when the subsumption fails as it is based on spatial continuous features. As spatial continuous features are not always available in a real-world environment, TCS will face a similar reward propagation problem like other Q-learning like algorithms.

Butz et al. [82, 83] applied XCSF [84], a derivative of the XCS, to learn functional approximation in robotic control problems. XCSF was applied to approximate kinematics model of a robot arm with seven degrees of freedom (DoFs) [82]. Each classifier

represents a kinematic models, the velocity kinematics of Jacobians, and XCSF evolves a population of classifiers with the goal of accurate and maximally general approximations. The result showed that XCSF can learn a locally linear forward-inverse model of the velocity kinematics of robotic arms [82]. Later, this work was extended to improve robustness against noise on sensors of the robot arm [83]. By including a Kalman filtering within classifiers, the extended approach can alleviate the negative effect of noisy sensors, improving the noise-robustness for successful learning and control. These applications of flexible and adaptive robot arm control suggest that XCS algorithms can be expected to be applied to an even broader range of robotic applications.

Inspired by cognitive psychology, Anticipatory Learning Classifier System (ACS) and the subsequent ACS2 approach were proposed to encapsulate the psychological mechanism of the anticipative behavioural control [85, 86]. Extended from classical LCSs' "condition-action" representation, ACS2 has "condition-action-expectation" triples. As the "expectation" part are embedded in the representation, ACS2 can achieve latent learning, which is defined as learning in the absence of environmental reward. In a robotic application, a mobile robot searches a maze and learns to achieve navigation to a location where rewards are not always available [87]. Furthermore, ACS2 was tested in real-valued environments, such as real-multiplexer and the cart pole problem, which provide promising results for adjusting a psychologically inspired mechanism into LCSs [86].

Williams et al. [17, 18] proposed a reward method suited to robotic applications to solve multistep problems. The reward method encapsulates short-term effects, such as collisions and number of steps taken, into the time factor (*time*) of long-term rewards (see Equation 4.7). Instead of updating [*A*] at the end of each iteration as the standard XCS reward method does, the robotics method records [*A*] in a Reward Stack [*R*] (see Figure 2.15). At the end of a task, the robotic reward method propagates the long-term reward r_l backward to all previously active XCS rules evenly (see Equation 4.8) or with a discount factor (γ) to emphasise the contributions of recently active rules (see Equation 4.9).

$$r_l = 1000 + 1/time \tag{2.11}$$

$$r_{a,i} = c * r_l / num_{step}, \tag{2.12}$$

where c is a constant, num_{step} is the number of steps, and i $\in$ step.

$$r_{a,i} = r_l * \gamma^i, \tag{2.13}$$

where γ is the learning rate.

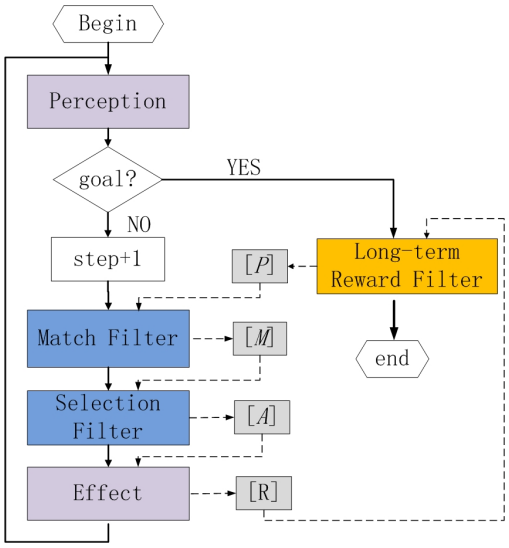


Figure 2.15: Robotic XCS Iteration Loop

Blue blocks are the standard XCS sub-processes, grey blocks are XCS rule sets, orange blocks are sub-processes of reward methods, and purple blocks indicate the agent's interactions with a maze environment. Arrows indicate the agent's working flow, and dotted arrows indicate classifiers/rules working flow.

2.6 Previous Work Of Emotion-inspired Cognitive Architecture

Previously, Williams [17, 18] proposed an emotion inspired cognitive architecture that can learn solutions for adaptive path-planning of robotic navigation tasks. This cognitive architecture evolves an emergent emotion model and generated appropriate, non-preset affective responses during navigations for the first time [19]. The cognitive architecture

has a three-layer network structure, and each layer contains its unique type of nodes as cognitive-affect units (see Figure 2.16 and Figure 2.17). They are reinforcers, emotions [5] and modifiers. Reinforcers generate stimuli from environmental perceptions, modifiers affect executions as responses to the environment, and emotions connect reinforcers with modifiers as summaries of perceptions and selections of affected executions. That is, the learned affective solutions encapsulate the reinforcers-emotion-modifiers mappings by the interpretable three-layer architecture (see Figure 2.16).

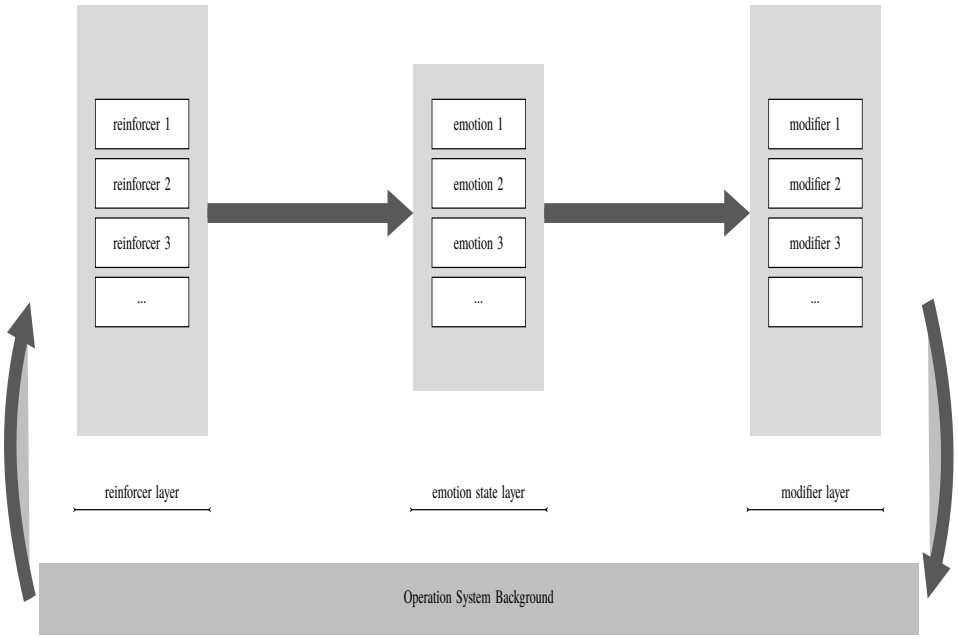


Figure 2.16: Three-layer architecture of the previous work

This three-layer cognitive architecture can automatically learn these affective solutions by an RL process. The cognitive architecture applies an XCS agent (Section 2.5) to learn the reinforcers-emotion-modifiers mappings by RL training. The agent learns both the context of the nodes and the connections between them. As a result, the three-layer architecture learns reinforcers-emotion-modifiers mappings as interpretable emotional responses in various scenarios. That is, emotions embedded in the reinforcers-emotion-modifiers mappings can be interpreted as a "fear" state and a "happy" state by these embedded mappings and their effects. For example, a "fear" state emerged as the robot navigates in a crowded scenario, and the robot learned to engage responses

[5] emotion states

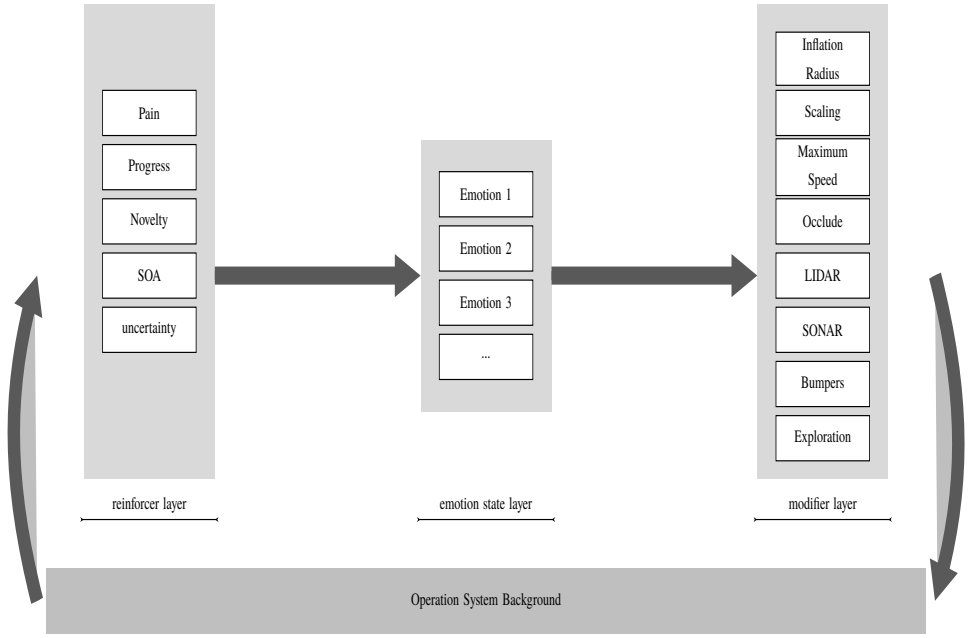


Figure 2.17: Implementation of the three-layer architecture of the previous work

that caused fewer collisions. In contrast, a "happy" state emerged as the robot navigates in an empty scenario, and the robot learned to move quickly. Without presetting reinforcers-emotion-modifiers mappings as affective solutions, the robot can learn to perform affective behaviours for the path-planning task.

However, the reinforcers-emotion-modifiers mappings of learned affective solutions lack diversity. The learned emotion states are embedded components of the solutions, yet lack symbolic diversity that is characterized by their natural counterparts. The psychological meanings of emotion states come from the external interpretations of their respective mappings that emerge probabilistically. Because these mappings are established by perceived stimuli, including environmental and proprioceptive ones, the emotion states would become homogeneous as stimuli lack enough (symbolic) diversity. In Williams' work, an increasing number of emotion states fail to increase diversity in terms of their symbolic meanings. The emotion states were therefore limited to two different symbolic meanings in the three-layer architecture: "happy" and "fear".

The limited diversity of emotion states also constrain the robot's flexibility. Because modifiers are embedded with an emotion state in a reinforcers-emotion-modifiers mapping, the diversity of the modifiers is also constrained by the limited diversity of

emotion states. As modifiers are associated with a robot's execution behaviours, the robot's behaviour flexibility is thus constrained in the three-layer cognitive architecture. In the previous work, the robots' behaviours are limited by two sets of modifiers in the affective solutions. As a result, the existing affective solutions fail to generate varied behaviours for task completion.

2.7 Pioneer Robotic Platform

Pioneer 3-DX provides a reasonably standard robotic platform that the proposed cognitive system can be deployed to. The Pioneer [88, 89] is a two-wheel two-motor differential drive robot equipped with sensors and an operating system. The sensors, including LIDAR, sonars, and bumpers, allow this mobile robot to detect the environment, and an external wireless router facilitates its communication with remote machines. ROS (Robot Operating System), an open-source operating system, is utilised on the hardware, providing a flexible framework for developing robot software [90]. ROS provides system services for the Pioneer, including hardware APIs, standard functionality, network and message passing system, etc. As the ROS network allows a ROS system to be deployed across multiple machines, additional remote machines can mount distributed computing resource for the entire network, thus increasing the limited computing budget of the robot. In this work, two desktop boxes[6] join the ROS network, which empowers the Pioneer and the proposed cognitive system.

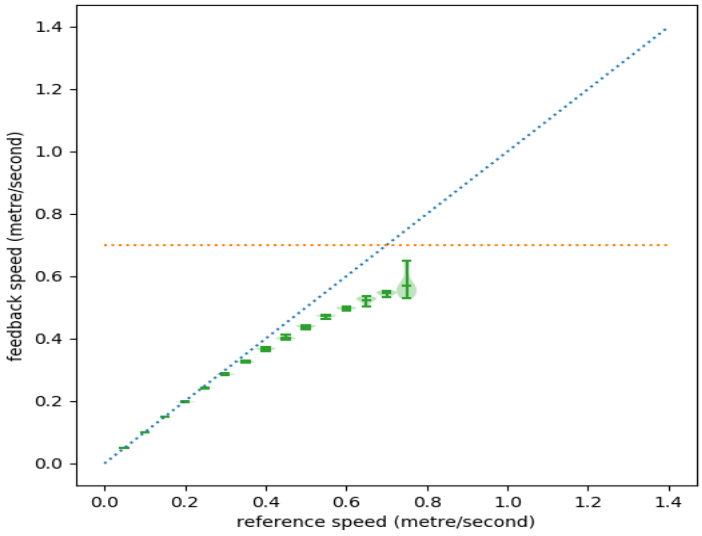
The Pioneer applies ROS environment for the robotic control system. In ROS, the software is organized in packages, each of which contains ROS nodes, a ROS-independent library, a dataset, configuration files, a third-party piece of software, or anything else that logically constitutes a useful module [91]. Packages that can be flexibly installed and then initiated for their functionality cooperate toward the robot's task completion. In the Pioneer, various packages are applied for perception, recognition, localization, planning, motor control, etc. From this point of view, the proposed cognitive system is equivalent to a package in this work. The proposed cognitive system contains 16 ROS nodes (see Chapter 3) that cooperate with other nodes and packages (i.e. navigation packages) for the Pioneer to complete tasks.

[6]The Dell desktop box has eight CPUs, Intel Core i7-4900, CPU 3.60GHz, 8G RAM.

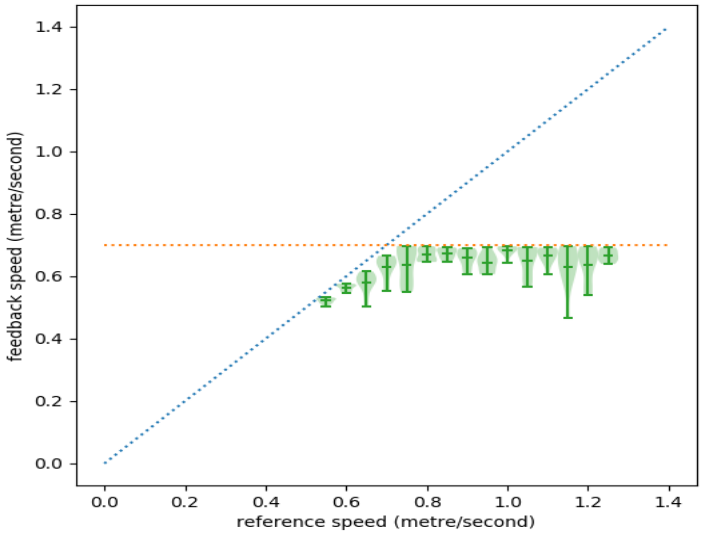
Although the ROS nodes can be principally clustered by packages, there is no boundary for messages that can be shared between nodes within the ROS network. To complete processes, every ROS node can subscribe to any message from the network or publish messages to the network. Two approaches for passing a message are both applied in this work: the publisher-and-subscriber approach and the service-and-client approach. Generally, the publisher-and-subscriber approach is applied to pass messages between processes to generate a flow of message, and the service-and-client approach is applied to communications within the same process for the synchronization of the message.

ROS provides a flexible software-developing environment that facilitates the design of an artificial cognitive system. Firstly, all ROS nodes are accessible to an artificial cognitive system, which is equivalent to a ROS package. Through ROS nodes, an artificial cognitive system can directly or indirectly engage with all the cognitive processes, including perception, recognition, localization, planning, motor control, etc. This will be similar to the brain's activities at a conscious level. Secondly, connections between nodes can be modified in a cognitive system. In a human brain, the cognitive system becomes mature as connections between different parts of the brain develop. Therefore, evolving the mappings of the nodes creates the potential for creating a mature cognitive system. Thirdly, homogeneous ROS nodes can be created, added, replaced, or removed from an artificial cognitive system. This flexibility is also shown in a human's brain. Different modules of a human brain have homogeneous components (see Figure 2.5) that, in some special cases, a module can be transferred from an original, damaged area to a neighbouring area.

Except for the software environment, the Pioneer's mobility is a critical factor that is related to its navigation performance. Although any reference velocity can be published by a ROS node through a velocity command, the execution of the command is limited by the robot's mechanism in the real-world. An experiment was conducted to verify the assumed linear relationship between the reference velocity and the execution velocity. In the experiment, the Pioneer was commanded to move straight forward with a constant reference velocity commanded by a ROS node. The experiment was focused on the forward velocity rather than the rotational velocity because (1) the robot's mobility is more sensitive to the forward velocity, and (2) the rotational velocity is also related to the forward velocity because of the two-motor differential driving mechanism of the robot.



(a) The 5 metre scenario



(b) The 10 metre scenario

Figure 2.18: The verification of the forward velocity of the Pioneer

The robot travelled for distances of 5 metres and 10 metres in two separated scenarios. The feedback velocity is equal to the travelled distance divided by the time-consumption.

The results indicate the mobility limitation of the Pioneer. The maximum forward velocity that the robot achieved in the scenarios is about 0.7 metres/seconds (see Figure 2.18). In addition, the relationship between the reference velocity and the feedback velocity become nonlinear when the reference velocity approaches the maximum velocity. Based on the result, the maximum value of the forward speed was set to 0.5 m/s for the real-world environment, compared to 4 m/s in the previous work [18]. Therefore, the previous setting is inappropriate because 4 m/s is a setting far beyond the speed range that the robot can achieve.

ROS also supports a simulation environment for the Pioneer. Gazebo is an open-source simulation tool that can offer various simulated robots and environments. To conduct experiments on the stand-alone Gazebo, a set of ROS packages are applied to provide necessary interfaces between the ROS and Gazebo. Gazebo offers the ability to rapidly test the Pioneer's performance and underlying algorithms before they are trained in real-world scenarios. Gazebo can also provide various simulated environments to test the scalability of the robotic application without safety concerns.

2.8 Chapter Summary

This chapter introduced the background of this work. The chapter began with a brief review of four diverse cognitive systems. SOAR, MDB, and ACT-R are examples that demonstrate the symbolic, emergent, and hybrid approaches of construction of a cognitive system respectively. Although these three cognitive architectures do not explicitly consider any emotion theory as their assumptions, research shows that emotion mechanisms play a critical part in cognitive processes, such as decision making. Thus, CAPS was introduced as a hybrid cognitive system, which takes emotion theories explicitly as its assumptions. As CAPS achieves its hybrid cognitive system through distributed computing nodes, the proposed hybrid cognitive system will be constructed by distributed computing nodes.

The success of CAPS led to a review of emotion theories and the brain's cognitive architecture in the next section. From this section, three emotion theories (e.g. Constructive Theory, Appraisal Theory, and Basic Emotion Theory) provided inspirations

for the design of the cognitive processes in the proposed architecture.

To apply a cognitive system for robot applications, this chapter also reviewed robotic behaviour-based subsumption systems. The behaviour-based subsumption operations in these systems has great potential as a basis to construct a completely autonomous robotic system. Therefore, based on the behaviour-based subsumption operations, this work will propose contingency-based subsumption operations to construct the proposed system.

Combining the inspiration of emotion theories and the composition processes of the subsumption systems, this work proposes a contingency-based subsumption system, the Affective Computing Multilayer Cognitive Architecture (see Chapter 3). Because an autonomous robot requires autonomous knowledge generation, machine learning techniques are discussed for this purpose. Accuracy-based classifier system (XCS) is the selected machine learning techniques because of its symbolic interpretation, evolutionary capability and expansibility for robotic applications. This part gave a brief introduction to two variants of the XCS algorithm underlying this work (see Chapter 4). The previous work, which this work is extended from, was introduced as the starting point of this work. At the end of the chapter, a robot platform for the proposed architecture was introduced. The mobile robot, the Pioneer, was introduced with its software operating system and hardware capabilities.

Chapter 3

Methodology

3.1 Introduction

This thesis aims to develop an evolutionary cognitive architecture (system) for a mobile robot that can learn adaptive solutions to complete tasks. The reviewed literature indicates that a desired cognitive system should contain the following features:

Firstly, the proposed robotic cognitive architecture can be composed of various computing nodes. Inspired by cognitive systems (Section 2.2), the proposed architecture is a network of computing nodes, which cooperate to provide solutions to a complex robotic task. Thus, each computing node in this work can realise a separated functional module (e.g. obstacle-recognition, decision-making, motor control) that is required to complete the task.

Secondly, these computing nodes can cooperate and interact with each other to achieve cognitive processes, which provide solutions for robotic tasks. Inspired by the emotion theories and the brain's cognitive architecture (Section 2.3), computing nodes can be classified into five categories by their roles in cognitive processes. The five categories are *primary reinforcer*, *secondary reinforcer*, *core affect state*, *policy*, and *behaviour*. These computing nodes constitute layers by their categories, thus the proposed cognitive system has a five-layer architecture.

Thirdly, cooperation and interaction of the computing nodes can be achieved by subsumption operations. Inspired by the behaviour-based subsumption system (Section 2.4), a high-level computing node can subsume low-level ones according to the contin-

gency [1] between these low-level nodes and their outcomes. As the robot interacts with the environment, various contingencies [2] can be achieved through contingency-based subsumptions, automatically establishing a cooperating network of the cognitive system. These contingencies can be encapsulated in rules [3], providing a clear interpretation of the cooperation and interaction of the computing nodes. Thus, the proposed contingency-based subsumption is applied in evolutionary processes of the automatic establishment of the cognitive architecture.

Fourthly, contingency-based subsumption operations can be achieved through machine learning techniques. Inspired by the Learning Classifier Systems (LCSs) (Section 2.5), contingency-based subsumption operations deploy Accuracy-based Learning Classifier System (XCS) as their underlying machine learning technique. Thus, variants of this evolutionary computing algorithm are proposed for the subsumption operations, allowing SAOC rules to evolve as the robot interacts with the environment.

This work proposes a five-layer cognitive architecture, Affective Computing Multilayer Cognitive Architecture (ACMCA), according to these four features. ACMCA attempts to learn an affective solution to robotic tasks.

The hypothesis is that affective solutions are combined efforts of: (1) diverse stimuli of two types of reinforcers in different perceptional levels, (2) symbolic emotion states in an affective decision-making level, and (3) flexible behaviours in different executional levels.

According to emotion theories, a solution should be composed of five symbolic components to achieve these combined efforts (Figure 3.1):(1) *primary reinforcers* that recognise the environmental features, (2) *secondary reinforcers* that generate rules of Stimulus-Action-Outcome Contingency (SAOC), (3) *core affect states* that are elicited by perceptual stimuli and affect decisions, (4) *policies* that provide flexible schemes for various scenarios of the task, and (5) *behaviours* that that execute action tendencies and robotic operations in each state of a task scenario.

This novel affective solution is extended from the previous solution (see Figure 2.16, Figure 3.1 and Figure 3.9 for the comparison.). That is, the new solution (1) extends the

[1] A contingency is an estimation of a relation between Stimuli, Action, and their Outcome. In this work, this contingency is termed Stimulus-Action-Outcome Contingency (SAOC).

[2] SAOCs, Stimulus-Action-Outcome Contingencies

[3] SAOC rules

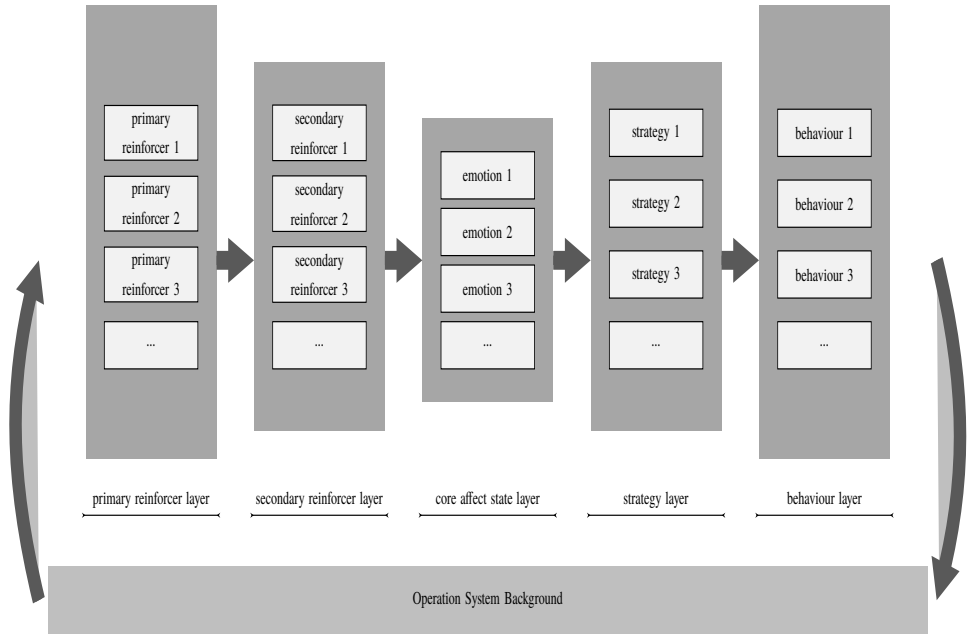


Figure 3.1: Affective solution of the Affective Computing Multilayer Cognitive Architecture (ACMCA)

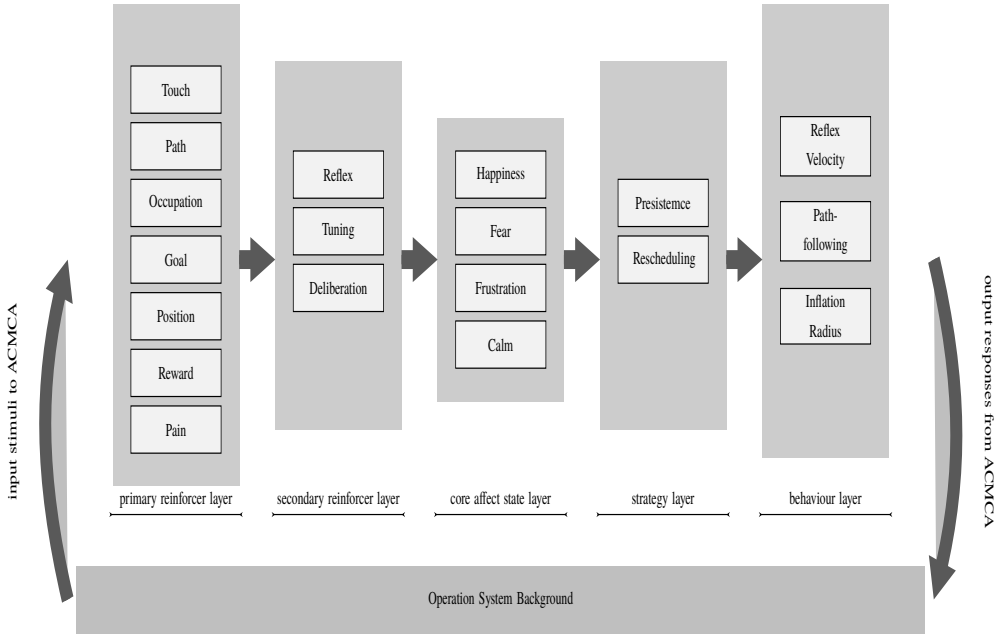


Figure 3.2: Implementation of the five-layer architecture of ACMCA for the Pioneer

existing reinforcers into *primary reinforcers* and *secondary reinforcers* for the perception levels of the affective solution, (2) extends the existing modifiers into *policies* and *behaviours* for the execution levels, and (3) converts the existing emotion states into *core affect states*. These extensions and developments are inspired by the three emotion theories, which are discussed in this chapter.

This chapter will start with an introduction to affective solutions that ACMCA attempts to achieve. The five categories of components of affect solutions (see Figure 3.1) are introduced along with their inspirations from emotion theories, which attempt to explain the affective and the cognitive process. The methodology of ACMCA will follow the introduction. 16 computing nodes are constructed for the application of ACMCA for the Pioneer (see Figure 3.9). The methodologies of these computing nodes are described in the order of the five layers. As each node encapsulates a functional module, ACMCA can provide solutions for a robot through the cooperation of these computing modules. This chapter will conclude with a brief summary, leading to the underlying machine learning technique proposed in the next chapter.

3.2 Symbolic Affective Solution

The symbolic affective solution is inspired by the three emotion theories: the Constructive Theory, the Appraisal Theory, and the Basic (Emotion) Theory. The five components of the affective solution are illustrated with their potential symbolic meanings and inspirations in this section. These five components are *primary reinforcer*, *secondary reinforcer*, *core affect state*, *strategy* and *behaviour* (Figure 3.1). Each component focuses on a different perspective of the affective solution: recognition, contingency-establishing, affective decision-making, scheduling, or behaviour. The five components of the symbolic affective solution are presented below.

3.2.1 Primary Reinforcer

Primary reinforcers (primary stimuli) aim to recognise scenario features and provide diverse raw perception for the robot. As inputs of the architecture, these primary reinforcers directly describe a scenario by recognising the surrounding environment and receiving the task objectives (see Figure 3.3). Primary reinforcers can cover scenario

features, such as obstacles in the environment, the goal position of a task, the time-consumption for the task, etc. By directly connecting to sensors, primary reinforcers recognise the environment and receive task requirements, providing perception for the architecture.

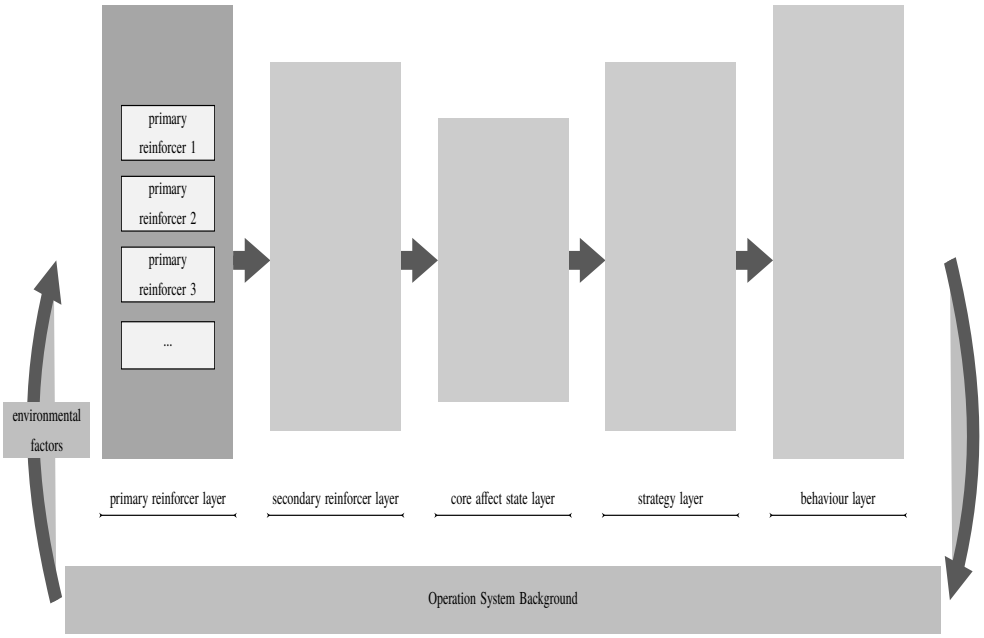


Figure 3.3: Primary reinforcers of the affective solution

The complexity of the primary reinforcers depends on the difficulty of their recognitions. Some primary reinforcers simply convert sensor readings. For example, a primary reinforcer, which represents the sense of touch, can directly store the bumpers' readings for further usages. Other primary reinforcers require data from pre-processing models and other primary reinforcers for the recognition. An example is that a primary stimulus, which recognises a robot's current position in an environment, requires mapping and localization models. As the recognition becomes more complex, a primary stimulus might rely on sophisticated models. For example, a primary reinforcer to recognise a dynamic obstacle might rely on a figure-recognising model that is trained by machine learning algorithms.

The definition of the primary reinforcer in this work is modified from the original concept of the primary reinforcer in Constructive Theory. By the definition of Constructive Theory, primary reinforcers are unlearned stimuli, which do not have any

learning ability; the learning ability is an exclusive character of secondary reinforcers [53]. However, this work slightly modifies this definition of the category. The functional capability (i.e. learning ability) is no longer applied to categorise a reinforcer. Instead, the construction capability (i.e. subsumption ability) is to separate primary reinforcers and secondary reinforcers. That is, a primary reinforcer is a raw stimulus and an unconstructive one, and a secondary reinforcer is a stimulus that is constructed by other stimuli, including primary reinforcers and other secondary reinforcers. By this modification, an obstacle-recognising reinforcer (i.e. the occupation node in Section 3.3.1.5), which learns patterns to recognise obstacles, is still a primary reinforcer, although it is capable of learning a recognition pattern. This primary reinforcer can be applied to construct a secondary reinforcer (i.e. the deliberation node in Section 3.3.2.3) through a subsumption process. Therefore, the modification allows a primary reinforcer to become a completed functional module, being consistent with the feature that is learned from the review of cognitive systems.

3.2.2 Secondary Reinforcer

A secondary reinforcer is a constructive stimulus that aims to establish a Stimulus-Action-Outcome Contingency (*SAOC*) inspired by Constructive Theory. According to Constructive Theory, secondary reinforcers [4] are stimuli "which, if their occurrence, termination, or omission is made contingent upon the making of a response, alter the probability of the future emission of the response [53]." That is, a secondary reinforcer is a stimulus that is constructed with a contingency. In the field of psychology, this contingency is termed as an "Action-Outcome (AO) Contingency", which represents a relationship between Actions and their consequential Outcomes. In this work, by encapsulating an input stimulus, the AO Contingency is extended as a "Stimulus-Action-Outcome Contingency (SAOC)" to construct secondary reinforcers. This SAOC is represented by SAOC rules in this work.

The structure of SAOC rules can provide a symbolic, abstract interpretation of a scenario. Generally, a SAOC rule contains Stimuli, Action, and Outcome. In a rule of SAOC, an input Stimulus is a condition, which can trigger its following Action with an expected consequential Outcome. Thus, a SAOC rule can be represented by an "if-

[4] secondary reinforcer is also termed as instrumental stimulus or secondary stimulus.

then" or "condition-action" representation, providing a clear symbolic interpretation. Its symbolic meaning can be interpreted as: if the input stimulus is perceived, the output action will be advocated as a cognitive response, expecting the outcome of the action with an estimation of relevance.

SAOC rules can transfer an input stimuli (such as primary reinforcers) to secondary reinforcers. Once SAOC rules are established, an input stimulus can be transferred from its recognition to a secondary reinforcer by associating with its outcome and estimation. For example, “money” is a secondary reinforcer, which is associated with the value stamped by a government authority rather than by the recognition of a banknote (primary reinforcer). When you are evoked by a primary stimulus of a banknote and you are also aware of its value of “ten dollars”, you take an action of spending the banknote for ten-dollar goods, receiving goods that are worth ten dollars as an outcome. Secondary reinforcers and SAOC rules can be constructed through an agent’s experience. By reserving an agent’s past experience, accurate SAOC rules can play a critical role in the agent’s decision-making process when the agent faces a similar scenario.

Secondary reinforcers can cover diverse SAOC rules as they encapsulate various components. The exhaustive number of diverse SAOC rules depends on combinations of three components of SAOC rules: the input stimulus, the output action and the outcome. In a SAOC rule, the input stimulus and the outcome can come from any primary reinforcers, and the output action can come from any strategy (see Section 3.2.4) and any behaviour (see Section 3.2.5). As each combination represents a different type of SAOC, the architecture can establish diverse SAOC rules, the facilitating decision-making process even affecting the emotion-eliciting process.

Each secondary reinforcer can deploy a machine learning agent (i.e. an XCS agent) to evolve accurate SAOC rules. Each SAOC rule can be encapsulated by an XCS classifier and a set of SAOC rules constitutes a population of classifiers in an XCS agent (see section 2.5). In a classifier, the condition part covers the input stimulus and output action of a SAOC rule, the action part predicts the outcome of this rule, and the statistics are updated based on the comparison of the outcome and its predicted value. Therefore, each time when an input stimulus occurs, an XCS agent will experience an iteration and apply its population of classifiers to evolve SAOC rules. As iterations go, accurate SAOC rules will automatically emerge within the XCS agent. Eventually, these SAOC rules can alter the probability of the emission of the output action toward the desired

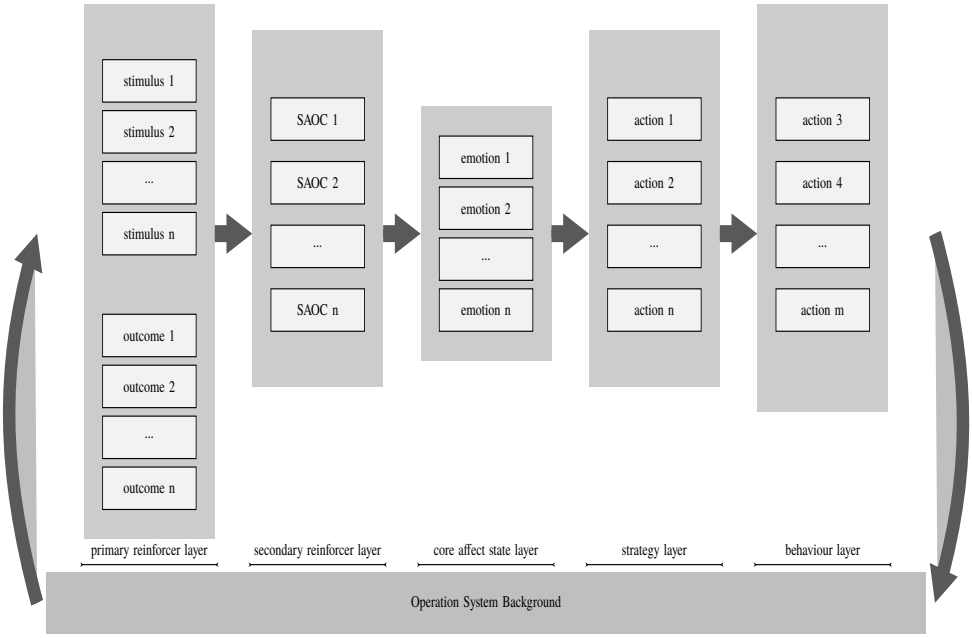


Figure 3.4: SAOC (Stimulus-Action-Outcome-Contingency) components in the Five-layer architecture of ACMCA

outcome. In this work, various types of SAOC rules are established then evolved by the secondary reinforcers (Section 3.3.2).

3.2.3 Core Affect State

The core affect state is an integral component of the emerged affective solution, summarising perception and execution (Figure 3.5). According to Appraisal Theory, a core affect state is "a neurophysiological state that is consciously accessible as a simple, nonreflective feeling [92]." In this work, core affect states are anthropomorphic emotion states that integrate, bridge and connect stimuli in perception and responses in execution. Therefore, at a given moment, perceptual stimuli elicit a core affect state, causing consequential affective responses.

The mappings from stimuli to core affect states are inspired by the Appraisal Theory. This theory assumes that core affect states integrate stimuli by measuring their hedonic (displeasure-pleasure) and their arousal (sleepy-activated) values. Hence, core affect states can be demonstrated and labelled within a two-dimensional space of the hedonic

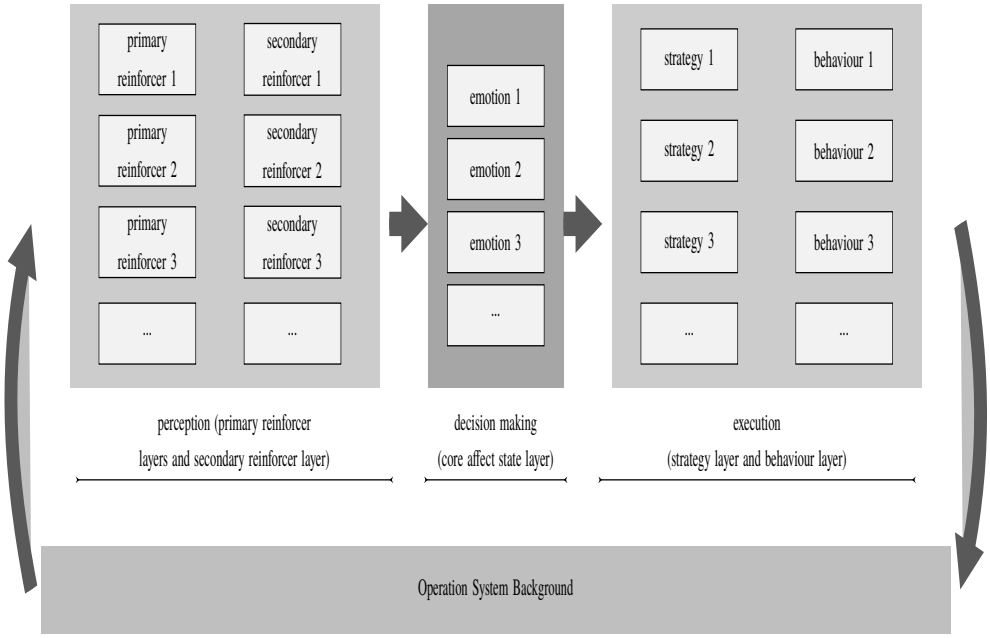


Figure 3.5: Core affect states of the affective solution

and the arousal measurement: core affect space (see Figure 3.6). For example, an elicited core affect state of "happy" in the first quadrant of the space has both higher hedonic and arousal values than "calm" in the fourth quadrant. Compared to a discrete state, whose symbolic meaning only comes from interpretations of external mappings in the previous work (see section 2.6), the symbolic meaning of the core affect states also include the state's internal hedonic and arousal values, making the artificial labels more explainable. Therefore, core affect states are elicited by stimuli according to the Appraisal Theory, and these core affect states are demonstrated with symbolic labels (i.e. happy, calm) for clear explanations in this work.

The mappings from core affect states to affective responses are motivated by Basic Emotion Theory. This theory treats emotions and feelings as "minimalist predictions" of affective responses [93]. For example, Panksepp suggests the seven basic affective responses according to each basic emotion, such as the Seeking for enthusiastic and Fear for anxious [20] [36]. Thus, a core affect state will activate its specific set of affective responses, leading to flexible behaviours in execution. Following this concept, the artificial core affect states will activate sequential responses and behaviours for robotic executions.

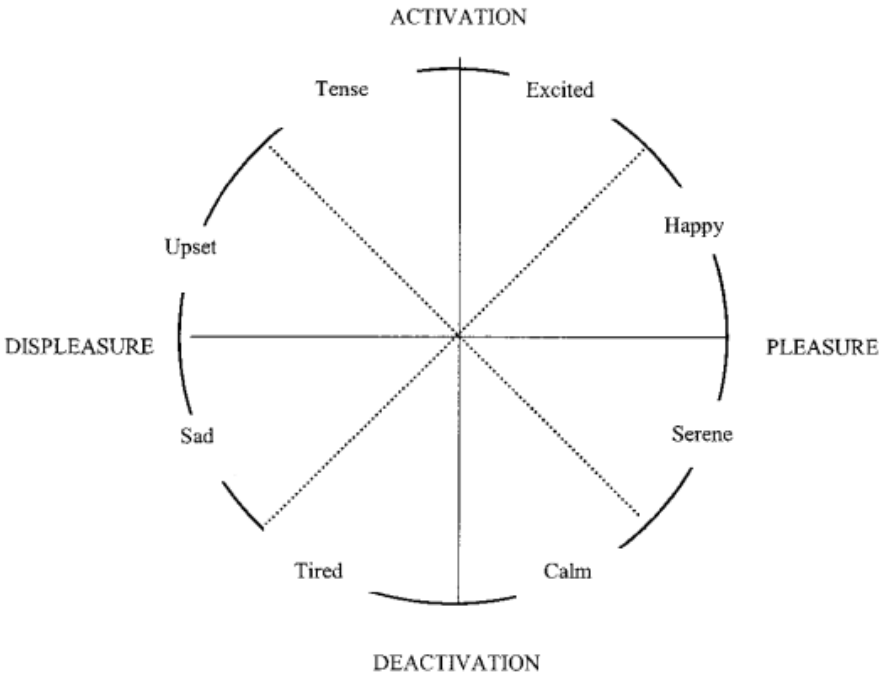


Figure 3.6: Core affect space [92]

The responses and behaviours engage various components of cognitive architecture. Neurological studies show that brain firings of affective responses are distributed in specific regions, driving affective tendencies and behaviours. For example, the Seeking system, which is associated with anticipatory-appetitive behaviours, is driven by several neurological structures, including the ventral tegmental area and lateral hypothalamus [43] [94]. Inspired by these studies, affective responses will activate various components of the architecture, including strategies (see Section 3.2.4) and behaviours (see Section 3.2.5) in this work.

3.2.4 Strategy

Strategy is a high-level executional component of the affective solution, representing affective responses over a task scenario. Distributed among the strategy layer of the architecture (see Figure 3.7), a strategy is an overall plan subsuming a scenario's goal and executional behaviours. When an emotion-provoking event is perceived in a scenario, diverse Strategies allow a versatile robot to respond to the event flexibly, including

switching the goal and the behaviours. Once a core-affect-state-associated strategy is trained and evolved, the strategy itself represents a characteristic responding pattern to that affective event, as the basic emotion theory suggested.

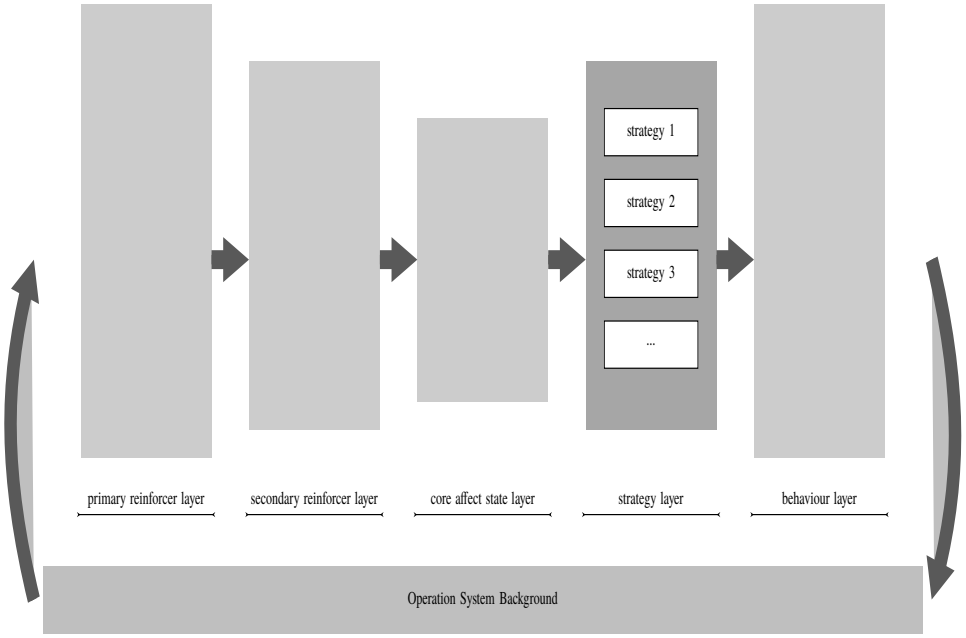


Figure 3.7: Strategies of the affective solution

3.2.5 Behaviour

Behaviour is a low-level executional component of the affective solution, representing affective responses over a step/state of the robotic task scenario. Distributed among the behaviour layer of the architecture (see Figure 3.8), a behaviour is an optional element that can directly or indirectly influence a robot's performance. For example, as a direct output of the architecture, a behaviour can be a specific velocity command that directly affects the robot's movement. A behaviour can also be a hyperparameter of a navigation model, through which the behaviour will indirectly influence a robot's navigation performance. Traditionally, these optional elements can be preset by engineers, thus introducing potential human bias. Alternatively, optional-element-encapsulated behaviours can evolve by evolutionary algorithms through the robot's training scenarios. Being subsumed into various SAOC rules (i.e. IR pattern (see Section 3.3.2.2)) for the

algorithm's training process, behaviours will automatically adapt to the given training scenarios.

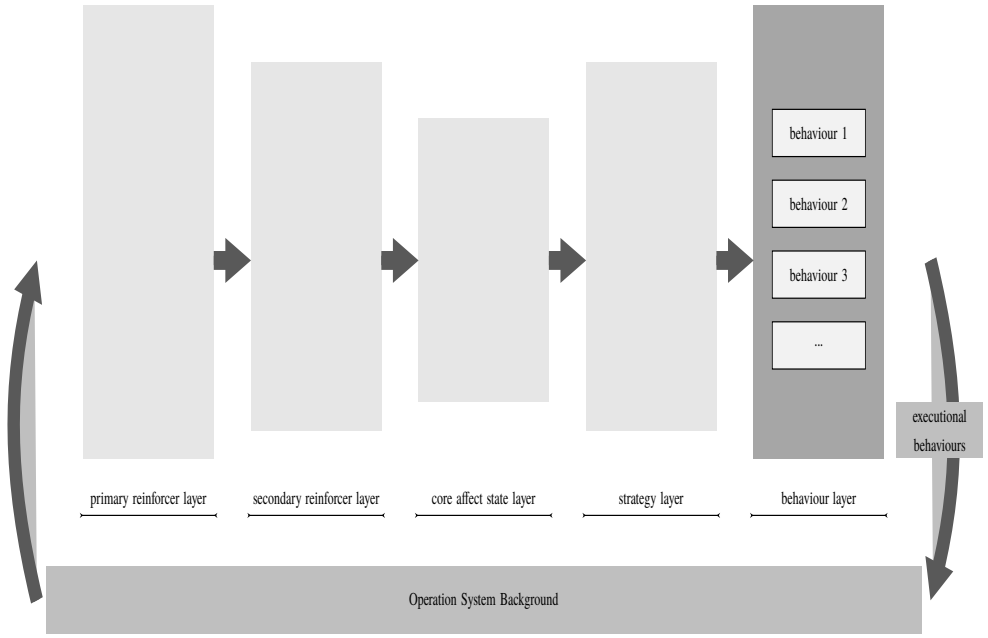


Figure 3.8: Behaviours of the affective solution

3.3 Five-layer Architecture

Affective Computing Multilayer Cognitive Architecture (ACMCA) is proposed to learn the symbolic affective solution for a robotic task. ACMCA has a five-layer architecture, corresponding to the five components of the solution (see Section 3.2). These five layers are the *primary reinforcer layer* and the *secondary reinforcer layer* as perception layers, the *core affect layer*, the *strategy layer*, and the *behaviour layer* for affective decision making and execution layers (see Figure **??**). Each layer contains a specific type of computing node, and the five types of computing node are termed: *primary reinforcer*, *secondary reinforcer*, *core affect state*, *strategy*, and *behaviour*. These nodes are distributed to form the operational system of a robotic application, communicating messages through the client-server approach for the entire cognitive architecture.

ACMCA is designed to be a universal cognitive system for robots. The scalability of ACMCA layers for heterogeneous robots increases from the outer layers toward the inner

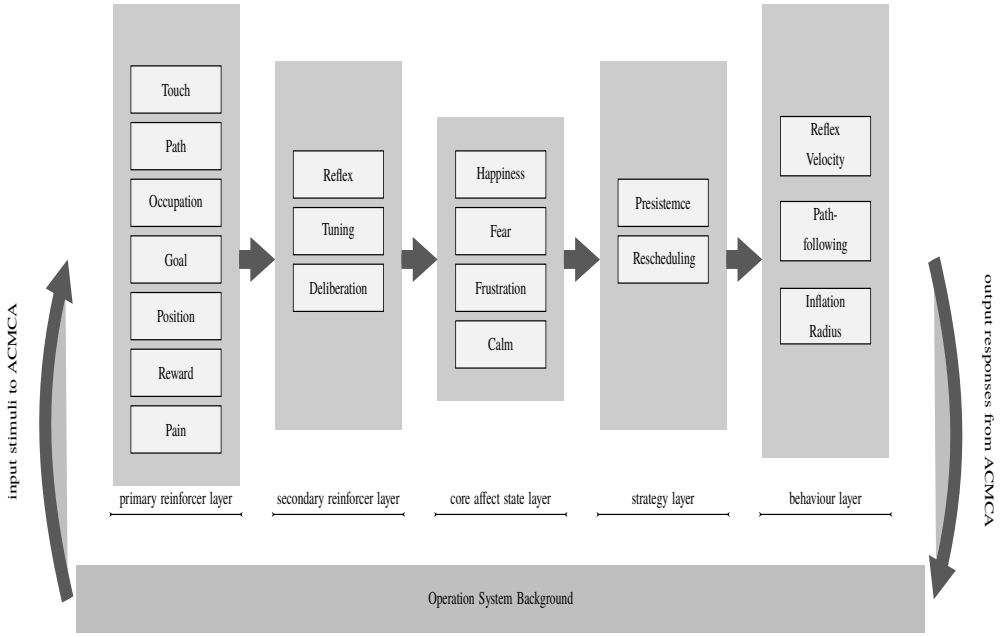


Figure 3.9: Implementation of the five-layer architecture of ACMCA for the Pioneer

layer. That is, the secondary reinforcer is more scalable than the primary reinforcer, the behaviour pattern layer is more scalable than the modifier layer, and the core affect layer is the most scalable of all of them. For example, if this work is applied for another mobile robot, primary reinforcers and modifiers may require adjustments for different sensors and heterogeneous mechanisms. Yet, the inner layers, *secondary reinforcer layer*, *core affect layer* and *strategy*, may require little modification.

3.3.1 Primary Reinforcer (Layer One)

Primary reinforcers aim to recognise environmental features in the primary reinforcer layer of ACMCA. Generally, any computing unit/node of the robot operating system can be considered as a primary reinforcer to perceive diverse raw stimuli. As this work is conducted on a mobile robot to complete its navigation tasks, these environmental features can come from any aspect of the task, including the experimental environment of the task, the requirement of the task, and proprioceptive perception during the execution of the task. Each primary reinforcer recognises an environmental feature, generating an internal estimation as a raw stimulus. In this work, the selection of primary reinforcers

from computing nodes includes *Touch*, *Path*, *Occupation*, *Goal*, *Position*, *Reward* (see Figure 3.10). Different primary reinforcers can be added into or removed from the primary reinforcer layer according to the task and/or robot morphology. These primary reinforcers will be subsumed by separated secondary reinforcers (see Section 3.3.2).

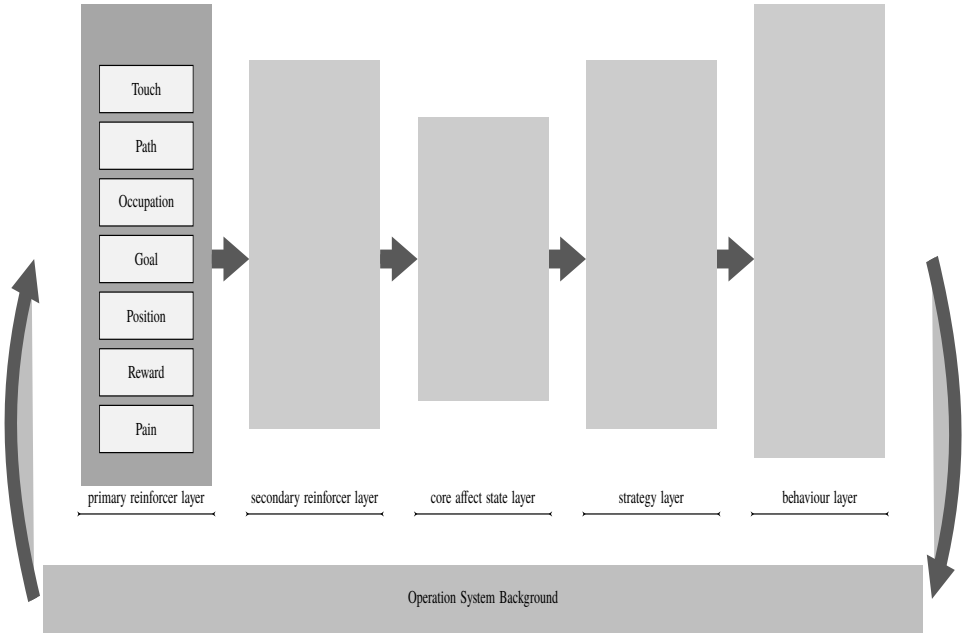


Figure 3.10: Implementation of primary reinforcers of ACMCA

This work manually selects these primary reinforcer nodes based on the symbolic meanings of the Stimulus-Action-Outcome Contingency (SAOC) rules of secondary reinforcers. An automatic selection of primary reinforcer would require sophisticated techniques (i.e, feature construction, reward assignment method, meta-learning method). Although a developing XCS algorithm (XCSCF, see Section 2.5) shows its potential to achieve automatic selection in solving binary problems, extending the XCSCF algorithm for robotic application is beyond the scope of this thesis.

3.3.1.1 Touch

The *Touch* node is a primary reinforcer, which could be chosen to represent the robot's touching sensation of the environment. Biologically, touching perception provides essential sensations for infants during their brains' structural and functional development

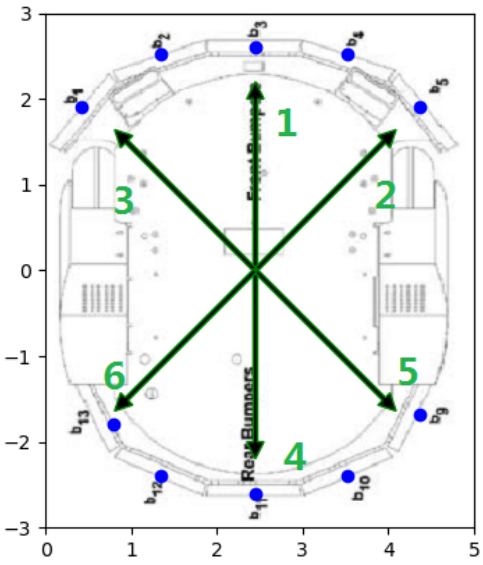


Figure 3.11: Top view of the Pioneer
Bumpers are marked with blues points. The green arrows represent the moving directions of the Reflex Velocities (Section 3.3.5.1)

[95, 96, 97]. Similarly, the robotic touching sensation could be applied for the composition of the robotic control system in this work. In this work, this sensation will lead to a quickly responding pattern, the reflex pattern (see Section 3.3.2.1 and 5.2), which can be applied to respond to dangerous environmental features.

The Touch node directly records the bumpers' readings. The node contains a 10-bit string, indicating the ten bumpers' detections of collision (see the ten bumpers marked as blue points in Figure 3.11). The value of each bit of the string is 1 for a detected collision and 0 for no collision.

The Touch generates an inner stimulus, termed *Pain*, when collisions are detected. Biologically, the peripheral stimulus of Pain can act as an impulse and provoke an automatic response, a reflex, even before it reaches the conscious level of the brain. In this work, a stimulus of Pain generated by the Touch node can be considered as another primary stimulus to provoke an emotional response (see Section 3.3.3). As a detected impulse, the value of the Pain is calculated as a scale value of -1000 for collisions and 0

for no collision.

It is possible to replace the Touch node with primary reinforcers of different sensations. For example, the sensation of vision from a camera can also be applied for a similar purpose. Yet, this work chose the touching sensation for two reasons. The vision sensation requires a more complex process than the touching, causing a response delay to an environmental hazard. The second reason is to increase the diversity of stimuli. As vision is widely applied to avoid predictable collisions, the touching sensation can be applied for unpredicted collisions. It will be interesting to compare different sensations' effects regarding a similar purpose in future.

3.3.1.2 Goal

The primary reinforcer, *Goal*, represents the destination of a navigation task. The Goal node directly records the destination position of the task issued for goal-oriented responses. This Goal node can be different parts of SAOC rules in various cases. For example, the Goal node can trigger a robot's sequential response at the beginning of navigation. In this case, this primary reinforcer is the input stimulus of the SAOC rules that aim to establish a navigation responding contingence (see Section 5.3). In another case, the Goal node can provide feedback at the end of navigation to score the navigation performance and carry out reward assignments. Thus, the Goal node and the current Position (see Section 3.3.1.3) have access to the Outcome part of the SAOC rules to estimate the effects of these SAOC rules. The Goal node can also be transferred into other primary reinforcers. The primary reinforcer of Occupation (see Section 3.3.1.5) encapsulates the Goal node to recognise the feasibility of the destination position.

3.3.1.3 Position

The primary reinforcer, *Position*, represents the robot's awareness of its current position. The Position updates the robot's current position by the simultaneous localization and mapping models of the robotic system. This proprioception can be applied for an internal estimate of the progress during a task and reward assignments at the end of a task.

3.3.1.4 Path

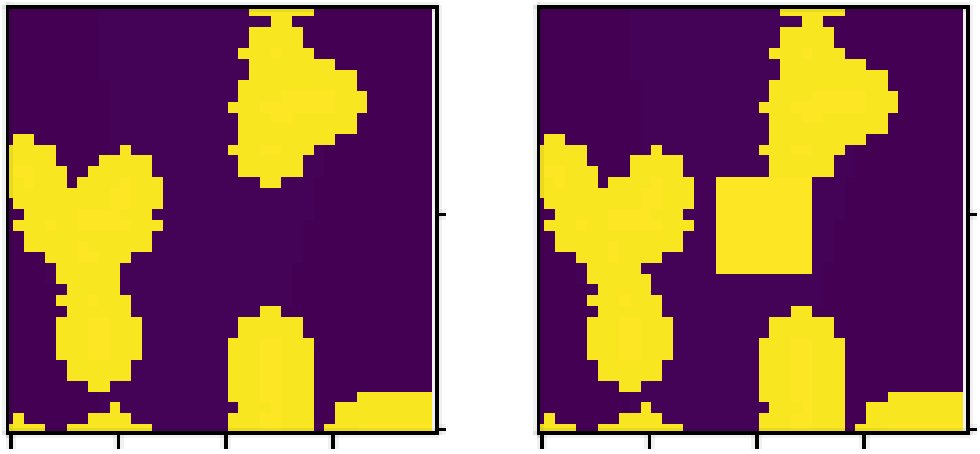
The primary reinforcer, *Path*, represents the task feasibility in terms of trajectory generation. When a Goal (see Section 3.3.1.2) is issued, the path-generation model of a robot will calculate trajectories between its current position and the destination position. The path node records the calculated result of the path-generation model. If a trajectory exists, the Path will be marked with a scale value of 1. Otherwise, the Path is 0 because the path-generation model cannot generate a valid trajectory.

The Path node can be applied for compositions of a primary reinforcer and a secondary reinforcer. The primary reinforcer of Occupation (Section 3.3.1.5) applies this node to recognise the appearance of a dynamic obstacle. The Occupation node applies a Convolutional Neural Network (CNN) (see Table 3.12 detailed in Section 3.3.1.5) to predict the appearance of the obstacle, where the Path node can provide the ground truth for the training of the CNN. The secondary reinforcer of the Tuning node (see Section 3.3.2.2) subsumes the Path node into SAOC rules. The Path node is the Outcome part of the SAOC rules in the Tuning node, representing the effect of the SAOC rules and the hyperparameter currently applied on the path-planning model.

3.3.1.5 Occupation

The primary reinforcer, *Occupation*, represents the task feasibility in terms of obstacle occupation. If a dynamic obstacle might be occupying the goal position, the task feasibility depends on the occupied state of the obstacle. For example, a dynamic obstacle can block the path, which a robot follows to its goal position. Occupation predicts the occupied state by a costmap figure of the goal position (see Figure 3.12). Based on the Path, Occupation is calculated as a predicted probability of the occupied state by a Convolutional Neural Network (CNN) model, which is a "killer" model for figure recognition [98]. A trained CNN model will allow the Occupation node quickly respond to a dynamic obstacle. Compared to the Path node, the Occupation is much quicker than the Path in the calculation, because the latter suffers the delay caused by simultaneousness of the distributed path-generation service. Therefore, the Occupation can replace the Path in real-world scenarios where the delay matters.

The Occupation deploys a trained CNN model for the predictions of obstacle occupation. The structure of the CNN model is shown in Table 3.1. A costmap represent



(a) Costmap figure of an empty path (type one)

(b) Costmap figure including an obstacle in the path (type two)

Figure 3.12: Costmap figures

These two costmap figures detect a real-world environment shown in Figure 5.3.

the environment detected by the path-planning model through sensors, and a costmap can be presented by a figure (see Figure 3.12). The CNN model takes a local costmap, which describes the 3 metres by 3 metres area around the goal position, as input for a prediction. The size of the figures is limited to 126 pixels by 126 pixels so that the CNN model can be trained within the available computing resource [5].

A dataset of costmap figures was established to train the CNN model. Costmap figures capture a targeted area where the robot navigates. These costmap figures are automatically labelled by the primary reinforcer Path to train the CNN model. According to the obstacle's occupation (see Figure 3.12), the label has two categories. If the targeted area is occupied by an obstacle, the label value is 1. Otherwise, the label value is 0. After training for 50 epochs, the model achieves 98.14% accuracy over the dataset. During the testing phase for 300 applications, the model achieves 100% accuracy.

3.3.1.6 Reward

The primary reinforcer, *Reward*, represents the score of a robot's performance in the latest task. This primary reinforcer provides feedback on the latest performance based

[5] a Dell desk box with eight CPUs. Intel Core i7-4900, CPU 3.60GHz, 8G RAM.

Table 3.1: CNN Model

Layer (type)	Output Shape	Parameters
activation-1 (Activation)	(None 126 126 32)	0
conv2d-2 (Conv2D)	(None 124 124 32)	9248 ((3*3*32+1)*32)
activation-2 (Activation)	(None 124 124 32)	0
max pooling2d-1 (MaxPooling2d)	(None 62 62 32)	0
dropout-1 (Dropout)	(None 62 62 32)	0
conv2d-3 (Conv2D)	(None 62 62 64)	18496 ((3*3*32+1)*64)
activation-3 (Activation)	(None 62 62 64)	0
conv2d-4 (Conv2D)	(None 60 60 64)	36928
activation-4 (Activation)	(None 60 60 64)	0
max pooling2d-2 (MaxPooling2d)	(None 30 30 64)	0
dropout-2 (Dropout)	(None 30 30 64)	0
flatten-1 (Flatten)	(None 57600)	0
dense-1 (Dense)	(None 512)	29491712 ((57600+1)*512)
activation-5 (Activation)	(None 512)	0
dropout-3 (Dropout)	(None 512)	0
dense-2 (Dense)	(None 3)	1539
activation-6 (Activation)	(None 3)	0

Total parameters: 29,559,107
Trainable parameters: 29,559,107
Non-trainable parameters: 0

on the time consumption for the task achievement. When the Position matches the Goal within a time consumption threshold, the last performance is considered to have accomplished the task. Then, Reward is calculated by the time consumption, termed *time*, based on a hyperbolic approach (see Equation 3.1). The constants in the equation are to simulate the hyperbolic relationship between the strength of a human's feeling and the duration of that feeling *time* [99]. That is, the strength of feeling that is elicited by the stimulus will decay as the stimulus lasts.

The *time* will also create pressure that drives the robot to perform efficiently for a task. The constants in the equation are designed for a specific scenario, and an

adjustment of these constants can create a desirable curve of time pressure for the targeted scenario. Nevertheless, the hyperbolic approach is schedulable for every scenario. In real applications, this hyperbolic approach is functional if the variance of *time* in the measurement does not override its measurement accuracy. In this case, no matter how small difference, a solution that brings a larger reward will become a superior one to the solution with a smaller reward in the long term.

$$Reward = 100000/(time + 50) - 1000 \tag{3.1}$$

where $time$ is the time consumption. The constant 50 represents a threshold of 50 seconds for a robotic task. When the time consumption is less than this threshold, the reward will be positive. Otherwise, there will be a punishment (negative reward) for the last performance. The constant 100000 and the constant 1000 are to set the maximum value of the hyperbola curve to a value of 1000.

3.3.1.7 Delay

The primary reinforcer, *Delay*, records the time that a robot has spent on a task. Psychologically, satisfaction of current performance decays over time under influences of emotions (i.e. frustration) (see Figure 3.13) [99]. The Delay node provides a real-time stimulus from an environment, allowing a robot to estimate its on-processing performance of a task at the given moment. When there is a dynamic environmental feature (i.e. a dynamic obstacle) that causes the obstruction of the task process, this real-time stimulus will allow ACMCA to construct a secondary reinforcer (i.e. the Deliberation Node, see Section 3.3.2.3), which can develop patterns to respond to the dynamic feature toward the task completion.

3.3.1.8 Velocity

The primary reinforcer, *Velocity*, records a robot's velocity at the given moment. Psychologically, a human's movement is a critical factor that can indicate his/her arousal level (see the Appraisal theory in Section 2.3) and/or action tendency (see the Basic Emotion Theory in Section 2.3) The Velocity node provides a stimulus that estimates a robot's activation of an on-processing performance. In the real-world scenario, an emergent

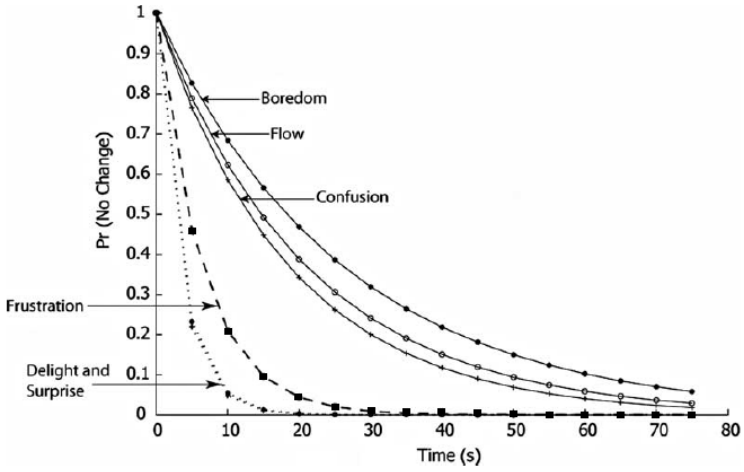


Figure 3.13: Exponential decay curves for the main effect of affect (i.e. Frustration) [99].

event occurs and disrupts the robot's navigation task with a dynamical change in the velocity. Therefore, this primary reinforcer provides the robot with a rapid perception of the emergent event, allowing the robot to respond to the event before a resume of the original task.

3.3.2 Secondary Reinforcer (Layer Two)

Secondary reinforcers are designed to establish Stimulus-Action-Outcome Contingency (SAOC) rules (Section 3.2.2) that can be applied for a robot's decision-making processes. Compared to primary reinforcers, Strategies, and Behaviours, secondary reinforcers are high-level nodes that can subsume these low-level nodes into SAOC rules. In a contingency-based subsumption operation, a SAOC rule subsumes a stimulus part as a condition, an action part as a response to the condition, and an outcome part as the consequential effect. That is, symbols of low-level nodes are encapsulated in the condition part, action part, or outcome part of SAOC rules. Through this construction of the SAOC rule, an accurate SAOC rule indicates a strong contingency between its encapsulated parts, thus it can be applied for decision-making processes. Each secondary reinforcer deploys an XCS agent (Chapter 4) to conduct this contingency-based subsumption and to evolve SAOC rules. As the XCS agent evolves when the robot interacts with the environment, high-accuracy SAOC rules can emerge in a secondary

reinforcer.

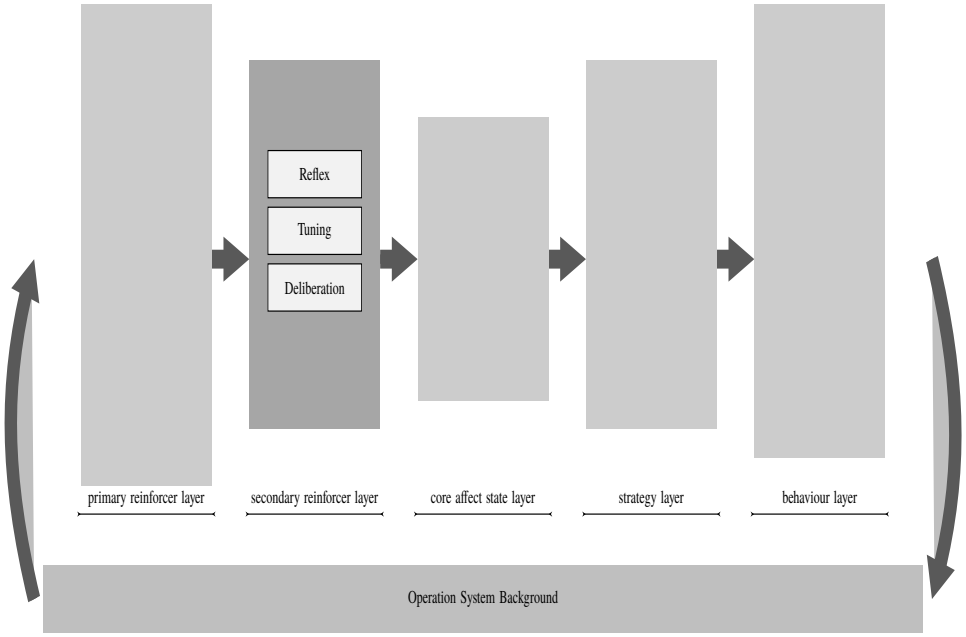


Figure 3.14: Implementation of secondary reinforcers of ACMCA

Three secondary reinforcers are proposed to investigate their capability to compose various SAOC rules. By encapsulating various Strategies (Section 3.3.4) or Behaviours (Section 3.3.5) in the action part of SAOC rules, SAOC rules can affect a robot's responses on the model level, the strategy level, and the behavioural level. To generate robotic responses on these levels, three secondary reinforcers are proposed: the *Reflex* node, the *Tuning* node, and the *Deliberation* node (see Figure 3.14). These secondary reinforcers will learn various appropriate responses to environment features.

3.3.2.1 Reflex Node

Reflex node is a secondary reinforcer that aims to construct a set of SAOC rules for a robot's response on a concrete behaviour level, and is illustrated in Figure 3.15. The set of SAOC rules that are learned in the Reflex node will allow the robot to rapidly respond to unpredicted obstacles without continuous collisions. As shown in Figure 3.15, the Reflex node applies an XCS agent (see the blue line that link between the Reflex node and the XCS agent) to learn this set of SAOC rules (Section 3.2.2 for a brief introduction

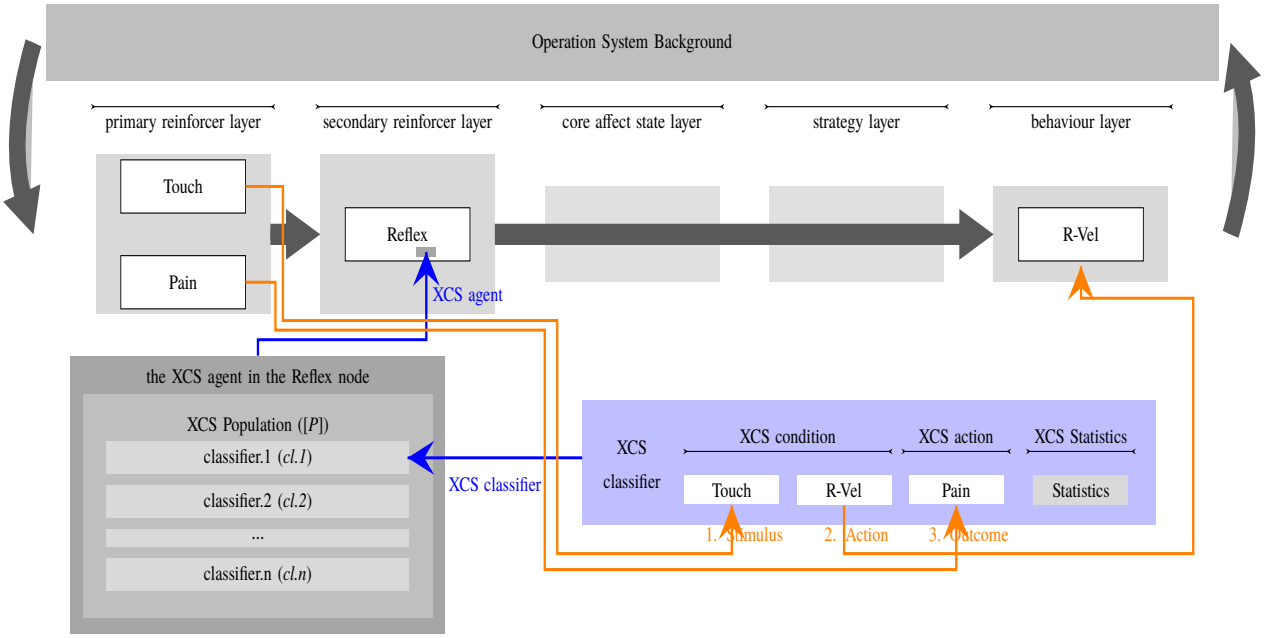


Figure 3.15: Reflex node of ACMCA

The blue block details the components of a classifier that encapsulates a SAOC rule. The blue lines refer to the affiliation between the node, the XCS agent, and a classifier. The red lines refer to the flow of the data to/from a classifier.

to applying an XCS agent to learn a set of SAOC rules). As proposed in Section 3.2.2, a classifier from the population of the XCS agent in the Reflex node encapsulates various components to represent a SAOC rule. In this XCS agent, the classifier (see the blue block in Figure 3.15) encapsulates *Touch* (Section 3.3.1.1), the *Reflex-Velocity* (also called *R-Vel* for short, see Section 3.3.5.1), the *Pain* (Section 3.3.1.1). That is, in a classifier, (1) The Touch perceives collisions, executes its sequential response, and transfers them into the stimulus input of a SAOC rule. It is covered in the condition part of the XCS classifier as an encapsulated perception (see the red dashed line 1 in Figure 3.15). (2) The Reflex-action is the output action of a SAOC rule and an executional behaviour of the robot. It also is covered in the condition part of the XCS classifier with the Touch as an encapsulated execution (see the red dashed line 2 in Figure 3.15). (3) The Pain is the outcome part of a SAOC rule and feedback of the executional behaviour from the environment. It is predicted by the action part of the XCS classifier as an encapsulated anticipation (see the red dashed line 3 in Figure 3.15). (4) The innate statistics update

when this XCS agent iterates.

This set of SAOC rules evolves within the classifiers by the XCS agent's training iteration. In the training, the XCS agent updates its classifiers and SAOC rules by the accuracy in the prediction of the Pain at each iteration. After the training, the accurate SAOC rules of the Reflex node, termed as *Reflex-Pattern*, will emerge in the XCS agent after sufficient iterations. The Reflex-Pattern is the first affective pattern that is learned by ACMCA in this work, and it can guide a mobile robot to move away from unpredicted collisions without presetting appropriate responses. The pattern will demonstrate that ACMCA can generate appropriate responses from the behavioural level. Experiments in learning the Reflex-Pattern are conducted and detailed in Section 5.2.

3.3.2.2 Tuning Node

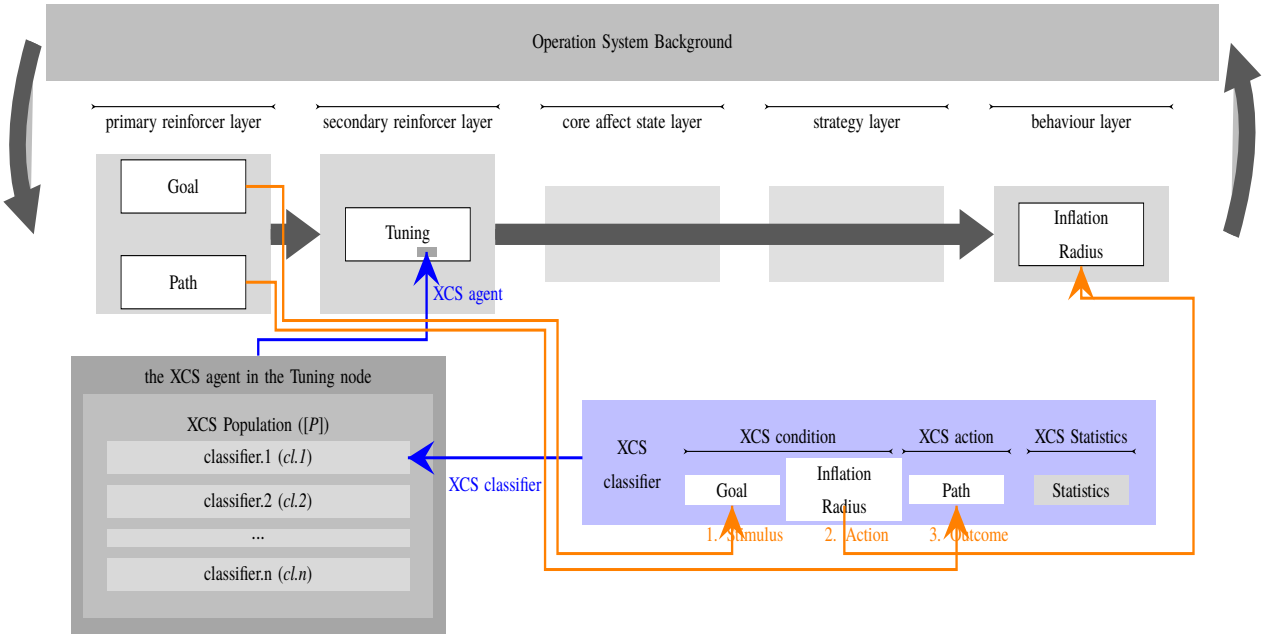


Figure 3.16: Tuning node of ACMCA

The blue block details the components of a classifier that encapsulates a SAOC rule. The blue lines refer to the affiliation between the node, the XCS agent, and a classifier. The red lines refer to the flows of the data to/from a classifier.

Tuning node is another secondary reinforcer that aims to construct a set of SAOC rules, which allow the path-planning model of a mobile robot to be adapted to its

scenario. The SAOC rules will provide appropriate hyperparameters of the model, so the model can be adaptive to different scenarios. Similar to the Reflex node, the Tuning node also deploys a separate XCS agent to learn this set of SAOC rules (see Figure 3.16). In the XCS agent, the classifiers include components of *Goal*, the *Inflation Radius* (*IR*), and the *Path* to encapsulate the SAOC rules to be learned (see Figure 3.16 for the components of the classifier, and see Section 3.3.1.2, 3.3.5.3, 3.3.1.4 for details of these components.) That is, (1) The XCS condition covers the goal position in the Goal that the path-planning model generates a path toward. The Goal component indicates the scenario that the SAOC rules and classifiers are applied to (see the red dashed line 1 in Figure 3.16). (2) The XCS condition part also covers the IR, the affecting hyperparameter of the model, to describe the action part of the SAOC rules. Therefore, different values of IR are associated with various Goals within the XCS condition (see the red dashed line 2 in Figure 3.16). (3) The XCS action part of a classifier is to predict whether a valid path is generated by the model. The Path of the primary reinforcer is compared to the prediction of the XCS action part for updating of the statistics of this classifier (see the red dashed line 3 in Figure 3.16). (4) Finally, the updated statistics are kept in the statistics part for the future process of the evolving algorithm.

This set of SAOC rules evolves within the classifiers by the XCS agent's training iteration. In the training, the XCS agent updates its classifiers and SAOC rules by the accuracy in the prediction of the Path at each iteration. After the training, the accurate SAOC rules of the Tuning node, termed as *IR-Pattern*, will emerge in the XCS agent after its sufficient iterations. The IR-Pattern is the second affective pattern that is learned by ACMCA in this work, and it can support an adaptive path-planning approach for a mobile robot moving without presetting hyperparameters. The pattern will demonstrate that ACMCA can generate appropriate responses from the model level. Experiments in learning the IR-Pattern are conducted and detailed in Section 5.3.

3.3.2.3 Deliberation Node

Deliberation node is the third secondary reinforcer in the secondary reinforcer layer. It aims to construct SAOC rules that will allow a mobile robot to reflect on its own policies and to reason its choices under "frustration". Similar to the two secondary reinforcers mentioned above, a separate XCS agent is deployed on the Deliberation node to learn

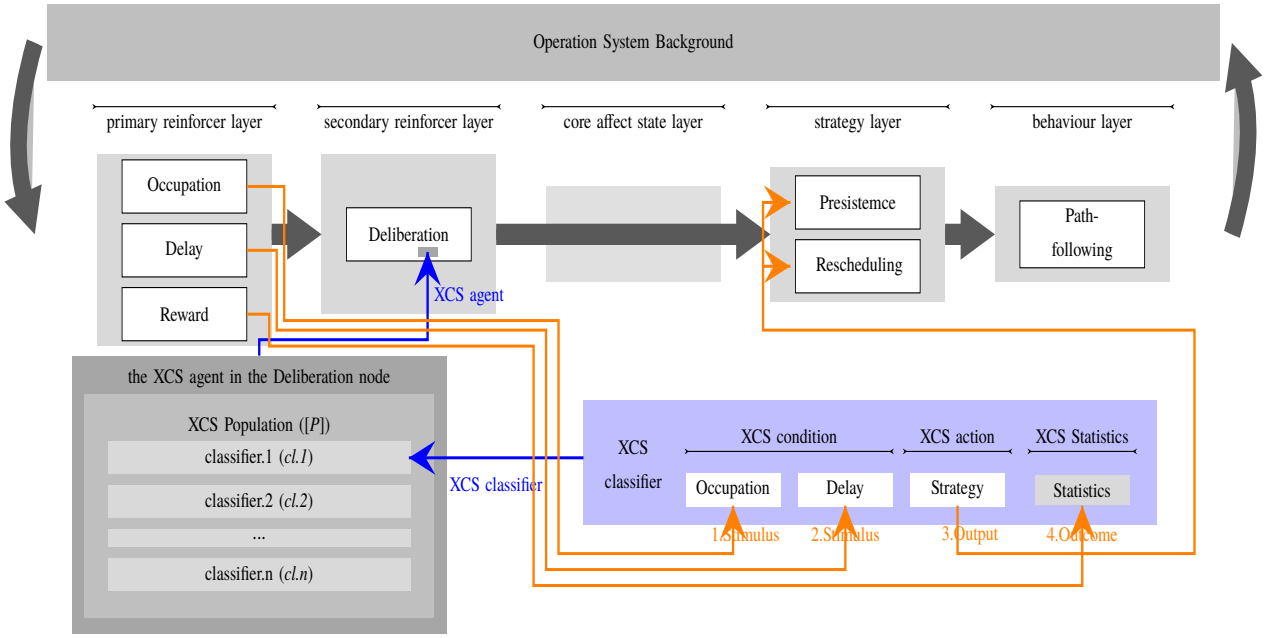


Figure 3.17: Deliberation node of ACMCA

The blue block details the components of a classifier that encapsulates a SAOC rule. The blue lines refer to the affiliation between the node, the XCS agent, and a classifier. The red lines refer to the flows of the data to/from a classifier.

the SAOC rules that are related to frustration.

The SAOC rules related to frustration encapsulate the *Occupation* node, the *Delay* node, the *Strategy* node, and the *Reward* node to represent a frustration pattern (see details of the Occupation node, the Delay node, the Strategy node, and the Reward in Section 3.3.1.5, 3.3.1.7, 3.3.4, and 3.3.1.6). (1) The SAOC rules have two input stimuli: the Occupation and the Delay. The first stimulus is the Occupation. The Occupation, which perceives a dynamic obstacle in the goal position, is associated with an event that can elicit an emotion of "frustration". The second stimulus is the Delay. The Delay, which records the time-delay caused by the obstacle, indicates the intensity of the frustration. Therefore, the intensity increases as the Delay lasts, providing a second stimulus to respond to. (2) The action part of the SAOC rules encapsulates the Strategy as the response to these two stimuli. Different choices in the Strategy produce various outcomes in the scenario. (3) The outcome part records the outcomes as the feedback of the execution of the Strategy. This part encapsulates the Reward as it measures the

robotic performance and the execution of the Strategy.

The SAOC rules are contained in the classifiers of the XCS agent of the Deliberation node (see Figure 3.17). Specifically, the condition part of classifiers contain the two stimuli of the targeted SAOC rules, and the action part of classifiers contain the action part of the SAOC rules. Classifiers whose condition part covers the current stimuli will be activated to advocate the Strategy node for execution. The statistics of these activated classifiers update according to the performance of the execution and the outcome of the SAOC rules. Therefore, the quality of the SAOC rules, such as their accuracies, can be ascertained from the statistics.

The SAOC rules in the Deliberation node evolve within the XCS agent as it iterates. Useful SAOC rules can automatically emerge as the classifiers of the XCS agent become accurate after multiple iterations. These accurate SAOC rules are termed as *frustration-pattern* because it is a pattern response to a "frustrating" environmental feature in the experiment (Section 5.4). The frustration-pattern is the third affective pattern that is learned by ACMCA in this work, showing ACMCA's ability to learn flexible responses from the Strategy level.

3.3.3 Core Affect State (Layer Three)

The core-affect-state layer aims to establish an emotion model that makes decisions from the highest hierarchy of ACMCA. Psychologically, conscious emotions are assumed to be processed at the highest-level mental process in the nested brain-mind hierarchies [36]. In this work, an emotion model, which serves as a decision-making mechanism at the highest level of ACMCA, can emerge at the core-affect-state layer. The emotion model attempts to summarise diverse stimuli, elicit core affect states, and activates affective responses of the robot.

The emotion model can elicit emotional states by summarising stimuli. The emotional states are termed as core affect states in this work, following the definition of Appraisal theory (Section 3.2.3). "Core affect refers to consciously accessible elemental processes of pleasure and activation, has many causes, and is always present [100]". Thus, core affect states are represented by hedonic and arousal values, which span a two-dimensional space, termed core affect space in this work (Figure 3.6). That is, diverse stimuli, including primary reinforcers and secondary reinforcers, are transferred

into hedonic and arousal values to elicit a core affect state at a given monent. Therefore, there are almost infinite, non-preset emotion states that can emerge in the robot's cognitive architecture.

The first-dimensional value to describe a core affect state is its hedonic value. The hedonic value measures the pleasure level of the robot, regarding reward that provides the "common current [101]" for diverse stimuli. Different methods of calculating this value can be introduced to create heterogeneous core affect states. One potential method is to calculate the hedonic value by different time-schedules. For example, it can introduce an anticipated reward, which a robot can predict, into the hedonic calculation to create "motivation-like" emotional states. Another potential method can compare the difference between a received reward and its expected reward. This difference can create emotion states that are labelled with "surprise". These methods suggest how diverse core affect states can be, thus they are worth being extended by future work. The calculation method of the hedonic value is described in Equation 5.2 (Section 5.5), serving as a "starting position" and a benchmark before diverse hedonic-value calculation method can be applied.

The second-dimensional value to specify a core affect state is its arousal value. The arousal value measures the activation level of the robot, referring to a sense of mobilization or energy [100]. This value summaries the robot's physiological state, suggesting the physiological coping ability or the action preparation of the robot. This work transfers the sense of mobilization of the robot into its arousal value through Equation 5.3 (Section 5.5). Energy consumption is not a significant factor when each iteration only lasts minutes in this work. Nevertheless, if necessary, there is no practical problem to take energy consumption into consideration in future work.

The emotion model can map emerged core affect states to affective responses of the robot. According to the Basic Emotion Theory, emotional states are treated as "minimalist predictions" of affective responses [93]. In the emotion model, emerged core affect states are the conditions that can trigger consequential responses. Instead of presetting mappings between core affect states and affective responses, the emotion model applies contingence-based subsumption operations to achieve these mappings. That is, the mappings of the emotion model are encapsulated in SAOC rules that are assigned to this model. Similarly to SAOC rules of the secondary reinforcers, the hedonic and the arousal values of core affect states are encapsulated in the Stimuli part,

selected affective responses are encapsulated in the Action part, and their effects are encapsulated in the Outcome part (see Section 5.5). These subsumption operations also deploy a machine learning agent to evolve accurate SAOC rules. As a result, the emotion model can select appropriate responses to stimuli-eliciting events.

Although this work does not pre-define emotion states and their responses, three basic emotions that dominate different areas of this core affect space are expected as follows: *Happiness*, *Fear*, *Frustration*, and *Calm* (see Figure 3.6 and 3.18).

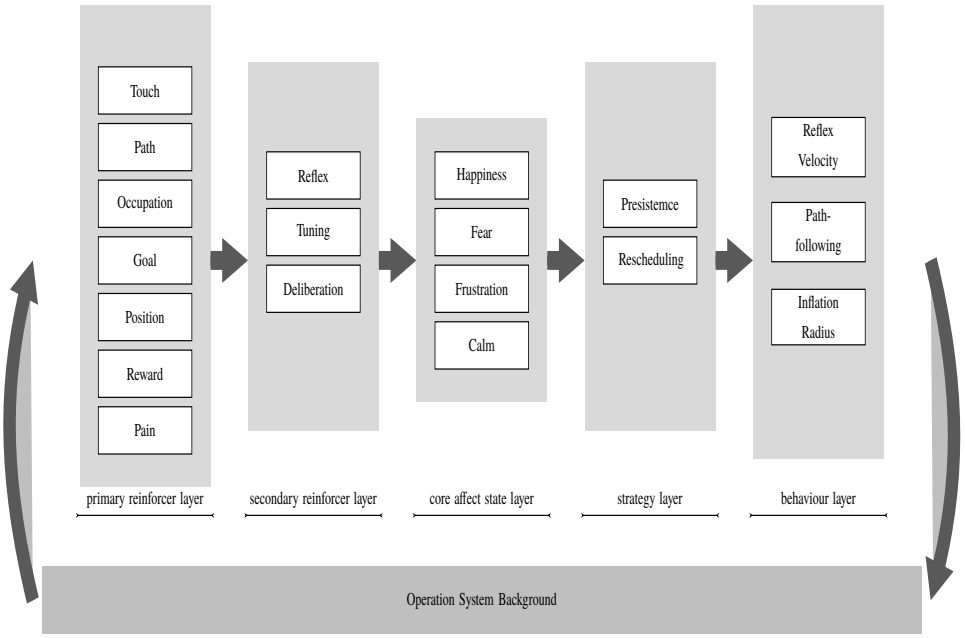


Figure 3.18: Implementation of core affect states of ACMCA

3.3.3.1 Happiness

Happiness can be used to label core affect states with high-scale hedonic value and high-scale arousal value. Happiness is elicited by reward-related stimuli, such as the target achievement, the expectation of a reward, or the absence of punishment. In this work, happiness can be an emotion state which activates goal-oriented behaviours.

3.3.3.2 Fear

Fear can be used to label core affect states with low-scale hedonic value and low-scale arousal value. Fear is triggered by stimuli of punishments, such as collisions. The responses of the fear aim to mitigate these hazard stimuli, leading to an avoid pattern.

3.3.3.3 Frustration

Frustration is a label which refers to core affect states with medium-scale hedonic value and medium-scale arousal value. Frustration occurs when anticipated reward is reduced, delayed, or removed completely [102]. From the perspectives of stimuli, frustration is described as the "withdrawal of an anticipated reinforcer" [103]. In this work, frustration is considered as a core affect state, which is elicited by the withdrawal of an anticipated reinforcer. The strength of Frustration is related to the length of the delaying time in receiving an anticipated reward (i.e. a reward of achieving a task). The frustration of the delay of anticipated reward could lead to affective responses that have negative effects on the original plan, such as rescheduling schemes.

3.3.3.4 Calm

Calm can refer to core affect states with medium-scale hedonic value and low-scale arousal value. Calm is considered as a reversal of anger. In a robot, it can lead to suppressing behaviours when they are not of benefit for the current scenario.

3.3.4 Strategy (Layer Four)

Strategy aims to provide flexible schemes and plans to deal with the environment's uncertainty factors over a task scenario[6]. In a real world application, a task scenario may contain multiple optional schemes, whose completions are impacted by an uncertainty factor. A strategy represents an optional scheme chosen as a response to an uncertainty factor for the current scenario (iteration). Encapsulated in the SAOC rules of Deliberation (Section 3.3.2.3), two strategies are proposed as affective schemes responding to the Occupation (Section 3.3.1.5). These two strategies are *Persistence* and *Rescheduling*,

[6]A task scenario is an iteration to complete the task. In a multistep scenario, a task scenario includes multiple and sequential state-action steps.

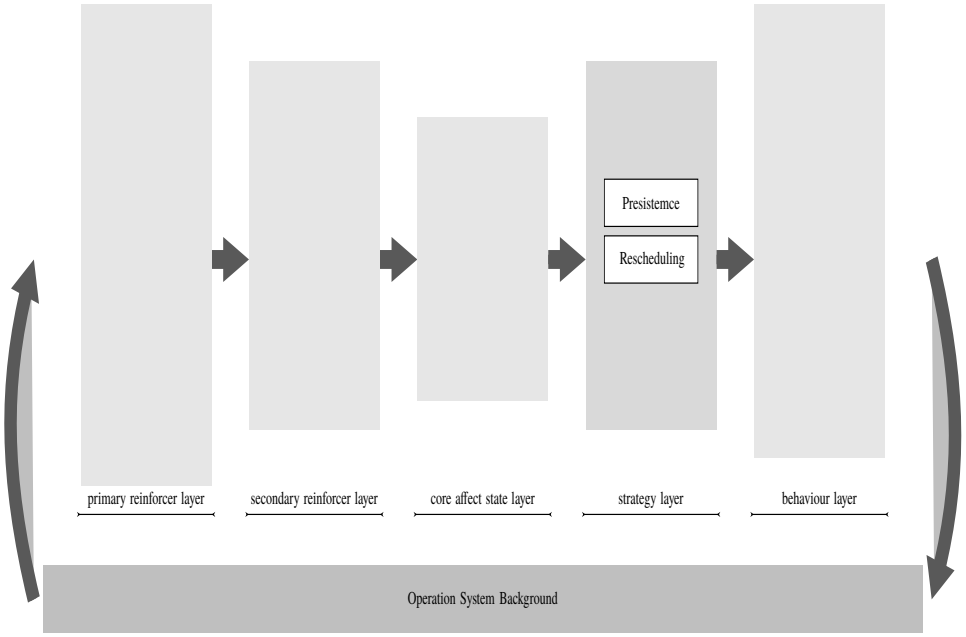


Figure 3.19: Implementation of strategies of ACMCA

providing flexibility for the task completion (see Figure 3.19). After the evolving SAOC rules are established, *Persistence* and *Rescheduling* can represent characteristic response patterns to a "frustration" event.

3.3.4.1 Persistence

Persistence stimulates a persistent pattern of sticking to its original scheme. By disregarding the influence of the uncertainty factor and the alternative schemes, Persistence sticks with its original Goal (see Section 3.3.1.2) until the end of a scenario. In this work, Persistence will keep the destination position of the Goal (see Section 3.3.1.2) intact for the executed behaviours (e.g. Path-following (see Section 3.3.5.2)) to complete.

3.3.4.2 Rescheduling

Rescheduling stimulates a rescheduling pattern to provide an alternative scheme for task completion. Under the influence of the uncertainty factor, Rescheduling selects an alternative scheme for the scenario. In this work, Rescheduling resets the Goal with an alternative position, suggesting a different position to move forward to.

3.3.5 Behaviour (Layer Five)

Behaviour aims to provide diverse executed behaviours for a robot's performance in states of a task scenario. A Behaviour is an operational element of a model applied to the robot. Through models, behaviours can influence the robot's performance at the given moment from various aspects (i.e, perception, execution, etc). Various optimal behaviours can be learned by SAOC rules and XCS agents for each state. These optimal behaviours in a state can be considered as a behaviour pattern of the state, leading to the task achievement in the final state of a scenario. In this work, behaviours include *Reflex Velocity*, *Path-following*, and *Inflation Radius* (see Figure 3.20). Each of these includes a set of specific operational behaviours that can be executed by the robot.

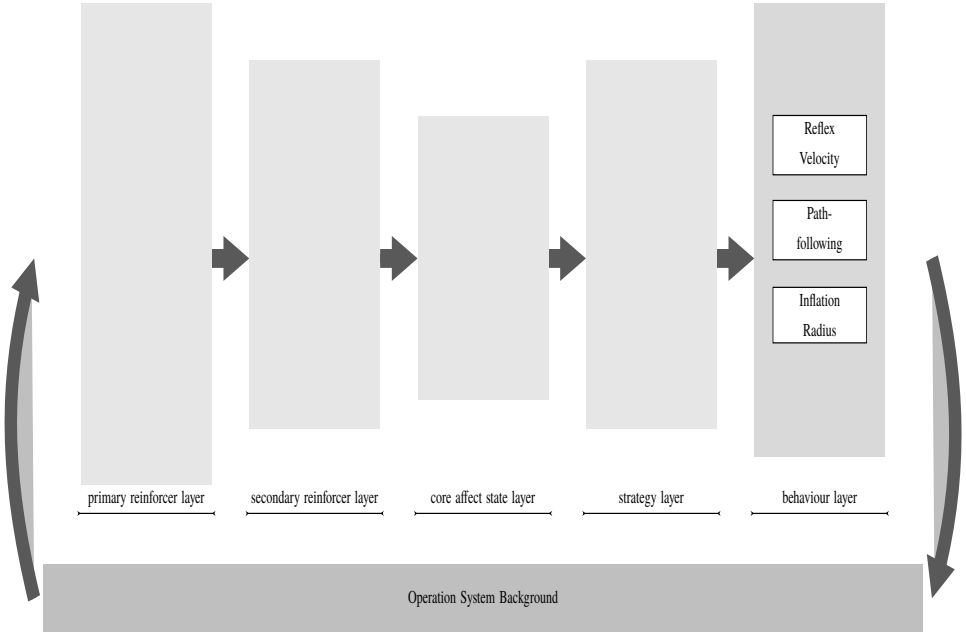


Figure 3.20: Implementation of behaviours of ACMCA

3.3.5.1 Reflex Velocity

Reflex Velocity is a behaviour that directly sets the velocity command to the motor controller of a robot. In this work, the move-base model is applied in a robot (e.g. the Pioneer (Section 2.7)) to control its movement via velocity commands. Reflex Velocities for the move-base model includes six velocity-commands that allow the Pioneer to move

in different directions, immediately responding to unpredicted collisions (see Figure 3.11). The detailed settings of Reflex Velocities are described in Table 3.2. These settings aim to create different movement of the robots and thus could be changed based on the robot's morphology. The settings are determined by the investigation of the mechanical mobility of the robot platform (the Pioneer) (see the investigation in Section 2.7). The settings of velocities can be arbitrary values in the linear range where the input velocity causes no steady-state error on the system. After the evolving SAOC rules of the Reflex agent (Section 3.3.2.1) are trained, Reflex Velocities are alternative behaviours of Path-Following for rescuing the robot from continuous collisions with obstacles that are unpredicted by the robot's path-planning model.

Table 3.2: Reflex Velocity

Index (Direction Index)	Linear Velocity (m/s)	Angular Velocity (rad/s)
NO.1 (Direction 1)	0.1	0
NO.2 (Direction 2)	0.05	-0.32
NO.3 (Direction 3)	0.05	0.32
NO.4 (Direction 4)	-0.1	0
NO.5 (Direction 5)	-0.05	-0.32
NO.6 (Direction 6)	-0.05	0.32

3.3.5.2 Path-following

Path-following is a behaviour that transfers a velocity command from a path-planning model to the motor controller of a robot. Generally, path-following is a default behaviour for most mobile robots, allowing a robot to execute velocity commands generated by a path-planning model. In this work, Path-following and its counterpart, Reflex Velocity, are two behaviours that influence the robot's executed performances. In the experiment of the emotion model (Section 5.5), random velocities are introduced to simulate velocities that are generated by the path-planning model. According to the Pioneer's mechanism (Figure 2.18) and the constraint of the experimental environments, random velocities are set in Table 3.3. Similarly to the settings in the Reflex Velocity, these settings are selected from the linear range where no steady-state error can be caused. The average

values of the settings in the Path-following are larger than that of the Reflex Velocity to simulate navigation performances.

Table 3.3: Random Velocity

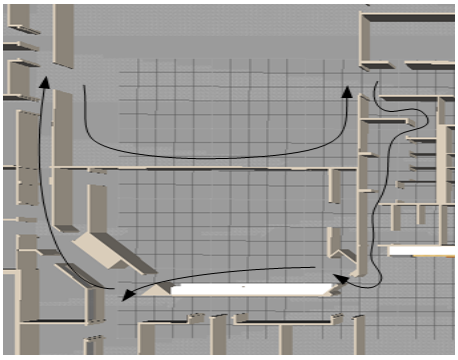
Index (Direction Index)	Linear Velocity (m/s)	Angular Velocity (rad/s)
NO.1 (Direction 1)	0.15	0
NO.2 (Direction 2)	0.1	-0.32
NO.3 (Direction 3)	0.1	0.32
NO.4 (Direction 4)	-0.15	0
NO.5 (Direction 5)	-0.1	-0.32
NO.6 (Direction 6)	-0.1	0.32

3.3.5.3 Inflation Radius

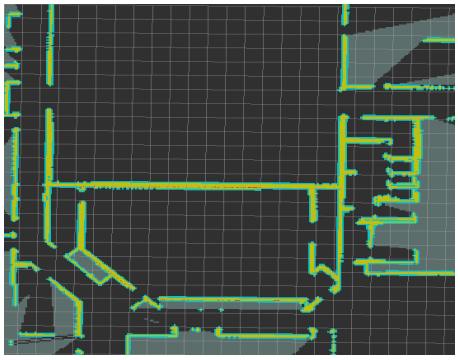
Inflation Radius is a behaviour that specifies a hyperparameter of obstacle-inflation-distance for a path-planning model. Generally, a hyperparameter is a parameter that has determining effects on the method's performance, beyond methods and implications that it is applied to. Inherited from a modifier of the previous work [17, 18], *Inflation Radius* (*IR*) specifies an obstacle-spreading distance within which the free area around the obstacle will also be marked with an obstacle-occupied cost in the perceived map (termed as costmap). For example, an overlarge value of inflation radius (e.g. 0.8 metres) will fill the space around obstacles with a large inflation (see Figure 3.21.c). Hence, narrow spaces, where the path-planning method should be able to generate a path through, will be considered as being occupied by an obstacle. As a result, different values of the IR will result in different perceived environments of the same scenario, especially in narrow spaces (see Figure 3.21).

IR has a dominant effect on admissible behaviours of the path-planning method. Because IR will directly affect the costmap that a path-planning method works on, the path-planning method would not necessarily generate a valid path if the IR is not fit for the scenario. For example, in scenario No.4 (see Figure 3.21.c), the IR with a value of 0.8 metres is appropriate for the wide-open area on the top right corner of the local zoom map, but this value will lead to an enclosed perception of the doorway. Traditionally,

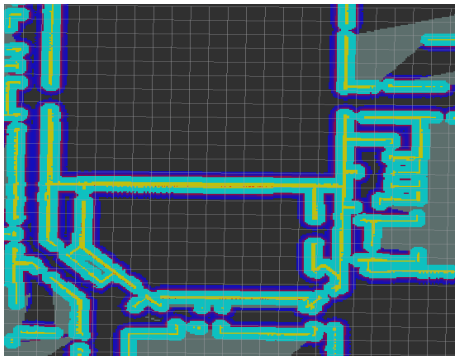
it is the engineer's responsibility to tune such a hyperparameter scenario by scenario to ensure the path-planning method can generate paths. In this work, the SAOC rules and XCS agents of the Tuning can automatically learn an appropriate IR for the generation of a valid path (see Section 3.3.2.2).



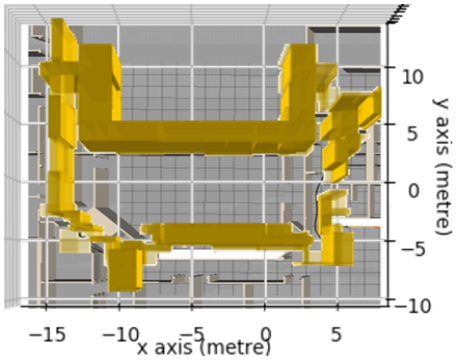
(a) Ground Truth Map



(b) Robotic Perception Map 1



(c) Robotic Perception Map 2



(d) Representation from Learned Knowledge

Figure 3.21: Different Representations of Willow Garage Office Environment. (a) Environment Viewed in *Gazebo* (targeted navigation path is illustrated by black arrows); (b) inflation radius = 0.1 metres; (c) inflation radius = 0.8 metres, (maximum value in this paper, and the default value of the navigation package is 5.5 metres); (d) TP Patterns (the yellow cuboids compared with (a).

3.3.6 Section Summary

There are three major benefits of the combination of various components in the five-layer architecture. Firstly, the diverse components provide potential search space from which

their combinations can achieve adaptive behaviours. The combination of these components, ACMCA, constructs a hierarchy of contingencies to achieve adaptive behaviours in various subsumption-architecture levels. Secondly, diverse interactions between various components can emerge in the ACMCA. These interactions of an applied robot can be explained by Constructive Theory, Appraisal Theory, and Basic Emotion Theory. Thirdly, the combination of components is also based on the SAOC contingency, leading to a novel contingency-based separating of the search space.

ACMCA will have better performances when these five layers are complete for perception, decision making, and executions. However, the rest of ACMCA will not break if only one component is removed. The first layer is the layer of primary reinforcer. The robot perceives the environment through primary reinforcers that are connected with sensors. As the input of ACMCA, removing this layer deprives the robot's external perception, leaving a "blind and deaf" robot wandering in the environment.

The second layer contains secondary reinforcers. Secondary reinforcers construct SAOC rules based on their primary reinforcers. These SAOC rules provide estimations on the perceived stimuli. When this layer is removed, the robot has no estimations from perception. This will significantly damage the system's adaptation, leaving a robot that never learns from its past experience.

The third layer contains core affect states. Core affect states establish flexible connections between perception and execution. Core affect states summarise the perception, thus advocating behaviours for the execution. The robot can learn to select strategies/patterns responding to the current perception. If the system removes the core affect state layer, the robot cannot switch its behaviours flexibly.

The fourth layer contains strategies that subsume behaviours. The subsumption of behaviours decreases the SAOC search space, thus increasing the efficiency of the learning agents in learning SAOC rules. If this layer is removed from the architecture, the learning agents will face a larger search space than the current one. As a result, the time consumption for training the robot will increase.

The fifth layer contains behaviours as the outputs of the ACMCA. Behaviours can affect the robots executions from a module level or a responsive level. The removal of this layer will remove the ACMCA's impact on the robot. That is, an Outcome/effect of the robot's behaviour is no longer related to the Action that SAOC rules advocate. The learnt SAOC rules can no longer be trusted, although the learning agents and the rest of

the architecture functions normally.

It is also noted that SAOC rules and the emotion mechanisms essentially lead to a hierarchy of contingencies, which can achieve adaptive behaviours in various subsumption-architecture levels. On one hand, these diverse SAOC rules separate the entire search space by interpretable contingencies, so that independent learning agents can effectively learn different types of SAOC rules in separated search space. Therefore, SAOC rules and the emotion mechanisms essentially guide searching directions that are not easily accessible in other approaches. On the other hand, emotion mechanism creates a novel contingency-based subsumption approach for robotic systems. SAOC rules in the high-level indicate a contingency-based subsumption approach that subsumes low-level components. This unique contingency-based subsumption approach has the potential to create completely autonomous robots that the classical behaviour-based subsumption approach fails to generate. In addition, the emotion mechanisms allow the robot to learn diverse SAOC rules that improve the adaptation of the robot. SAOC rules are achieved by secondary reinforcers, core affect states, and strategies, while other approaches/architectures have not yet achieved these SAOC rules.

The emotion mechanisms are inspired by Constructive Theory, Appraisal Theory, and Basic Emotion Theory. Constructive Theory emphasises the reinforcer, which associates Stimulus with responsive Action and their sequential Outcome (i.e. reward). This work constructs various reinforcers (e.g. secondary reinforcers), which establish SAOC rules from experiments. Appraisal Theory advocates that emotion states (e.g. core affect states) are elicited by stimuli. These emotion states can be summarized in a two-dimensional space (e.g. core affect space) by arousal and hedonic values of emotion states. The work does not preset any high-fidelity emotional states. In contrast, artificial core affect states are elicited in our experiment. Basic Emotion Theory focuses on the action preparation on behaviours. Each basic emotion is associated with a specific behaviour pattern. This work contains Strategy nodes, that include different behaviour patterns, which are connected with elicited core affect states. Therefore, the work bridge these three main-stream emotion theories by applying them in perception, decision-making, and execution of the robot.

In addition, this work allows the robot to learn adaptive behaviours that can be labelled with emotional states. As mentioned above, although we do not preset any emotion states, the robot can learn to perform behaviours that can be interpreted as

emotional responses. In the experiment of Reflex-learning (see Section 5.2), the robot can perform "aggressively" to avoid continuous collisions by spinning and pushing away obstacles. Although the robot never matched the human's concept of angry, people should keep themselves in a safe location when the " pissed-off" robot scratched the obstacles in the experiment.

3.4 Chapter Summary

This chapter described the methodology of ACMCA, which learns diverse affective solutions for a robotic task. Section 3.2 introduced five components of the affective solutions inspired by the three major emotion theories: Constructive Theory, Appraisal Theory, and Basic (Emotion) Theory. The solutions extend *primary reinforcer* and *behaviours* from the previous solution of the reinforcers-emotion-modifiers mappings, and introduces *secondary reinforcer*, *core affect states*, and *policy* as its novel components.

Section 3.3 introduced five sequential layers and nodes within these layers. In the primary reinforcer layer, the primary reinforcers, including *Touch*, *Path*, *Occupation*, *Goal*, *Position*, and *Reward*, provide raw stimuli from recognising environmental factors. In the secondary reinforcer layer, the secondary reinforcers, including *Reflex*, *Tuning*, and *Deliberation*, construct instrumental stimuli and establish various populations of SAOC rules for affective responses. In the core affect space, core affect states link between the first two preception layers and the last two execution layers, expecting four basic emotions to emerge in ACMCA for various scenarios: *Happiness*, *Fear*, *Frustration*, and *Calm*. In the strategy layer, the strategies, including *Persistence* and *Rescheduling*, provide flexible schemes for task completion. Finally, in the behaviour layer, the behaviours, including *Reflex Velocity*, *Path-following*, and *Radius Inflation*, focus on the operational behaviours to improve performance.

This chapter also proposes contingency-based subsumption operations to construct ACMCA. With underlying machine learning algorithms, contingency-based subsumption operations can automatically construct a multilayer and multiple hierarchy cognitive system, what look-up tables would be hard to construct. The underlying machine learning algorithms will be introduced in Chapter 4. Experiments of ACMCA and proposed algorithms will be introduced in Chapter 5.

Chapter 4

Algorithms

4.1 Introduction

The previous chapter introduced the methodology of Affective Computing Multilayer Cognitive Architecture (ACMCA), which aims to learn a novel affective solution for robotic tasks. Computing nodes are distributed among the five-layer hierarchy of ACMCA. Each node represents a separate solution-component, and all the nodes cooperate to achieve the affective solution. The cooperations between nodes are automatically established through the contingency-based subsumption operations, which allow nodes in high-level of the hierarchy to subsume nodes in low-level by encapsulated contingencies [1]. These subsumption operations of high-level nodes are realised by an underlying machine learning technique. As the robot interacts with the environment, the contingency-based subsumption can establish an appropriate network of ACMCA computing nodes, providing an affective solution toward task completion.

The XCS algorithm is selected as the underlying machine learning technique for its symbolic interpretation, reasoning ability, and affordable computing budget. However, although the standard XCS algorithm has comparatively significantly better performance in binary problems (e.g. n-bit multiplexer problem) than other EC algorithms [104], it has shortfalls that hinder its robotic applications (Section 4.2.2 and 4.3.1). Therefore, two variants of XCS algorithms, the *mitosis* approach and the *XCSCR*, are proposed in this chapter to mitigate these shortfalls that challenge real-world robotic applications.

[1] They are Stimulus-Action-Outcome Contingency (SAOC) rules, which are introduced in Section 3.2.2

Real-world robotic applications bring challenges for the standard XCS algorithm. Two universal challenges occur when the standard XCS algorithm is applied in ACMCA for a mobile robot. The first challenge for the standard XCS algorithm is the noisy environment of a real-world robot. Different from binary problems, a real-world robotic application will unavoidably contain noisy data niches. These noisy data niches interfere with the performance of the rule-based standard XCS algorithm. This is because its innate "overgeneralized tendency" can magnify the negative impacts of noisy data niches on accurate rules (Section 4.2.2). Because of these interferences, the standard XCS algorithm will (1) lack robustness to maintain its accurate rules, and (2) have a tendency to become stuck at a local optimum. Therefore, nodes that deploy the standard XCS algorithm fail to achieve maximally generally and accurate solution-components.

This work proposes the *mitosis approach* for the XCS algorithm (Section 4.2) to meet the first challenge in real-world robotic applications. Inspired by the mitosis procedure in the biological cell-division circle, this approach introduces novel procedures into the standard algorithm's evolutionary process. These procedures allow the parent rule to pass its accurate elements down to children rules in the evolutionary process. The passing of accurate elements becomes an accuracy pressure that is introduced into the algorithm's iteration loop, balancing the overgeneralized tendency and the negative impacts of noisy data niches on accurate rules. Driven by the accuracy pressure, the mitosis approach can perform better than the standard approach in terms of prediction accuracy, pattern robustness and pattern accuracy (Section 5.3.2). After deploying the mitosis approach to assist subsumption operations, high-level nodes can achieve accurate solution-components with interpretable symbolic meanings.

The second challenge is that the XCS algorithm's credit assignment method for robotic tasks is flawed. In a multistep navigation scenario, a robot receives a long-term credit at the final step. Then, a credit assignment method is activated to assign this final credit to previous steps of a policy [2] that led to that credit. As the robot repeats the task, an ideal credit assignment method can identify global optimal policies from a vast policy space. However, the functionality of the XCS credit assignment method (Section 2.5.2) might be not fast enough for robotic applications in the real world. This

[2]A policy is a state-action mapping of an agent. A "state" represents the state of the world where the agent is, i.e. the surrounding environment of a mobile robot. An "action" indicates what action the agent should take in that state, e.g. moving forward.

is because the credit assignment method requires sufficient rewards to search global optimal policies from the vast policy space, yet long-term, positive rewards are so scarce that it is insufficient to guide the searching. That is, especially at the initial learning phase, it is rare for the robot to achieve the goal and receive long-term, positive rewards. Therefore, if it only relies on the long-term, positive rewards, the credit assignment method cannot learn policies effectively for multistep robotic applications.

This work proposes the *XCSCR*, an *XCS algorithm with a combined reward method* (Section 4.3), to meet the second challenge. The XCSCR explores both long-term and short-term rewards to search for optimal policies. The XCSCR technique also adjusts the rewards' impacts on the policy search by different training phases to encourage an early emergence of optimal policies. The XCSCR evolves its population of policies toward optimal policies by the pressure that is introduced by the current best policies. As a result, the XCSCR enables the policy to emerge earlier and more frequently than the existing benchmark approaches in multistep problems (Section 4.3). Therefore, a robot with the XCSCR can handle a multistep scenario more effectively than those with the benchmarked algorithms.

4.2 Mitosis Approach

This section presents the *mitosis approach* for the XCS algorithm. We start with an introduction of the learning process in the standard XCS algorithm (Section 4.2.1) as the benchmark for the mitosis approach. The next subsection (Section 4.2.2) discusses the overgeneralized tendency of the standard XCS Algorithm, which is the reason why the overgeneral pressure frequently dominates the evolutionary processes of XCS. After identifying the overgeneralized tendency, subsection 4.2.3 introduces where the novel mitosis approach is added to the benchmark learning process. The following subsections 4.2.4 and 4.2.5 detail two mitosis procedures during the learning process, generating accurate "descendants" and creating an accuracy pressure against the overgeneralized tendency. The final subsection 4.2.6 provides a summary of the mitosis approach.

4.2.1 Main Loop of Standard Approach of XCS algorithm

A learning agent of the standard XCS algorithm learns classifiers through its iteration loops (Section 2.5). In its iteration loops, the XCS learning agent interacts with an environment in a sequence of operations: perceiving the current state from the environment, selecting an action by the perceived state and classifiers, executing the selected action to the environment, and updating classifiers by the reward from the environment (Figure 4.1). Classifiers engage in these sequential operations to update their worth. At the beginning of an iteration loop, the XCS agent perceives a current state, which is in a niche from the environment, as a perception. Then, the XCS agent initiates its Matching method to choose classifiers from the Population ($[P]$). The niche-coverage of the condition part of the chosen classifiers must be able to cover the perception. If there is no classifier in $[P]$ that meets this requirement, the Covering method will be activated to generate a perception-covered classifier. The chosen classifiers or the perception-covered classifier form a Match set ($[M]$). As an iteration loop progresses, the Selection method chooses an Action from the available Actions, which come from the classifiers in $[M]$. Classifiers with the chosen Action in $[M]$ forms an Action set ($[A]$). After the execution of the chosen Action, the agent will receive a reward from the environment. Based on the reward, classifiers in $[A]$ are updated (Section 2.5) and return to $[P]$ at an end of the iteration loop. In addition to these methods, a Genetic Algorithm (GA) method will be activated conditionally within $[A]$ to generate new classifiers, and a Deletion method removes excess classifiers from $[P]$. The interactions between these two methods can create a generalization tendency, which has the potential to create overgeneral classifiers and become a fundamental problem for the standard algorithm.

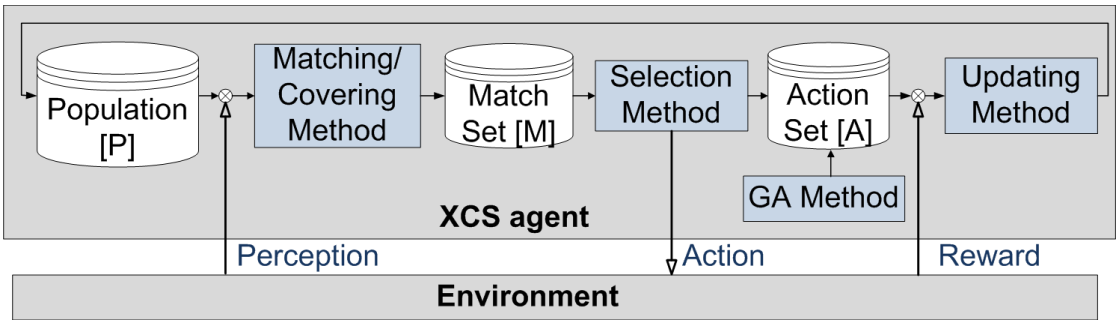


Figure 4.1: Learning Processes in Standard XCS

4.2.2 Overgeneralized Tendency of XCS Algorithm

The standard XCS algorithm suffers from an overgeneralization tendency. This tendency creates overgeneral classifiers by extending the niche-coverage of their condition parts. It is an intrinsic tendency [69] that originates from interactions between niched evolution (i.e. Genetic Algorithm (GA) in $[A]$) and panmictic deletion (i.e. removal of excess rules from $[P]$) [105]. In niched evolution, the classical GA of the standard XCS creates a much stronger generalisation pressure than specification pressure. That is, GA tends to create more general classifiers than specific ones because the former has a closer Hamming distance than the later [106]. Yet, the panmictic deletion aims to remove unfit classifiers from the population, and the fitness of a classifier is not completely reliant on its generalisation. As a result, the standard XCS algorithm tends to create overgeneral classifiers if the algorithm has no explicit method to balance this pressure.

The XCS algorithm lacks the ability to distinguish overgeneral classifiers from other inaccurate classifiers. Generally, inaccurate classifiers contain inaccurate niche-coverage. Overgeneral classifiers distinguish other inaccurate classifiers because overgeneral ones mix major accurate niche-coverage with little inaccurate niche-coverage. In the GA procedure, the crossover and mutation operations can extend the niche-coverage of an accurate classifier to inaccurate niche-coverage of an overgeneral classifier. Before the inaccurate niche can encounter a negative reward, the overgeneral classifier can subsume the accurate classifier because of its larger niche-coverage than the accurate one, leading to the accurate classifier being replaced by the overgeneral one. When the inaccurate niche eventually affects the panmictic deletion method, the overgeneral classifier can be deleted from $[P]$. In this case, as accuracy classifiers have been driven out of $[P]$, no classifier existing in $[P]$ can accurately cover the data-niche. As a result, the over-generalization tendency causes the following problems [107]:

1 An accurate classifier lacks robustness to maintain its accurate state;

2 As the overgeneral classifiers prosper, classifiers tend to be stuck at a local optimum, where those classifiers stop evolving toward maximally general and accurate classifiers.

The standard XCS algorithm's performance is not robust in a noisy environment of typical robotic real-world applications. Noisy data from a noisy environment are

inaccurate instances, which interfere with the performance of accurate classifiers. When the overgeneral pressure frequently dominates the evolutionary processes, an accurate classifier can eventually be driven out by its overgeneral counterpart because of these inaccurate instances. Because the standard approach cannot distinguish accurate niche-coverage from inaccurate niche-coverage in overgeneral classifiers, accurate classifiers are rare, especially in a noisy environment.

To amend the overgeneral pressure that frequently dominates the evolutionary processes of XCS, this work proposes the mitosis approach which introduces a novel accuracy pressure into the algorithm. The mitosis approach is inspired by the biological cell's mitosis procedure in which an original cell passes down its chromosome to two new cells. Mother chromosomes of the original cell are replicated and then are separated into two daughter nuclei for new cells. Similarly, the proposed artificial mitosis method will generate child classifiers that inherit an accurate niche-coverage from a parent classifier but abandon inaccurate niche-coverage. Compared to the classical evolutionary process that introduces overgeneral pressure, the accuracy pressure introduced by the mitosis approach allows generated child classifiers to move towards accuracy. As training progresses, the accurate children will eventually replace their inaccurate parents in a classifier population. As a result, the mitosis approach is anticipated to increase the accuracy of the entire classifier population.

4.2.3 Mitosis Approach Overview

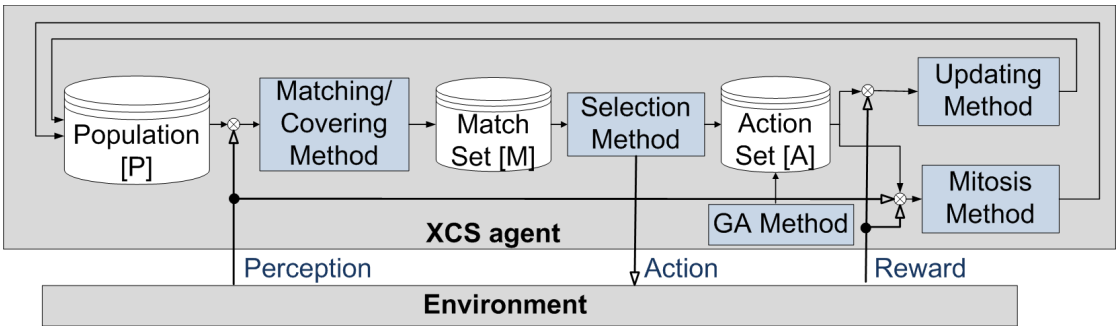


Figure 4.2: Mitosis Approach in XCS algorithm's Learning Processes.

The mitosis method is added into XCS for the mitosis approach (comparing Figure 4.2 with the standard counterpart in Figure 4.1). Structurally, the mitosis method is

in parallel to the Updating method, which updates the classifier Population according to a reward. Yet the mitosis method needs to be activated before the execution of the updating method so that the qualified overgeneral classifiers can be processed by the mitosis method. Otherwise, the qualified overgeneral classifiers could be modified by the activated GA procedure [3] of the updating method. Because the GA procedure is an evolutionary process without a specific evolutionary direction, the updating method could remove the accurate elements from overgeneral classifiers. This is what the mitosis process aims to preserve. The mitosis method introduces an accuracy pressure to guide its evolutionary process for a more accurate generation of children than the existing one. The mitosis proceeds the updating method and GA procedure so does not interfere with those procedures behind.

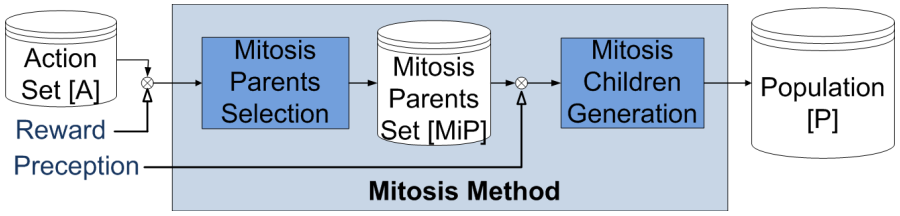


Figure 4.3: Mitosis' Two-step Procedure.

A classifier's generation process takes two steps in the mitosis method (see Figure 4.3). Firstly, the mitosis-parents-selection procedure chooses qualified overgeneral classifiers from $[A]$. This selection procedure allows the following procedures to be activated only by qualified overgeneral classifiers, potentially improving the efficiency of the mitosis process. All the qualified overgeneral classifiers are denoted as mitosis parents and are preserved in a mitosis parent set ($[MiP]$) waiting for the second step. The second step, mitosis-children-generation procedure, will generate new classifiers (denoted as mitosis children), and inserts them into $[P]$. The procedure removes inaccurate niche-coverage from the mitosis parents and preserves accurate niche-coverage and other accurate elements for the mitosis children. These accurate mitosis children in $[P]$ create the accuracy pressure to amend the general pressure in $[A]$.

[3] A procedure is one computing progress in a method, and a method can include multiple procedures.

4.2.4 Mitosis-Parents-Selection Procedure

The mitosis-parents-selection procedure identifies mitosis parents which have been potentially affected by the general pressure yet contain accurate elements. Three requirements are provided to identify mitosis parents (see Algorithm 1). Firstly, mitosis parents are classifiers that made an incorrect prediction in the current iteration. This feedback of prediction will directly separate inaccurate classifiers from accurate classifiers. The parents cannot be created by the Covering method (Section 4.2.1) because this necessary feedback is not available when the Covering method is activated. Secondly, mitosis parents must have achieved the maximum accuracy (ϵ_0) [4] before this iteration. The statistics of the maximum accuracy indicates that substantial accurate elements are contained in the current classifier. This requirement will filter overgeneral classifiers from the rest of the incorrect classifiers. Thirdly, mitosis parents have to reach the value of the maximum reward ($threshold.prd$). This requirement attempts to identify qualified overgeneral classifiers that are close to optimum solutions. If a classifier's absolute value of the predicted reward ($cl.prd$) has achieved the value of the maximum reward, this classifier is more likely to reach the global optimum solution than those who have not. If $[MiP]$ is empty, the next procedure, the mitosis-children-generation procedure, will not be activated. Therefore, the mitosis-parents-selection procedure potentially increases the efficiency of the mitosis approach by focusing on qualified overgeneral classifiers. In addition, without this procedure to filter out unqualified classifiers, the mitosis approach risks the introduction of an undesirable over-specified pressure into $[P]$.

Algorithm 1 *Mitosis Parents Selection*

Inputs: Reward R, Action set $[A]$, Mitosis Parents Set $[MiP]$, a classifier cl, a classifier's accuracy $cl.acc$, a classifier's predicted reward $cl.prd$, a preset difference threshold of predicted-reward $threshold.prd.diff$.

Outputs: Select qualified *mitosis parents*.

1: **function** SELECT MITOSIS PARENTS(R, $[A]$)
2: $[MiP] \leftarrow empty$
3: **for** cl in $[A]$ **do**

[4] ϵ_0 is a common statistics parameter of the XCS algorithm, which sets the accuracy threshold for classifiers. If a classifier's accuracy is beyond this threshold, this classifier can be considered as an accurate one.

4: **if** $cl.acc > threshold.acc$ and $cl.prd > threshold.prd$ **then**
5: **if** $|cl.prd| < |R|$ and $|cl.prd - R| > threshold.prd.diff$ **then**
6: $[MiP] \leftarrow cl.id$
7: **end if**
8: **end if**
9: **end for**
10: **return** $[MiP]$
11: **end function**

4.2.5 Mitosis-Children-Generation Procedure

The mitosis-children-generation procedure passes accurate elements from the selected mitosis parent to child classifiers after the mitosis-parents-selection procedure (see Algorithm 2). If there are qualified mitosis parents in $[MiP]$, the mitosis-children-generation procedure will initiate a loop of the Mitosis-division procedure to iterate each mitosis parent. In the loop iteration, the Mitosis-division procedure generates new classifiers based on accurate elements that are extracted from the qualified mitosis parent. This generation is an evolutionary procedure (Algorithm 3) including two sub-procedures: chromosome-morphing procedure and telophase procedure. The chromosome-morphing procedure (Algorithm 4) separates accurate elements from an inaccurate data-niche through the chromosome morphing operation, providing the condition parts for mitosis child classifiers. Next, the telophase procedure (Algorithm 5) combines these condition parts with action parts and statistics to generate completed mitosis child classifiers.

Algorithm 2 *Mitosis Children Generation*

Inputs: Mitosis Parents Set $[MiP]$, Mitosis Children Set $[MiC]$, Stituation σ, Population $[P]$

Outputs: Target: Generate *mitosis children* set and insert it into populstion set.

1: **function** GENERATE MITOSIS CHILDREN(σ, $[MiP]$, $[P]$)
2: $[MiC] \leftarrow empty$
3: **for** cl in $[MiP]$ **do**
4: $[NewBorn] \leftarrow$ MITOSIS DEVISION(cl, σ) //parent divide mitosis children are returned in $[MiC]$
5: **end for**

```
    if [MiC] is not empty then
        [P] ← [MiC]
    end if
    return [P]
end function
```

The chromosome-morphing procedure generates accurate condition parts for child classifiers by detecting and removing inaccurate niche-coverage within the mitosis parent. The detection is based on overlaps between the current perception and the niche-coverage of the condition part of the mitosis parent. Because the mitosis parent is inaccurate under the current perception, these overlaps must contain an inaccurate niche-coverage. Thus, condition parts that have removed one of the overlaps have a more accurate niche-coverage for the mitosis child. By extracting the accurate niche-coverage for new classifiers, the chromosome-morphing procedure actually introduces an accuracy pressure to affect the evolutionary process.

The telophase procedure generates new mitosis children by combining components that inherit accurate components from the parent classifier. These components are the condition part, the action part, and the standard statistics. The condition part comes from the chromosome-morphing procedure, the action part directly inherits from the mitosis parent. The statistics of child classifiers also inherit from their parent, except the numerosity and the experience. The numerosity specifies the number of copies of a classifier. The experience indicates how many times a classifier has been activated. As these two statistics measure the new classifier's performance, they are not inherited from the parent classifier. In this procedure, these two statistics are reset to their initial values of one, following the default settings of a newborn classifier.

Algorithm 3 *Mitosis Devision*

Inputs: An Individual Mitosis Parent cl, Stituation σ, New Generation's Gene Set $[Gene]$

Outputs: Target: Generate *mitosis children* through the mitosis process

```
function MITOSIS EXECUTION PROCESS(cl, σ )
    [Gene] ← CHROMOSOME MORPHING(cl, σ)    //chromsomes devision and morphing, and reunite into new chromsomes
    [NewBorn] ← TELOPHASE(cl, [Gene] )
```

4: **return** $[NewBorn]$
5: **end function**

Algorithm 4 Chromosome Morphing

Inputs: Situation σ, A Dimension of Situation $attribute$, An Individual Mitosis Parent cl, A Mitosis Parent's Chromosome Set$[Chs]$, A Chromosome in The Chromesome Set $chromosome$, Morphed Chromatids $[MoCht]$, A Chromosome/Chromatid Set Prepared for Future Mitosis Children $[ChCht]$, New Generation's Gene Bank $[Gene]$

Outputs: Target: Chromosome breaks and morphs into new chromosome for next generation.

1: **function** CHROMATIDS MUTATION(cl, σ, $[MoCht]$)
2: $[Chs] \leftarrow [cl]'s\ ChromosomeSet$
3: $[MoCht] \leftarrow empty$
4: **for** $chromosome$ in $[Chs]$ **do**
5: $i \leftarrow 0$
6: $Attribute \leftarrow \sigma.attribute[i]$
7: $chromatid \leftarrow empty$
8: // Begin morphing for each chromosome individed
9: **if** $attribute.low <= chromosome.low$ and $chromosome.high <= attribute.high$ **then**
10: pass // no chromatids generated
11: **end if**
12: **if** $chromosome.low <= attribute.low$ and $chromosome.high <= attribute.high$ **then**
13: $chromatid.low = chromosome.lower$
14: $chromatid.high = attribute.low$
15: $[MoCht] \leftarrow chromatid$
16: **end if**
17: **if** $chromosome.low <= attribute.low$ and $chromosome.high <= attribute.high$ **then**
18: $chromatid.low = chromosome.lower$
19: $chromatid.high = attribute.low$
20: $[MoCht] \leftarrow chromatid$
21: **end if**

22: **if** $chromosome.low <= attribute.low$ and $attribute.high <= chromosome.high$ **then**
23: $chromatid.low = chromosome.lower$
24: $chromatid.high = attribute.low$
25: $[MoCht] \leftarrow chromatid$
26: $chromatid.low = attribute.high$
27: $chromatid.high = chromosome.high$
28: $[MoCht] \leftarrow chromatid$
29: **end if**
30: **if** $[MoCht]$ is not empty **then**
31: **for** $chromatid$ in $[MoCht]$ **do**
32: // Begin envelope individual chromatids as new chromosomes set for an individual
33: combine $chromatid$ with other chromosomes in $[Chs]$ to produce $[ChCht]$
34: $[Gene] \leftarrow [ChCht]$
35: **end for**
36: **end if**
37: **end for**
38: **return** $[Gene]$
39: **end function**

Algorithm 5 Telophase

Inputs: Mitosis Parent $[cl]$, A Chromosome/Chromatid Set Prepared for Future Mitosis Children $[ChCht]$, New Generation's Gene Bank $[Gene]$, New Born Generation $[NewBorn]$, a new born child $cell$, a child's Chromosome Set $cell.chromosomes$, a child's cytoplasm $cell.cytoplasm$

Outputs: Target: Generate children cells by enveloping all elements of a classifier together.

1: **function** TELOPHASE(cl, $[Gene]$)
2: **for** $[ChCht]$ in $[Gene]$ **do**
3: $cell.chromosomes \leftarrow [ChCht]$ //fetch a new suit of chromosomes
4: $cell.cytoplasm \leftarrow$INHERIT CYTOPLASM($cl$) //inherit statistics and action from parent

5: $[NewBorn] \leftarrow cell$ // envelope a new cell
6: **end for**
7: **return** $[NewBorn]$
8: **end function**

4.2.6 Mitosis Method Summary

This section proposed the mitosis approach for the XCS algorithm, which introduced an accuracy pressure to amend the overgeneralized tendency of the benchmark XCS algorithm. The mitosis approach excluded inaccurate niche-coverage of overgeneralized classifiers and passed on only the accurate niche-coverage to child classifiers. In this approach, accurate classifiers can be robust against interferences from noise. Therefore, the mitosis approach is deployed on learning agents of ACMCA nodes to learn various environmental patterns in experiments (Chapter 5). The analysis of the mitosis approach is presented in the discussion of the results of the experiments (Section 5.3.2).

Generally, this algorithm cannot work for completely unseen data. It depends on the data distributions between the training dataset and the unseen one. If they have similar data distributions, the algorithms can work on the unseen dataset. If they have different data distributions, the algorithms cannot work as effectively as its performance on the training dataset. For example, an indoor, crowded office will generate different sensor (i.e. LIDAR) distributions to a wide-open playground. What the algorithm learned in the office, such as a speed limitation, will not be necessarily applied to the playground.

4.3 XCSCR Method

This section proposes *XCSCR*, an *XCS with a combined reward method*, to guide the search for global optimal policies in multistep problems. This section contains nine subsections. Subsection 4.3.1 introduces the problem: the vast policy-searching space of multistep problems that occurs when a robot assigns a final credit to previous policies in a life-long learning scenario. The next two subsections discuss the background of the credit assignment and reward updating procedures. Subsection 4.3.2 introduces the reward updating procedure in single-step scenarios, and then Subsection 4.3.3 extends this procedure to multistep scenarios. Subsection 4.3.3 also introduces another bench-

mark procedure, which is proposed to bootstrap the policy-searching speed for robotic applications. The next five subsections detail the XCSCR method. Subsection 4.3.4 provides a brief overview of the method, and Subsections 4.3.5, 4.3.6, 4.3.7, and 4.3.8 detail the four XCSCR mechanisms that differentiate the impact of various long-term and short-term rewards. Finally, Subsection 4.3.9 provides a summary of the proposed XCSCR method.

4.3.1 Vast Policy-Searching Space of Multistep Problems

An XCS agent can be considered to be a Reinforcement Learning (RL) agent when it seeks to improve performance from its interactions with an environment. An XCS agent performs actions in an environment and receives rewards as feedback. In multistep scenarios, an agent takes steps and performs sequential actions in an environment before the agent achieves a reward in the final step. The rewards are distributed by a reward method within the agent to estimate the contributions of each action. Actions that would contribute to the maximum reward are selected as optimal policies as estimations become accurate. By successively applying learned policies, an agent can optimise its performance in the environment.

An RL agent often faces a vast policy space when searching for a global optimal policy in multistep scenarios. A multistep scenario, such as a way-finding maze, requires an RL agent to take steps and perform sequential actions to complete a task. For example, the RL agent is required to complete a navigation task by choosing four directions within ten steps in the maze 4 environment (Section 5.6.1). There are three global optimal policies compared with a policy space with naïvely 4^{10} policies[5]. The vastness of a policy-searching space increases the difficulty of solving the credit assignment problem. Specifically, the vastness of a policy-searching space requires efficient estimations of the worth of actions within policies to search for an optimum policy.

The standard reward methods cannot efficiently estimate actions and policies when positive rewards are scarce [64]. An estimation relies on assigned rewards through Bellman's equation (see Section 4.3.3), which does not discriminate long-term rewards between positive ones and negative/neutral ones. Actions and policies can be estimated effectively when positive rewards are frequently achieved as they guide the search. If the

[5]the number of possible combinations of four actions that the agent can attempt to take in ten steps

number of the negative/neutral rewards is overwhelming, the estimations generated by scarce positive rewards will be quickly offset by the interference of the negative/neutral rewards. As the reward method fails to estimate actions and policies efficiently, global optimal policies are elusive to the RL agent in multistep problems.

These problems are particularly apparent in real-world robotics experiments where it is impractical to conduct millions of iterations, which are common in simulations. Similarly, although modern computing power is increasing, those available to an autonomous robot are often limited. Previous work by Williams et al. [17, 18] has adapted Learning Classifier Systems for policy learning in real-robots, but this work still suffered from credit assignment problems regarding scarce rewards in a real-world environment. In addition, the work did not take into account the benefits of an autonomous agent in terms of being able to identify short-term reinforcement, e.g. the ability to note a collision during a policy learning trial. Therefore, a reward method that addresses these credit assignment problems would be beneficial to real-world robotic applications.

This section proposes XCSCR, a combined reward (CR) method for an Accuracy-based Learning Classifier System (XCS) algorithm, to achieve global optimal policies in multistep problems. That is, XCSCR needs to introduce immediate short-term negative rewards to encourage exploration when long-term, positive rewards are scarce. XCSCR also exploits the policies that led to long-term, positive rewards to balance potential interferences from long-term, negative rewards. In addition, XCSCR emphasises the effect of long-term, positive rewards, and includes a dynamic threshold to drive policies to move toward globally optimal ones. The effects of XCSCR on policy search will be demonstrated with experiments using interpretable multistep maze problems.

4.3.2 Reward Updating Procedure of Standard XCS Approach

A standard XCS agent estimates rules through iterations of these interactions. Based on how the agent perceives the environment in each iteration, rules in the XCS agent will advocate an action as the agent's effect on the environment (Figure 4.3.a). All rules in the XCS rules' population [P] that match the current perception will form a Match Set [M] through the match filter. Next, an action from [M] is selected by the selection filter through either an exploration or an exploitation method. In exploration, the selection filter selects an action randomly from all options. In exploitation, the selection filter

selects the most promising action with the maximum worth in [*M*] (Equation 4.6). The selected action will be executed by the agent and rules in [*M*], who vote for the executed action, will form an Action Set [*A*]. After the action execution, the agent will immediately receive a step reward r_a from the reward filter as feedback from the environment in single step problems. The worth (fitness) and other statistics of the rules in [*A*] are updated according to the step reward r_a. As the iterations progress, the fitness reflects the utility of a rule to guide the evolutionary search of better solutions (policies of actions).

The reward filter updates each rule in [*A*] by updating the statistics of a rule, including the predicted reward r_p, the prediction error ϵ, and the fitness F based on its r_a. The predicted reward r_p is updated by a learning rate β (Equation 4.1).

$$r_p = r_p + \beta * (r_p - r_a) \tag{4.1}$$

Prediction error ϵ is also updated in a similar way (Equation 4.2).

$$\epsilon = \epsilon + \beta * (|r_p - r_a| - \epsilon) \tag{4.2}$$

Finally, fitness fit, the worth of the rule, is updated through calculations of absolute accuracy κ and relative accuracy κ' (Equations 4.3, 4.4, 4.5).

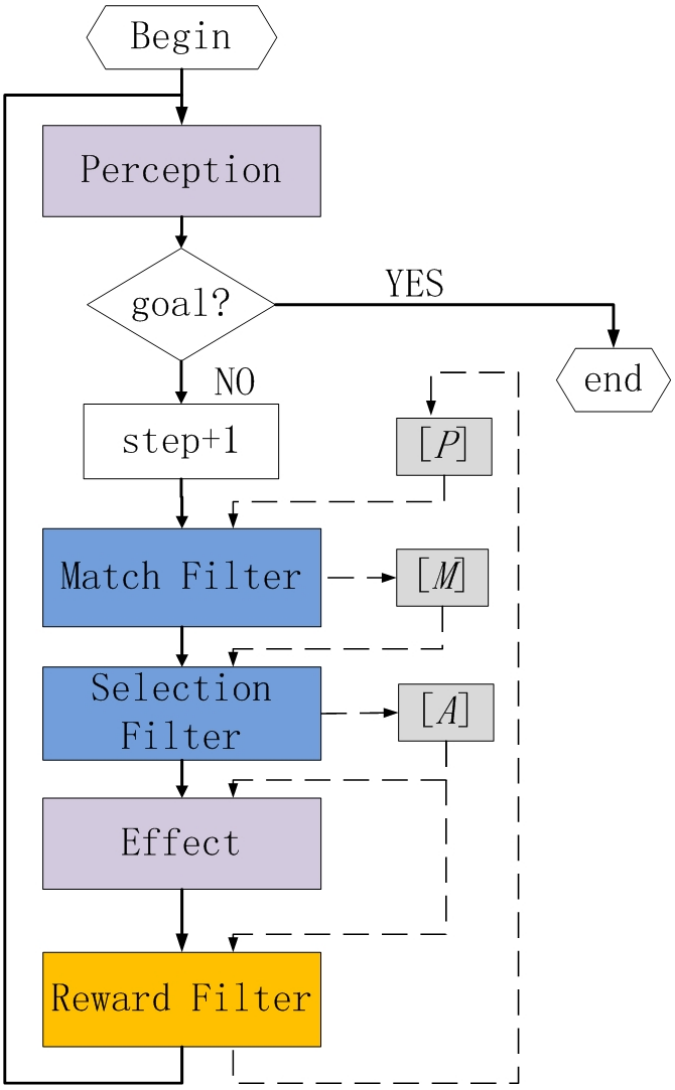
$$\kappa = \begin{cases} 1, & if\ \epsilon \leq \epsilon_0 \\ (\epsilon/\epsilon_0)^{\nu}, & otherwise. \end{cases} \tag{4.3}$$

$$\kappa' = \kappa / (\textstyle\sum_{[A]} \kappa) \tag{4.4}$$

$$fit = fit + \beta * (fit - \kappa') \tag{4.5}$$

$$worth_{a_j} = \frac{\sum_{cl_k \in [M] | a_j} r_{p_k} * fit_k}{\sum_{cl_k \in [M]} fit_k}, \tag{4.6}$$

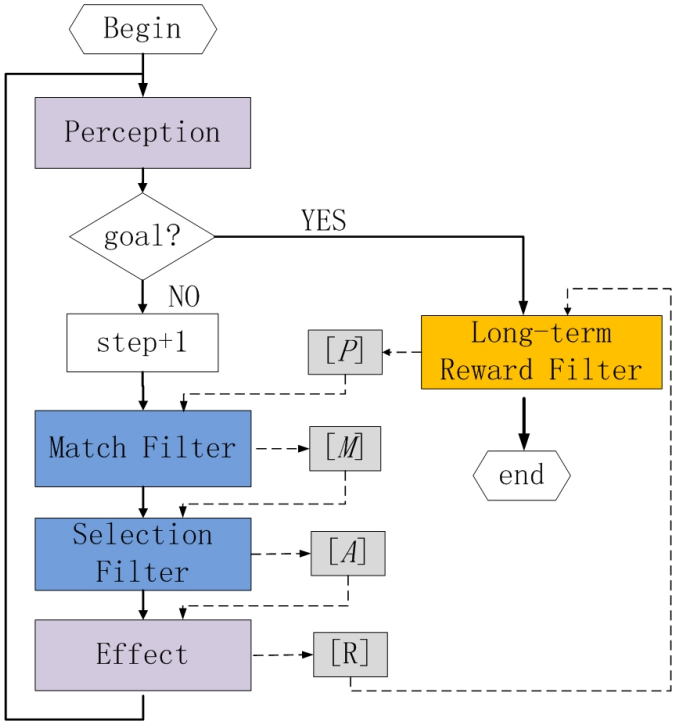
where j,$k \in \mathbb{N}$, *cl* is a classifier, a_j is the action of *cl*, fit is the fitness of *cl*, r_p is the predicted reward of *cl*, $worth_{a_j}$ is applied to a rule set [*M*].



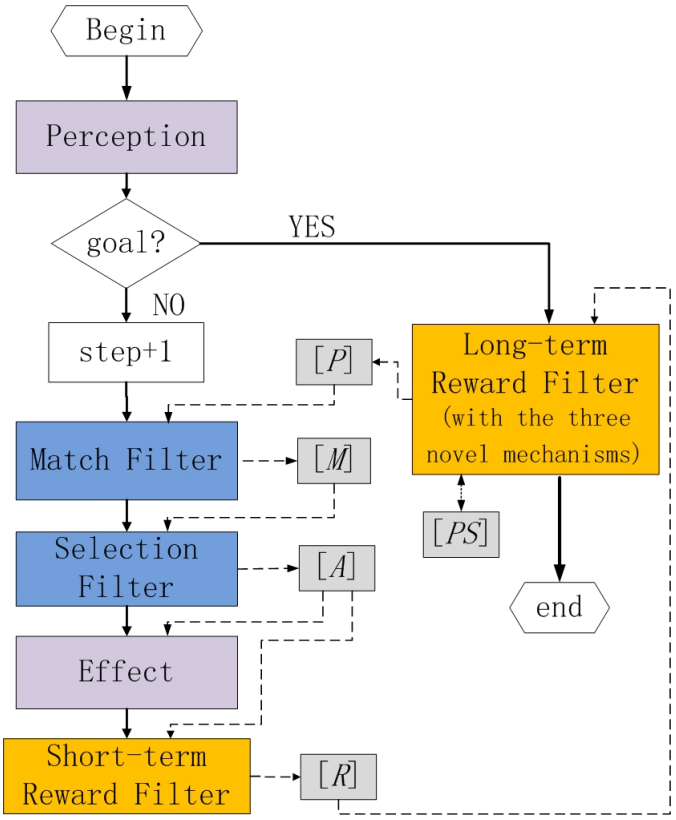
(a) Standard XCS Iteration Loop

Figure 4.3: XCS algorithms' learning iteration

Blue blocks are the standard XCS sub-processes, grey blocks are XCS rule sets, orange blocks are sub-processes of reward methods, and purple blocks indicate the agent's interactions with a maze environment. Arrows indicate the agent's working flow, and dotted arrows indicate classifiers/rules working flow.



(b) (cont.) Robotic XCS Iteration Loop



(c) (cont.) XCSCR Iteration Loop

Figure 4.3 (cont.): XCS algorithms' learning iteration

4.3.3 Reward Propagation in Multistep Problem

XCS algorithms have been applied to solve multistep problems [76], such as maze problems introduced by Wilson [69]. In maze problems, the agent has to execute actions in sequential iterations to complete a navigation task. Rewards are provided only in certain states in multistep problems. A long-term reward r_l represents the completion of the task. r_l equals an arbitrary value of 1000 if the task is completed. Otherwise, r_l equals zero in non-goal states or if the task fails (e.g. the agent has taken more steps than the step-threshold in an epoch of a trial). A short-term reward (immediate reward) r_s traditionally represents the agent's execution effects during each iteration. r_s equals an arbitrary value of -1 as a cost of the agent moving a step forward. r_s equals an arbitrary value of -50 if a collision occurs.

Traditionally, reward propagation is provided by a reward method for applying the standard XCS iteration loop in multistep problems (See Section 2.5.2). The reward method is internally responsible for assigning external rewards, including short-term rewards and any long-term reward, for each iteration. The step reward r_a is the reward which the agent assigns to recent active rules in $[A]_i$ for the step i, as in single step problems. The standard XCS reward method encapsulates long-term rewards into short-term rewards through a Q-learning like algorithm [69, 76]. The maximum potential reward available is propagated to previous actions $[A]_{i-1}$, even if that action is not taken in $[A]_i$. Although Q-learning can mathematically guarantee the approximation of estimated r_a, it might take a long time before the effect of a r_l can propagate to r_a's of early states. The propagation speed of the standard reward method might not be fast enough for robotic applications in the real-world.

Williams et al. [17, 18] proposed a reward method suited to robotic applications to solve multistep problems. The reward method encapsulates short-term effects, such as collisions and the number of steps taken, into the time factor (*time*) of long-term rewards (Equation 4.7). Instead of updating [*A*] at the end of each iteration as the standard XCS reward method does, the robotics method records [*A*] in a Reward Stack [*R*] (Figure 4.3.b). At the end of a task, the robotic reward method propagates the long-term reward r_l backward to all previously active XCS rules evenly (Equation 4.8) or with a discount factor (γ) to emphasise the contributions of recently active rules (Equation 4.9).

$$r_l = 1000 + 1/time \tag{4.7}$$

$$r_{a,i} = c * r_l/num_{step}, \tag{4.8}$$

where c is a constant, and $i \in$ step.

$$r_{a,i} = r_l * \gamma^i, \tag{4.9}$$

where $i \in$ step.

The reward encapsulation of the adapted robotic reward method [17, 18] fails to identify the rules that directly respond to short-term rewards (Section 2.5.2). The effects of short-term rewards were encapsulated as a time factor (Equation 4.7), which sums up the agent's short-term performances in the standard reward filter (Figure 4.3). Although the time factor encapsulates the effects of collisions if they happened during a trial, the time factor fails to record when these negative rewards happened. Thus, information to identify responsible rules is neglected by this method.

As the adapted robotic reward method fails to identify the rules responsible for the negative rewards, this leads to a purely random search in an agent's initial training phase. Before the establishment of effective policies, long-term, positive rewards are scarce and the agent frequently receives long-term, negative rewards. All the long-term rewards are indiscriminately propagated to rules as the reward method suggests (Equations 4.8 and 4.9). The overwhelming majority of negative rewards could be unintentionally propagated to rules that could otherwise lead to positive rewards. Therefore, all rules could be estimated as equally bad (Figure 4.4). In this case, the agent will essentially take purely random actions because explorations rely on the difference between the worth of rules.

4.3.4 XCSCR Method Overview

This work proposes XCSCR, an extension of XCS with a combined reward (CR) method, to guide the search for global optimal policies in multistep problems. CR discerns the impact of various long-term and short-term rewards to address different challenges arising from the scarcity of long-term, positive rewards. Its combined reward method

includes a short-term reward mechanism and a modified long-term reward mechanism. The short-term reward mechanism aims to encourage exploration at the learning agent's initial training phase, when the agent faces a vast policy-searching space. The long-term reward mechanism aims to exploit the long-term, positive reward effectively. The long-term reward mechanism is developed from the standard reward method with three novel reward mechanisms: an imprinting mechanism, a learning-rate switching mechanism, and a learning step-threshold mechanism. The imprinting mechanism aims to balance the negative effects of an agent's explorations on current best policies in multistep problems. The learning-rate switching mechanism differentiates long-term rewards between positive ones and negative ones. The learning step-threshold mechanism aims to create optimisation pressure to drive local optimal policies toward global optimal policies.

4.3.5 Short-term Reward Mechanism

Short-term rewards are defined as immediate rewards that the agent receives in each step. Traditionally, in a Markov model, short-term rewards are arbitrary negative values that demonstrate the immediate effects of an agent's recent step [108]. If the agent (i.e. a robot) runs to an obstacle-free cell, a small negative value, such as -1, will be assigned to this step as its cost. If a step leads to a collision, another negative value with a larger magnitude, such as -50, can be the short-term reward as a severe punishment to the step. It is hypothesised that a training process would drive local optimal policies toward global optimal policies by taking short-term rewards into consideration in an optimisation function.

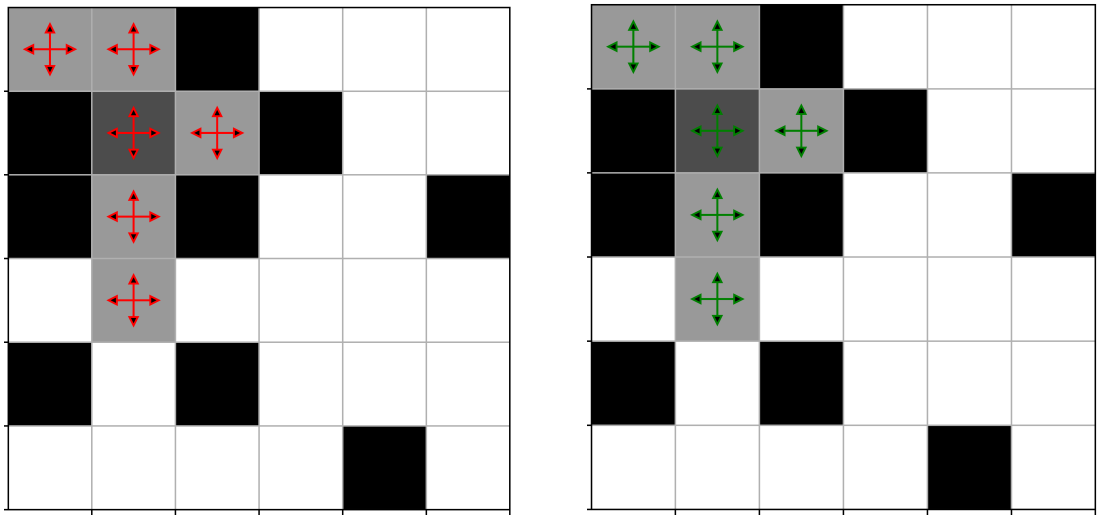
[h] **Algorithm 6** Short-term Reward Filter

Inputs: Short-term Reward r_s, Action Set $[A]$, Population$[P]$.

Outputs: Action Set $[A]$.

1: **function** UPDATE SHORT-TERM REWARD(r_s, $[A]$)
2: **if** $r_s <= \theta_{r_s}$ **then**
3: $a_j \leftarrow [A]$ //get the last executed action a_j
4: $worth_{a_j} \leftarrow (a_j, [P])$
5: $worth_{a_j} = worth_{a_j} + r_s$
6: Call Function: reward procedure ($[A]$, $worth_{a_j}$)

7: **end if**
8: **return** $[A]$
9: **end function**



(a) All the policies have the same worth.

(b) No optimal action/policy learnt.

Figure 4.4: Worth of rules shown by actions.
When the long-term, positive rewards are scarce, all actions in the initial steps are equally bad. Red arrows: the worth (fitness) of rules, magnitude suggests the negative value of the worth. Green arrows: optimal actions/policies, magnitudes are fixed. Light grey cells suggest they have been visited by the agent in this epoch of a trial. The dark grey cell suggests the location where the agent ended up at the end of the epoch.

Based on this hypothesis, this work proposes a short-term reward mechanism that attributes any severe negative short-term rewards only to the latest activated rules. The short-term reward mechanism is implemented by a short-term reward filter, which is inserted at the end of an XCS iteration loop (Figure 4.3.c). The short-term reward filter aims to immediately respond to severe negative short-term rewards, such as severe collisions, at the current step/state (see Algorithm 6). In the filter, the worth of the responsible rules in an $[A]$ will decrease by adding a short-term reward r_s, a negative reward in this case. Then, the calculated worth will be applied to update $[A]$ through the XCS standard rules' updating procedure (Equation 4.6 for calculating an action's worth

from a chosen rule set). The updated [*A*] is recorded in [*R*] for the long-term reward mechanism at the end of the iteration loop. In addition, minor negative short-term rewards, such as a step's cost, could be ignored through setting the parameter θ_{r_s} for saving on the computing budget.

The short-term reward mechanism bootstraps the policy search when long-term positive rewards are scarce in the initial training phase. The mechanism can differentiate potential useful rules from useless ones because the useless rules are responsible for severe negative effects and their worth decreases. If the long-term positive rewards are scarce, this differentiation can encourage the agent to explore the targeted states appropriately (see Section 5.6.2 and Figure 5.11.a).

4.3.6 Imprinting Mechanism

An exploration and an exploitation method are common optimisation components for machine learning algorithms, including XCS algorithms. In XCS algorithms, the exploration and the exploitation methods are applied to select an action in the selection filter (Figure 4.3). In exploitation, the filter will select the action with the best action worth, $worth_{a_j}$ (Equation 4.6). The filter will also activate exploration with a probability parameter p_{explr} to choose an action purely randomly. Previous studies show that a sensitive balance between the exploration and the exploitation is critical for XCS to learn optimal policies in single-step problems [105, 109].

Algorithm 7 Imprinting Procedure

Inputs: Current Policy p, Policy Set [*PS*], long-term reward r_l, Population Set [*P*].
Outputs: Policy Set [*PS*], Population Set [*P*].

1: **function** IMPRINTING PROCEDURE(p, [*PS*], r_l)
2: **if** r_l == 1000 **then** // p achieved the goal position
3: Call Function: update [*PS*] procedure (p, [*PS*])
//(see Algorithm 8)
4: **return** [*PS*]
5: **else if** $(RandomNumber[0, 1) > \theta_x$ and [*PS*] is not empty **then**
6: randomly select a Policy p_i from [*PS*]
7: Call Function: behavioural cloning procedure (p_i, [*P*])
//(see Algorithm 9)

```
        return [P]
    end if
end function
```

However, the exploration and the exploitation can hardly achieve the sensitive balance that drives XCS to learn optimal policies efficiently in multistep problems. This is because the standard reward-assignment mechanisms for multistep problems do not differentiate explored steps and exploited steps when the mechanism propagates rewards [76, 108]. The Q-learning reward-assignment mechanism, the even reward-assignment mechanism (Equation 4.8) and the discount reward-assignment mechanism (Equation 4.9) propagate rewards based on the sequence of steps (see Section 4.3.3). Thus, effects of exploration and exploitation activated in different steps can interfere with each other. For example, an iteration could fail because of an explored action, which is activated by the probability of p_{explr}, in a step. Without an additional algorithm to differentiate steps between exploration and exploitation, rules would be punished with the explored rules for the failure. Since the exploration is activated with the probability of p_{explr}, the best rules will be affected. Therefore, global optimal policies, which consist of the best rules, are hard to achieve or make stable in a multistep XCS agent.

Algorithm 8 Updating $[P]$ Procedure

Inputs: Current Policy p, Policy Set $[P]$.

Outputs: Policy Set $[P]$.

```
function UPDATING [P] PROCEDURE(p, [P])
    if [P] is empty then
        initialise [P] with p
    else if p.length == p_i.length (p_i ∈ [P]) and p ∉ [P] then
        cover p by [P]
    else if p.length < p_i.length (p_i ∈ [P]) then
        [P] ← empty
        cover p by [P]
    end if
    return [P]
end function
```

Algorithm 9 Behavioural Cloning Procedure

Inputs: Selected Policy p_i, Population Set $[P]$.
Outputs: Population Set $[P]$.

```
1: function BEHAVIOURAL CLONING PROCEDURE(p_i, [P])
2:     Behavioural Cloning Set [BC] ← empty
3:     for each state-action pair in p do
4:         [BC] ← ([P] ∩ state-action pair)
5:     end for
6:     Call Function: reward procedure ([BC])
7:     return [P]
8: end function
```

An imprinting mechanism seeks to restore the exploration and exploitation balance by emphasising exploitation of the current optimal policies. Similar to ideas in deep RL's behavioural cloning, the imprinting mechanism increases the worth of rules that applied in the current best policies. This will be achieved by inserting an imprinting procedure to the long-term reward filter (Figure 4.3.c). The imprinting procedure manages a Policy Set [*PS*], for exploitation reinforcements (see Algorithm 7). When a policy emerges during iterations, the imprinting procedure will update the [*PS*] through an updating PS procedure to keep the current best policies (see Algorithm 8). When the agent fails in an iteration because of explorations, the imprinting procedure will activate [*PS*] with a probability θ_x (e.g. θ_x = 0.5) to emphasise exploitation. The emphasising process will randomly choose a current best policy from [*PS*] to undergo the behavioural cloning procedure. A behavioural cloning procedure allows rules, which led to the chosen policy in [*P*], to increase their worth as if they had led to a current successful iteration (see Algorithm 9). Through these procedures, the imprinting procedure allows [*P*] to learn from "expert policies". The worth of rules that have been interfered with by the exploration method can recover their worth in an implicit way. In future work, both the updating PS procedure and the behavioural cloning procedure can employ individual XCS agents for their learning and evolving. XCS agents in these two procedures can co-evolve with an XCSCR agent at different hierarchies.

4.3.7 Learning-rate Switching Mechanism

The learning-rate can significantly impact an agent's training progress. In equation 4.1, a learning-rate β specifies the predicted reward's updating process. If β is set to a large value, the predicted reward r_p will rely on the current r_a more than the historic estimation. Otherwise, if β is set to a small value, the predicted reward r_p will have a tiny update toward the current r_a. In the classical XCS algorithm, the learning-rate is well-tuned and pre-set as a fixed parameter.

The learning-rate switching mechanism proposes to compensate for the scarcity of long-term, positive rewards by switching the learning-rate to appropriate values in training iterations. To encourage XCS rules learning aggressively from scarce, positive results, the β in Equation 4.1 is set to a large value, such as 0.2, when the agent receives long-term, positive rewards. Negative rewards are more frequent than the positive ones in a training process. When the agent receives a punishment, the β is switched to a small value, e.g. 0.05, to learn from punishments slowly. The reason for this discretization is to avoid an optimal rule in a step of an iteration being overwhelmed by "wrong rules" applied in the same iteration. As a result, the learning-rate switching mechanism emphasises the role of scarce long-term, positive rewards.

4.3.8 Learning Step-threshold Mechanism

Step-threshold was introduced into multistep learning agent as a threshold for computing resources. Step-threshold is a parameter which specifies the maximum number of steps allowed for the agent to search in an iteration. The agent needs to terminate a training iteration before a memory crash if the agent has taken too many steps in a vast space. Once the agent takes a number of steps over the value of step-threshold, the agent terminates this trial with a failure to achieve the target result. Traditionally, the step-threshold is pre-set according to the environment.

The learning step-threshold mechanism applies this threshold to generate pressure for better policies. The step-threshold θ_{step} is set as a parameter that is learnt from discovered optimal policies. Following Bellman's function, the step-threshold is updated whenever it is larger than the number of steps $step_a$ taken in the currently applied policies (Equation 4.10). When the step-threshold approaches the number of steps taken in the global optimal policies, it will create an optimal pressure driving local optimal policies

toward global optimal ones.

$$\theta_{step} = \theta_{step} + \beta * (step_a - \theta_{step}) \tag{4.10}$$

4.3.9 XCSCR Method Summary

This section proposed the XCSCR method that can bootstrap the emergence of global optimal policies in early learning iterations. The combined mechanisms of XCSCR discerned different impacts of long-term and short-term rewards. Experiments are conducted in three standard maze environments to test the performances of the XCS algorithms on their search global optimal policies (see Section 5.6). The combined reward method of XCSCR allows ACMCA to assign the final credit to co-evolving learning agents that were previously activated in a life-long learning scenario.

4.4 Chapter Summary

This chapter described two novel XCS algorithms for robotic applications of ACMCA: the *mitosis* approach and the *XCSCR* method. By introducing a novel accuracy pressure into the algorithm, the mitosis approach allows ACMCA learning agents to learn accurate knowledge even in noisy scenarios. Extending the mitosis approach from single-step scenarios to multistep scenarios, XCSCR should allow learning agents to co-evolve in multistep scenarios. Experiments that were conducted to illustrate ACMCA and these XCS algorithms are discussed in chapter 5.

Chapter 5

Results

5.1 Introduction

The previous two chapters introduced the framework of Affective Computing Multilayer Cognitive Architecture (ACMCA), which attempts to learn affective solutions for robotic applications. The framework, which includes the nodes and layers of ACMCA and its underlying machine learning algorithms, is described in these two chapters. Under this framework, the affective solutions are composed of different solution components that are distributed among the five-layer-and-three-level hierarchy. High-level solution components (e.g. secondary reinforcers and the emotion model) can be constructed through subsumptions of low-level ones. Through subsumptions, each constructed solution component provides a contingency-reasoning for a robot's responses by considering different environmental aspects. This contingency is represented by the Stimuli-Action-Outcome Contingency (SAOC) rules, which encapsulate environmental aspects, the robot's responses and their sequential effects. The contingency-based subsumptions are automatically processed by the machine learning agents of AMCMA nodes as the robot interacts with the environment. As various SAOCs are established by ACMCA, this SAOC-based subsumption system can provide affective solutions for robotic applications.

This chapter describes the implementation of ACMCA, which aims to learn affective solutions for robotic navigation scenarios. The implementation is addressed through five experiments: *Reflex-learning*, *IR-tuning*, *Deliberation-establishing*, *Emotion model*, and *Combined reward assignment*. Each experiment is engaged with different nodes

and layers of ACMCA to achieve a different component of the affective solutions. As described in the methodology chapter, the first three experiments aim to achieve three affective patterns of various behavioural levels from different navigation scenarios; the fourth experiment aims to learn an emotion pattern that coordinates the learned affective patterns; and the fifth experiment aims to extend the implementation from single-step scenarios to multistep scenarios (i.e. a robot's life-long learning scenarios).

The five experiments in this chapter unfold as follows:

(1) *Reflex-learning*: Section 5.2 conducts the experiment regarding the secondary reinforcer of *Reflex* node, which aims to produce reflex-patterns for reflex-like responses based on "pain-causing" stimuli (i.e. collisions). Following the methodology detailed in Section 3.3.2.1, this section tests the Reflex's performance of learning reflex-patterns in a collision scenario. The reflex-patterns, which effect a robot's instant behaviours, seek to allow the robot to respond to the unpredicted "pain-causing" collision to prevent the further repetition of it.

(2) *IR-tuning*: Section 5.3 conducts the experiment for the secondary reinforcer of *Tuning* node, which aims to produce IR-patterns for the adaptation of the path-planning model. Following the methodology detailed in Section 3.3.2.2, this section tests the Tuning node's performance in learning IR-patterns in real-world and simulated scenarios. The IR-patterns, which effect a robot's underlying path-planning model, attempt to achieve an automatic tuning of the model to complete navigation tasks.

(3) *Deliberation-establishing*: Section 5.4 conducts the experiment for the secondary reinforcer of *Deliberation* node, which aims to establish frustration patterns. Following the methodology detailed in Section 3.3.2.3, this section tests the Deliberation node's performance in learning *frustration-patterns* in a dynamic real-world scenario. These patterns allow a robot to reschedule its plans for a task when its original expectation of this task is failing to be achieved.

(4) *Emotion model*: Section 5.5 details the experiment for the node of *core affect state*, which aims to achieve an emotion model that responds to emotion-eliciting events. Following the methodology detailed in Section 3.3.3, this section tests this node's performance in learning its emotion model. The emotion model is

a high-level, abstract knowledge, which is achieved through the subsumptions of low-level knowledge. This emotion model allows a robot to respond appropriately to events that occur in a navigation task.

(5) *Combined reward assignment*: Section 5.6 conducts the experiment for *combined reward assignment* to explores ACMCA's potential implementations of the life-long learning scheme. Following the adapted XCSCR algorithm proposed in 4.3, the combined reward assignment will allow a robot to assign the final credit to previous, contributed policies. The method will allow a robot to co-evolve multiple learning agents together (yet not necessarily simultaneously) in multistep scenarios.

5.2 Experiment One: Reflex-learning

A robot's immediate response to hazardous environmental factors is necessary for the robot to avoid severe damage caused by these unpredicted factors. An immediate response of a robot is associated with a specified, low-level command, such as an emergency-stop instruction. Traditionally, engineers might preset certain rules regarding immediate responses under safety concerns. For example, a mobile robot might have a preset rule that freezes the robot's movement when collisions are detected. As "well-intentioned" as these preset rules might be, they might introduce human bias into the system and limit the robot's capability. In the last example, the freezing rule can be redundant at best for a rescue robot that aims to search for survivors in hazardous environments (i.e. a collapsed mine).

Instead of hard-coded responses, this section tests an approach that will allow the robot to learn appropriate unconditioned responses to environmental factors. The *Reflex-learning* node, a secondary reinforcer of ACMCA (Section 3.3.2.1), seeks to automatically learn an appropriate response to the factors, such as collisions with obstacles. As discussed in the methodology chapter, the Reflex-learning node aims to learn Stimuli-Action-Outcome Contingency (SAOC) rules that establish unconditioned responses of "reflex action" to the unconditioned stimulus from collisions. After multiple iterations, accurate SAOC rules of the Reflex-learning node will emerge, allowing the robot to avoid the repeat of impulse collisions. These accurate rules are termed as *reflex-patterns*.

The organisation of this section is addressed as follows: Subsection 5.2.1 starts with description about the scenario set-up of the experiment. The probability of collisions in the scenario is designed to be relatively higher than normal navigation scenarios to trigger a reflex-learning process of the Reflex node, which encapsulates a set of SAOC rules regarding collisions. The SAOC rules subsume solution components of the Touching node, the Reflex Velocity node and the Pain node to measure and reason the contingency between these solution components. As the robot responds appropriately to the collisions, accurate SAOC rules can emerge in this reflex-learning process. The next subsection 5.2.2 presents the result of the reflex-learning process. The SAOC rules in the Reflex node approach accuracy as the robot interacts with the environment. Accurate SAOC rules, termed *reflex-patterns*, emerge in this reflex-learning process. These reflex-patterns are visualised for interpretation. Finally, discussion 5.2.3 summarises the function of the Reflex node and the Reflex-patterns that were learnt by this node.

5.2.1 Scenario Set-up

The experiment is conducted within a representative indoor environment, i.e. a corner of an office, for the Pioneer to learn the *reflex-patterns* (Figure 5.1, and see details of the Pioneer in Section 2.7). The environment provides a restricted area, where random movements of the Pioneer are expected to frequently [1] cause diverse collisions. The Reflex node, a secondary reinforcer of ACMCA, is deployed to learn the reflex-patterns through iterations. When a collision is detected by the Pioneer's bumpers as a primary stimulus, it activates the Reflex node and triggers a learning iteration. In an iteration, the reflex-learning agent, which is underlying the activated Reflex-learning node, advocates a reflex-action to respond to this stimulus (see Section 3.3.2.1). The reflex action is then executed as an immediate response of the Pioneer to the collision, aiming to avoid continuous collisions. The feedback from the execution of the reflex action from the environment is then applied to estimate the fitness of the response, finishing an iteration. After multiple iterations, the Reflex-learning node will learn how to respond to collisions by selecting the highest fitness responses.

The reflex-patterns are encapsulated in a SAOC rule set that the reflex-learning agent

[1] That is, there are approximately 3 collisions per minute in the crowded area, which is shown in Figure5.1.b.

(a) a corridor

The corridor is an empty area, where few collisions happen to the robot.

(b) a corner

The corner is an crowded area, where collisions are events that frequently happen to the robot.

Figure 5.1: Two real world environments. The environments are applied for the robot to learn reflex-patterns and an emotion model (see section 5.5).

aims to establish. As described in 3.3.2.1, reflex-patterns are fit SAOC rules that contain three major parts: Stimuli part, the Action part and the Outcome part. The SAOC rules that interact with the environment during a trial of the task are illustrated as follows:

(1) Stimuli part. The representation of the bumpers' readings is encapsulated in the Stimuli part of the SAOC rules. The Stimuli part contains 10 attributes, representing the 10 bumpers' reading from the Touch node (see Section 3.3.1.1). Each attribute is an encoding of the values of the lower and upper boundaries that cover the bumper's reading from the environment. In this case, the value for the attribute records the corresponding reading of the bumper, and the lower and upper boundaries are generated by the algorithm to cover this value [2]. When a collision is detected by the bumper, the value of 1 will be recorded in the value of the corresponding attribute and then covered by the lower and upper boundaries. Otherwise, the value of 0 will be applied to the corresponding attribute. As the localization of bumpers around the Pioneer is fixed, the stimuli part can easily tell where the collisions happen around the robot.

(2) Action part. The action part subsumes the responding actions from the Reflex-Velocity node. It contains the index of the Reflex-Velocity (see Table 3.2), thus it advocates the corresponding Reflex-Velocity for the robots' execution. An appropriate advocation will allow the robot to react immediately from the previous colliding state. In contrast, inappropriate advocations will introduce future collisions, which will be recorded as the feedback of the execution.

(3) Outcome part. The Outcome part contains the feedback from the environment to update the fitness of the SAOC rules applied. After the robot executes the Reflex-Velocity, which is advocated by the SAOC rules, the sequential feedback is recorded for the Outcome part. After an execution of the Action, the feedback records the detection of collisions through the Touch node. The two seconds' delay is required to observe the effect of the execution because a default protection [3] will cancel any robot's actions when collisions are detected. Ideally, when the advocated Reflex-velocity is appropriate, the robot will move away for the

[2]Table 3.2 only shows the value of the reading and ignores their lower and upper boundaries for simplicity.

[3]The default protection takes approximately 1.5 seconds, leaving 0.5 seconds for the execution.

colliding location. Otherwise, the robot will still collide in the same place, which will damage the robot and its environment.

The reflex-learning agent learns the reflex-pattern for its SAOC rule set through training iterations. When the robot operates in a narrow corner, it will collide with walls sooner or later, initiating the reflex-learning agents' learning process. Initially, the reflex-learning agent randomly selects SAOC rules to execute and updates the fitness of the selected rules from their resulting feedbacks. After sufficient iterations of the SAOC rules' updating, high fitness rules can emerge in the reflex-learning agent. Thus, these fitness rules, termed as reflex-patterns, provide empirical collision solutions that the robot has experienced.

5.2.2 Result of the Reflex-learning

The SAOC rules are evolved by the reflex agent of the Reflex-learning node (Section 3.3.2.1) in the experiment. As a result, the reflex-patterns, the fittest SAOC rules in the reflex agent, emerge as learnt solutions for reacting to collisions. The solutions demonstrate that moving away from the colliding positions is an appropriate response.

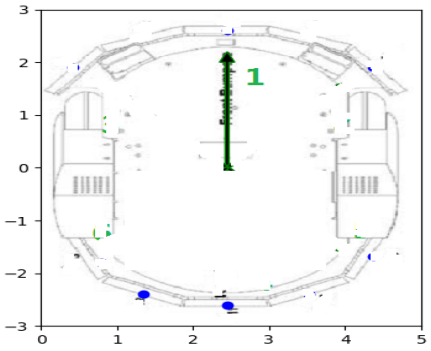
These solutions are demonstrated in Table 5.1. Each row represents a solution that is represented by an XCS macro-classifier within the reflex agent. The columns C1V to C10V indicate the readings of the bumpers. These columns also indicate the location of the collisions because the bumpers are fixedly arranged around the robot (see Figure 3.11), setting the condition part of the solution. For example, if C1V equals 1, the robot has a collision in the front. As shown in the table, collisions were detected by a single bumper or by two neighbouring bumpers simultaneously. When the condition part of the solution matches the current detections, the Reflex-action that is encapsulated in the action part of the solution is activated for the robot's execution.

The learned reflex-pattern meet the expectation of the robot's appropriate responses to collisions. That is, the reflex-patterns can prevent the robot from a further repetition of collisions. The learned reflex-patterns are shown in Table 5.1, and they are also visualised in Figure 5.2, The result shows that these patterns correctly advocate the robots' actions that drive the robot away from the position where the collisions are detected.

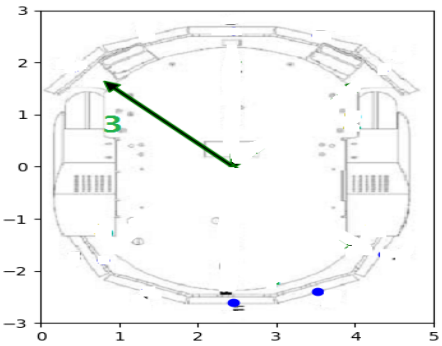
Table 5.1: Reflex-patterns

ID	C1V	C2V	C3V	C4V	C5V	C6V	C7V	C8V	C9V	C10V	ACT	PRD	ERR	ACC	FIT	EXP	AS	NUM
105	0	0	0	0	0	0	1	1	0	0	one	985.68	49.78	1	0.224	124	34.44	2
1	0	0	0	0	0	0	0	1	1	0	three	956.45	96.6	0.014	0.217	121	90.164	15
131	0	0	0	0	0	0	0	1	1	0	three	956.44	96.61	0.014	0.159	120	90.164	11
361	0	0	0	0	0	0	0	1	1	0	three	956.44	96.65	0.014	0.13	114	90.164	9
305	0	0	0	0	0	0	0	1	1	0	three	956.44	96.65	0.014	0.101	117	90.164	7
851	0	0	0	0	0	0	0	1	1	0	three	956.41	96.99	0.014	0.071	96	90.164	5
586	0	0	0	0	0	0	0	1	1	0	three	956.43	96.78	0.014	0.057	106	90.164	4
538	0	0	0	0	0	0	0	1	1	0	three	956.44	96.75	0.014	0.043	109	90.164	3
965	0	0	0	0	0	0	0	1	1	0	three	956.39	97.16	0.014	0.042	92	90.163	3
1025	0	0	0	0	0	0	0	1	1	0	three	956.38	97.27	0.014	0.028	90	90.163	2
117	0	0	0	0	0	0	0	1	0	0	three	938.13	146.54	0.004	0.24	167	58.899	11
231	0	0	0	0	0	0	0	1	0	0	three	938.13	146.54	0.004	0.152	162	58.899	7
456	0	0	0	0	0	0	0	1	0	0	three	938.13	146.54	0.004	0.044	149	58.899	2
54	0	1	0	0	0	0	0	0	0	0	four	993.81	28.97	1	0.968	132	40.679	35
11	0	0	1	1	0	0	0	0	0	0	four	990.83	39.06	1	0.81	100	39.448	19
235	0	1	1	0	0	0	0	0	0	0	five	946.74	68.36	0.039	0.172	71	48.699	12
692	0	1	1	0	0	0	0	0	0	0	five	945.76	78.14	0.026	0.098	60	48.691	9
23	0	1	1	0	0	0	0	0	0	0	five	946.84	66.89	0.042	0.156	73	48.7	8
341	0	1	1	0	0	0	0	0	0	0	five	946.66	69.11	0.038	0.091	69	48.699	7
504	0	1	1	0	0	0	0	0	0	0	five	946.49	70.42	0.036	0.107	66	48.698	6
576	0	1	1	0	0	0	0	0	0	0	five	946.35	71.46	0.034	0.105	64	48.697	6
941	0	1	1	0	0	0	0	0	0	0	five	945.36	77.41	0.027	0.054	56	48.681	5
1071	0	1	1	0	0	0	0	0	0	0	five	943.89	90.19	0.017	0.021	53	48.667	5
763	0	1	1	0	0	0	0	0	0	0	five	945.69	75.59	0.029	0.052	58	48.687	4
1160	0	1	1	0	0	0	0	0	0	0	five	944.09	85.7	0.02	0.025	51	48.654	4
1224	0	1	1	0	0	0	0	0	0	0	five	943.82	84.8	0.021	0.021	50	48.645	3

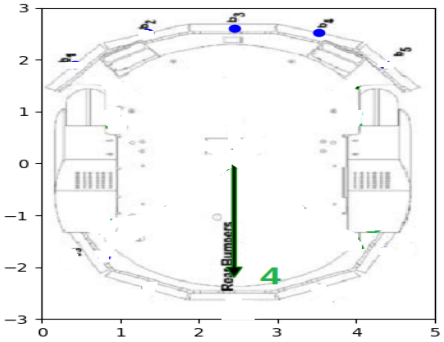
[1] The condition part of classifiers are shown by readings' values (C1V-C10V) and ignores their lower and upper boundaries for simplicity.



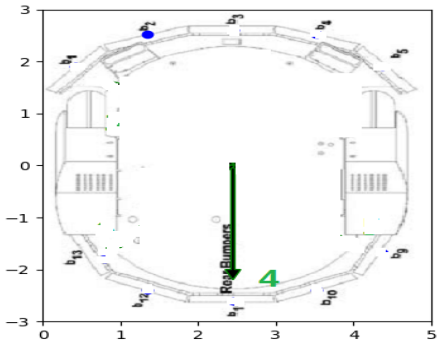
(a) Reflex-pattern NO.1



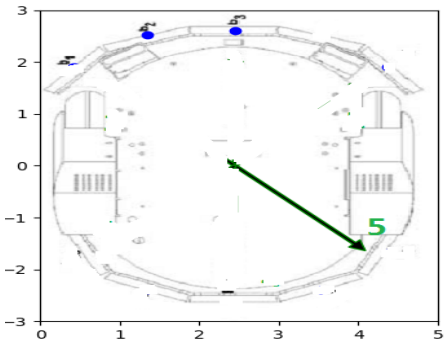
(b) Reflex-pattern NO.105



(c) Reflex-pattern NO.11



(d) Reflex-pattern NO.54



(e) Reflex-pattern NO.235

Figure 5.2: Reflex-pattern

Bumpers that detect collisions are marked with blues points. The green arrows represent the moving directions that are selected by reflex-patterns.

These reflex-patterns are high worth rules, which are shown by their statistics parameters. These statistics parameters include predicted-reward (PRD), prediction error (ERR), accuracy (ACC), fitness (FIT), experience (EXP), action set size (AS), and numerosity (NUM) (see the columns of "PRD", "ERR", "ACC", "FIT", "EXP", "AS", and "NUM" from Table 5.1, and see the definition of these parameters in [76]). The parameter of the fitness denotes a classifier's fitness, which is a normalized value of all the classifier in the same data niche. The classifiers in the table have a higher value of fitness than their counterparts in the same data niche. This suggests that these classifiers are fit reflex-patterns, which are learnt from the experiment. Their parameters of the predicted-reward and the accuracy approach their maximum scaled values. This suggests that these classifiers are accurate patterns that have a positive effect (e.g. avoid a repetition of collisions). The classifiers shown in the table are macro-classifiers, and the parameter of numerosity reflects the number of micro-classifiers in the marco-classifier. The numerosity of all the classifiers in the table is larger than two, suggesting the system has converged on these classifiers.

Notice that there are equivalent patterns in the SAOC rule set. Equivalent patterns are accurate rules that have the same condition, the same action part, and similar statistics. For example, these patterns are equivalent: NO.1, NO.131, NO.361, NO.305, etc. In terms of algorithm, a potential subsumption operation of the equivalent patterns may bring a clearer view of interpretation than the current one. Similar to the GA operation of XCSAM [110] be conducted on the best action set, this subsumption operation can be conducted on the evolved patterns. But the subsumption operation may have little impact on the robot's performance. This is because the effort of subsumed patterns has the equivalent effect on the decision-making process of the reflex-agent to all the equivalent pattern together. This subsumption operation also requires additional memory and computing resources, which are generally constrained by the robotic application. Therefore, this work chooses not to implement this subsumption operation.

In addition, the pattern with the action two and the action six did not emerge as other patterns matured. The explanation is the experiment might not generate enough scenarios for the reflex-agent to learn these patterns. A pattern with the action two requires the robot to move in the direction of the action six, and vice versa. As the patterns with action one, three, four, five matured, the robot's sequential movements in the directions of action one, three, four, and five become overwhelming. Therefore,

scenarios, where the robot moves in the other two directions, were inhibited by these overwhelming actions.

5.2.3 Summary

This section introduced the experiment for learning the reflex-patterns through the Reflex-learning node in a real-world environment. Instead of presetting the linkings between collisions and responses, the Reflex-learning node learns these linkings as the reflex-patterns. The learned reflex-patterns allow a robot to respond appropriately to the unpredicted "pain-causing" stimuli instantly from continuous damage. These reflex-patterns are one of many low-level solution components that can be achieved by secondary reinforcers. These solution components can be applied to automatically construct high-level solution components (i.e an emotion model) by a high-level ACMCA node. Therefore, a completed solution, which includes both low-level and high-level solution components, can be generated by ACMCA thought its evolutionary subsumption processes. That is, ACMCA can evolve its architecture to learn the complete solution for complex tasks.

5.3 Experiment Two: IR-tuning

Fine-tuned parameters can be critical for a model to function appropriately for its robotic application. A hyperparameter of a method is a parameter that has determining effects on the method's performance from a higher perspective (e.g. the number of inputs available to a control system). Traditionally, it is the engineer's responsibility to tune hyperparameters scenario by scenario to ensure the model functions well. Thus, these fine-tuned hyperparameters are preset for a scenario as prior knowledge. However, this manually tuning approach increases the potential to introduce human bias into the system, and the job of tuning could become a tedious burden for engineers who have to make the model adaptive to the targeted scenario.

Instead of manually tuning a hyperparameter, this section proposes an automatically-tuning-hyperparameter approach for navigation models. The Tuning node, a secondary reinforcer in the architecture (see Section 3.3.2.2), can automatically learn appropriate values of *Inflation Radius* (*IR*) (see Section 3.3.5.3), a hyperparameter of the path-

planning model, for various scenarios. Actually, whether a path can be generated by the path-planning model is a contingency that an IR setting can determine when the robot operates in crowded areas. Learning this IR contingency from an environment can lead to the adaptation of the path-planning model to this environment without presetting or manually tuning the hyperparameter.

The Tuning node is proposed to learn this IR contingency for the path-planning model. Following the proposed methodology (Section 3.3.2.2), this section conducts the experiment of the Tuning node in various scenarios. The Tuning node learns the SAOC rules that encapsulate the IR contingency through the proposed XCS algorithms, which includes the mitosis approach of the XCS algorithm (Section 4.2). As the robot interacts with the environment, the most fit SAOC rules, which encapsulate IR-patterns, emerges. These IR-patterns and their encapsulated contingencies can be visualised within the environmental map for interpretation and analyses.

The proposed mitosis approach of the XCS algorithm aims to discover accurate IR patterns from the experiments. Compared to the standard XCS algorithm, the mitosis approach introduces an accuracy pressure to balance the overgeneralized pressure, which is rooted in the standard algorithm. The hypothesis is that the XCS algorithm with balanced accuracy pressures can evolve more accurate rules than the one dominated by the overgeneralized pressure. The proposed algorithm compares with the benchmarked algorithms in various navigation scenarios. The comparison includes quantitative and qualitative analyses of learned IR-patterns of both algorithms. These analyses also indicate that accurate learned IR-patterns can be learnt by the Tuning node with the novel algorithm, instead of hand-coded hyperparameters for targeted scenarios as prior knowledge.

The following section starts with the scenario set-up of the experiment. Different navigation scenarios are included in the experiences, and a description of the IR contingency encapsulated in the Tuning node is detailed. In the next part, the results of these experiments are illustrated with quantitative and qualitative analyses. The learned fitness SAOC rules, termed IR patterns, are visualised and compared to the navigation environment. A summary of the Tuning node and the mitosis approach is provided at the end of the section.

5.3.1 Scenario Set-up

Experiments were conducted in both simulated and real-world environments. The simulated environments were in the Willow Garage Offices (see Figure 3.21.a), which is implemented using a *Robotic Operation System* backend engine, the *Gazebo* simulator [17]. A mobile robot is to navigate along a targeted path (see Figure 3.21.a), which is separated into 31 segments. Each segment was contained in a 3m by 3m square area, also termed *IR-tuning window*. Although it is not a necessary requirement, each IR-tuning window was the same size as the local perception map in the *ROS* path-planning module to facilitate the analysis of the results. An IR-tuning window is the navigation environment of a scenario, which includes a navigation environment, a navigation task, and the robot's admissible behaviours to fulfil the task in the environment. Thus, the segmentations ensured that the 31 scenarios contain various territories (see Figure 3.21), such as wide-open areas, doorway areas, long narrow corridors and irregular territories.

31 navigation scenarios are included in the experiences for three reasons. Firstly, 31 different scenarios can generate 31 groups of results, which are sufficient to conduct a two-tailed statistical significance test. Secondly, 31 navigation scenarios can test the scalability of the Tuning node and the underlying algorithm because these scenarios contain various territories. Thirdly, 31 navigation scenarios facilitate the experiment. When a robot finishes the experiment in one scenario, it can automatically move to the next scenario along the circle path.

The Tuning node aims to find appropriate values of the hyperparameter of IR for each scenario. Each scenario has its optimum IR value and the same IR value for all scenarios are non-optimum. A large IR value can lead to valid paths being block in a narrow area, and a small IR value can lead to unnecessary collisions occurring.

The real-world environment was an office of the Alan MacDiarmid building of the VUW university (see Figure 5.3). As shown in the figure, a filing cabinet was set in the middle of the open area to create narrow corridors in the environment. The Tuning node had the same objective as in the simulated environments above. Although the Tuning node is capable of operating in the physical world, repetition for statistical analysis is too time-consuming, so the major tasks were conducted in the simulation environment to verify its functionality and scalability. One task was in the real-world experiment to show its practical effects.

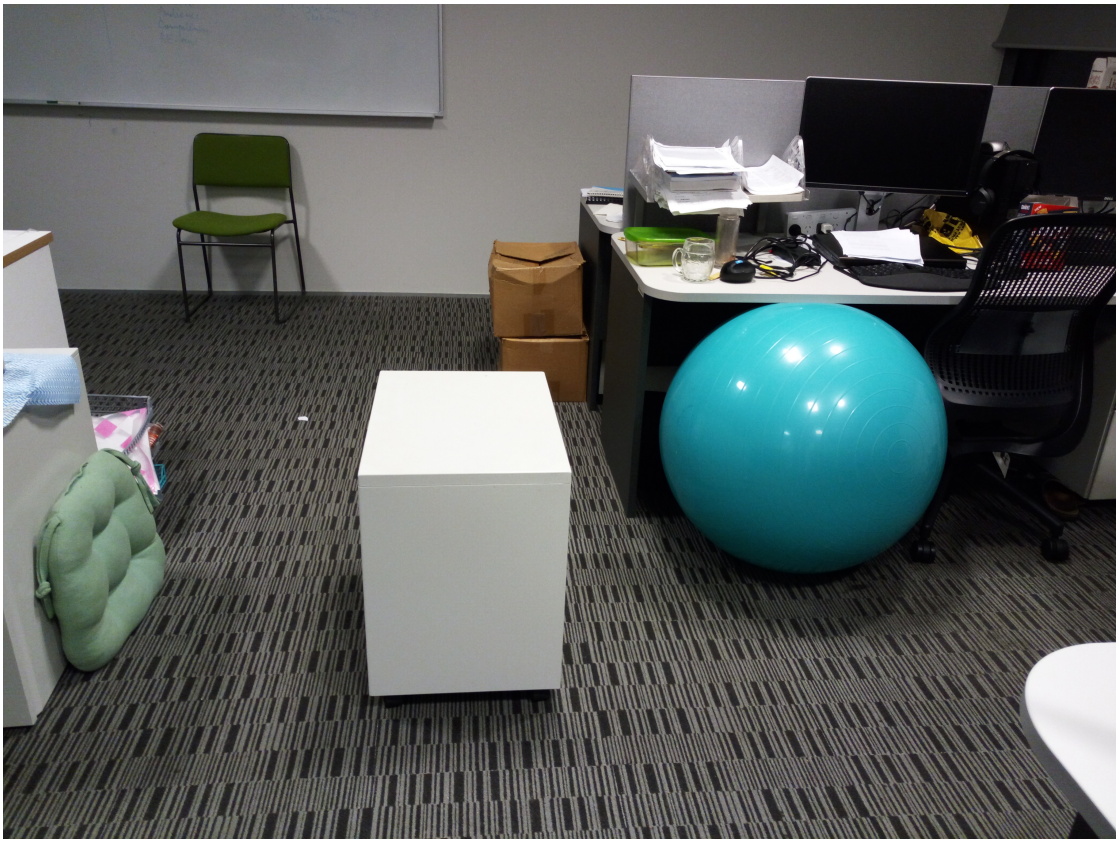


Figure 5.3: A filing cabinet blocks the path in the office.
A robot should learn appropriate IR value to pass through the left side of the cabinet without collisions. This cabinet is simulated as a dynamic obstacle (Section 3.3.1.5) by our sim2real approach.

This approach is similar to sim2real [111, 112, 113], which avoid fine-tuning in the real world, yet train wholly in simulation, and perform a zero-shot transfer to the real world. The knowledge that is learned by the Tuning node in the simulations is transferable to a real-world robot.

The mobile robot in the simulated environment and the real-world one is *Pioneer* [17], which is equipped with a *LIDAR*, front and rear bump sensors, and ultrasonic sensors. The common *ROS* navigation stack [18] was applied as its the path-planning module. The navigation task of a scenario is to require the path-planning module to generate a path to a goal position. As the Pioneer interacts with the environments, the Tuning node will learn solutions about the path-planning module's admissible behaviours.

The path-planning module's admissible behaviours largely rely on hyperparameters, as described in the beginning of this Section. These hyperparameters include *Inflation Radius* (*IR*), *Cost Factor*, *Publishing Frequencies* and so on [90]. Among these hyperparameters, IR has a dominant effect on admissible behaviours of the path-planning method. The IR specifies an obstacle-spreading distance from obstacles [90], and this hyperparameter is critical to results of the path-planning method: whether the path-planning method can generate a path within the current scenario or not. This is because different values of the IR will result in different perceived environments of the same scenario, especially in narrow spaces (see Figure 3.21.b, 3.21.c and 3.21.d). An over-large value of IR (e.g. 0.8 m) will fill space around obstacles with large inflation (see Figure 3.21.c). Hence, narrow spaces, where the path-planning method should be able to generate a path through, will be considered as being occupied by an obstacle as well. Doorway areas in the scenario NO.4 and NO.24 (see Figure 5.5.b and 5.5.h) are such narrow spaces where the path-planning method could fail to generate a valid path through these areas (see Figure 3.21.c). In contrast, there are opposite consequences (i.e, collisions) if IR is too small in these areas.

The Tuning node aims to establish a contingency between the engaging environment, the module's admissible behaviours, and their feedback from the environment through experiments. As described in Section 3.3.2.2, the contingency is represented by the set of SAOC rules and the population of classifiers in an XCS agent of the node. Specifically for the experiments, the physical meanings of each attribute of the condition and action parts of classifiers are detailed as follows:

(1) The first two attributes of the condition respond to the targeted position of the navigation task through the Goal node (see the Goal node in Section 3.3.1.2). Because a targeted position is kept in an orthogonal coordinate system (an x-axis-and-y-axis plane), the two attributes are needed. Classifiers that can be applied to the targeted position will cover the x-axis coordinate and the y-axis coordinate of the targeted position by their first two attributes.

(2) The third attribute of the condition is associated with the hyperparameter value of the IR of the path-planning model (see the IR node in Section 3.3.5.3). The IR value is then updated to the path-planning model as its new hyperparameter by the IR node. This updating process is completed through the dynamic reconfigure

process of ROS (see details of the dynamic reconfigure process in this link [114]).

(3) The attribute of the action part predicts whether any valid path can be generated by the model. Then, this prediction will be compared with the "ground truth" that comes from the Path node (see the Path node in Section 3.3.1.4) as the feedback of the model.

The XCS agent of the Tuning node is evolved through multiple iterations in the experiments. In each iteration, the targeted position and the IR value are inputs of the XCS agent. The targeted position is created as a random position within the IR-tuning window, and the IR value is a random value selected from a score of 0m to 0.8m [4]. When the XCS agent has access to these inputs, the agent selects a set of classifiers from the population. The selected classifiers can cover the inputs by their condition parts. Based on the selected set, the XCS agent then predicts the existence of any valid path. If the path-planning model is not going to generate any valid path to the targeted position under the IR value, the prediction is assigned by an arbitrary value of 0. Otherwise, the prediction is an arbitrary value of 1. After the prediction, the XCS agent is suspended and the Tuning node updates the IR value to the path-planning model, waiting for the path-planning result from this model. The path-planning model responds with an arbitrary value of 0, if there is not any valid path generated, and otherwise responds with an arbitrary value of 1. The response of the path-planning model resumes the XCS agent to update the classifiers in the selected set. Compared to the response, if these classifiers made a correct prediction, the *fitness* and the *prediction accuracy* of these classifiers will increase. Otherwise, the fitness will decrease, which increases the probability of deletion of the classifiers from the population. Generally, high fitness classifiers can emerge from the population as they evolve through multiple iterations. These SAOC rules, which are encapsulated by these high fitness classifiers, are termed as the IR-patterns. In the next subsection, IR-patterns are visualised for statistical analysis.

SAOC datasets are established in 31 navigation scenarios (Figure 3.21.d) for the training and iterations of the XCS agent. Each SAOC instance in the datasets records the targeted position, the IR values, and the response of the path-planning model as the label.

[4]The upper boundary is defined as 0.8 m. This is because we assume that 0.8 m is a sufficient distance to keep the robot away from an obstacle. This upper boundary can be modified according to various robotic applications.

Thus, SAOC instances are labelled data that were applied for training the XCS agent. Notice that the SAOC datasets contain noise, which is instances with wrong labels. Take noise instances in Scenario NO.28 (Compare Figure 5.5.c) for example. This scenario contains a wall occupied on the left side (Figure 5.5.b). There is no path that can lead a robot to the wall-occupied area, yet some noise instances in that area suggest otherwise (see blue points in the wall-occupied area). As discussed in Section 4.2.2, these noisy instances will prevent the standard XCS algorithm from evolving toward maximal general and accurate classifiers, because of the algorithm's overgeneralized tendency that the mitosis approach aims to amend.

5.3.2 Result of IR-tuning

The experiments show that the Tuning node was able to increase navigation adaptation of the mobile robot. The learned IR-patterns allow the path planning to generate valid paths in all 31 navigation scenarios (Figure 3.21.d and Section 5.3.2.1). These IR-patterns were visualised for ease interpretation (Section 5.3.2.2). Experiments also showed that the emotion inspired reasoning mechanism applied by XCS algorithms was capable of learning SAOC knowledge in the 31 navigation scenarios. The mitosis approach of XCS algorithm performed better than the standard approach on almost all of the 31 navigation scenarios in terms of *prediction accuracy* (Section 5.3.2.1), *pattern robustness* (Section 5.3.2.2), and *pattern accuracy* (Section 5.3.2.2).

5.3.2.1 Quantitative Analysis

Quantitative analysis is based on the *prediction accuracy* of learned SAOC rules in the Tuning node. The learned SAOC rules are applied to predict each SAOC instance's label in a dataset, then the number of correct predictions of the dataset will be recorded to calculate the prediction accuracy. The prediction accuracy is the ratio of the number of correct prediction to the number of instances in a SAOC dataset. Notice that inaccurate instances in a SAOC dataset keeps this prediction accuracy from achieving 100% accuracy. The quantitative analysis includes the standard XCS approach as the benchmark for the mitosis approach proposed.

A one-hidden-layer neural network is also included to demonstrate potential interferences from the SAOC datasets. These datasets come from various scenarios. Different

scenarios contain different numbers of inaccurate instances, thus causing a different level of interferences. The one-hidden-layer neural network provides a base-line that demonstrates the potential interferences to the XCS algorithms.

The quantitative analysis of the prediction accuracy highlighted that the mitosis approach achieved better prediction accuracy than the standard approach (see Table 5.2). The mitosis approach had statistically significant improved prediction accuracies in 23 of the 31 navigation scenarios based on two-tailed T-test with 0.05 p-value (see Table 5.2, each navigation scenario was run for 30 trails with known seeds). In the remaining eight navigation scenarios, prediction accuracies were not significantly different between the two approaches, yet the mitosis approach achieved a higher value of the mean of prediction accuracy in seven of these eight scenarios and a slightly worse performance in one scenario. As the mitosis approach had overall better two-tailed performances, this also indicates that the accuracy pressure that was introduced by this method could effectively amend the overgeneralized tendency of the standard XCS algorithm.

Table 5.2: Statistics of The Two Approaches and The Standard One-hidden-layer Neural Network

Scenario ID	1	2	3	4	5	6	7	8	9	10
Mean 1	0.836	0.909	0.902	0.823	0.818	0.875	0.777	0.798	0.826	0.814
Mean 2	0.826	0.901	0.892	0.814	0.808	0.866	0.76	0.779	0.813	0.816
Mean 3	0.8	0.851	0.84	0.732	0.721	0.787	0.644	0.706	0.641	0.63
P value	0.005	0.009	0.004	0.117	0.159	0.022	0.002	0	0.005	0.704
Difference	0.01	0.008	0.01	0.009	0.01	0.009	0.017	0.019	0.013	-0.002
Scenario ID	11	12	13	14	15	16	17	18	19	20
Mean 1	0.847	0.805	0.812	0.894	0.94	0.886	0.909	0.926	0.923	0.925
Mean 2	0.834	0.77	0.791	0.86	0.94	0.868	0.904	0.918	0.908	0.914
Mean 3	0.614	0.688	0.611	0.858	0.836	0.807	0.897	0.904	0.922	0.914
P Value	0.002	0	0.006	0	0.786	0	0.09	0.009	0	0
Difference	0.013	0.035	0.021	0.034	0	0.018	0.005	0.008	0.015	0.011
Scenario ID	21	22	23	24	25	26	27	28	29	30	31
Mean 1	0.886	0.918	0.911	0.836	0.805	0.819	0.837	0.8	0.772	0.805	0.857
Mean 2	0.882	0.912	0.9	0.818	0.785	0.805	0.824	0.782	0.762	0.797	0.838
Mean 3	0.841	0.877	0.827	0.747	0.633	0.764	0.703	0.681	0.766	0.69	0.796
P Value	0.071	0.005	0.019	0.014	0.026	0.003	0.014	0.001	0.07	0.154	0.002
Difference	0.004	0.006	0.011	0.018	0.02	0.014	0.013	0.018	0.01	0.008	0.019

[1] Mean 1: mean value of prediction accuracies of the *mitosis approach* for 30 trials.

[2] Mean 2: mean value of prediction accuracies of the *standard approach* for 30 trials.

[3] Mean 3: mean value of prediction accuracies of the standard one-hidden-layer neural network for 30 trials.

[4] P value: P-value of prediction accuracies the *mitosis* and *standard approach* by the two-tailed T-test among 30 trials.

[4] Difference = Mean1-Mean2, the difference of the prediction accuracies between the *mitosis* and *standard approach*.

5.3.2.2 *IR-Pattern* Analysis

A major advantage of the two XCS approaches over the applied neural network is that the learned knowledge of the XCS approaches is easy to interpret. The neural network learns knowledge through matrix modules, in which mappings and weights usually require extra effort to interpret. In contrast, XCS preserves knowledge in rules, where the "if-then" structure is comprehensible. In addition, because SAOC knowledge is directly encoded into XCS rules (Section 3.3.2.2 and Figure 3.16), the learned, qualified SAOC knowledge, termed *IR-Pattern*, can be conveniently harvested from a qualified classifier for visualisation.

IR-patterns were visualised as cuboids in the three-dimensional *SAOC Space* for interpretation (see Figure 5.4 and 5.5). A shape of a cast from a pattern to the x-axis-and-y-axis plane covers the targeted area within the navigation environment where these patterns apply. The length of cast from a pattern to the modifier axis indicates an inflation radius value, which is what the path-planning module applies to the targeted area. Therefore, patterns provide more generalized knowledge than specific SAOCs. Besides, the SAOC patterns are also called *Inflation Radius (IR) patterns*, because these patterns describe the learned inflation radius distribution on the x-axis-y-axis plane. Four types of IR patterns, which are generated by the *XCS* algorithm, can be categorised into four types by symbolic meanings of classifiers (Table 5.3, 5.4). The action part (*act*) predicts whether it is open spaces or obstacles. The strength of the classifier, *prd*, indicates the strength of this prediction. For example, in TP pattern, the *act* equals 1, suggesting a prediction of open spaces; the *prd* equals 1000, indicating the strength of this prediction is very strong. Therefore, the TP pattern suggests an open space. In the TN pattern, the *act* equals 0, suggesting a prediction of obstacles; the *prd* equals 0, indicating the strength of this prediction is very weak. Therefore, the TN pattern also suggests its application in an open space. Likewise, the FP and FN pattern work for space that contains obstacles.

Experiments show that the proposed cognitive architecture was capable of being applied to various territories. The 31 navigation scenarios contained various territories, such as wide-open areas, long narrow corridors, doorway areas and irregular territories. Four navigation scenarios are selected as examples for three typical territories in Figure 5.4 and Figure 5.5. Navigation scenario No.7 was for a long corridor with 45-degree

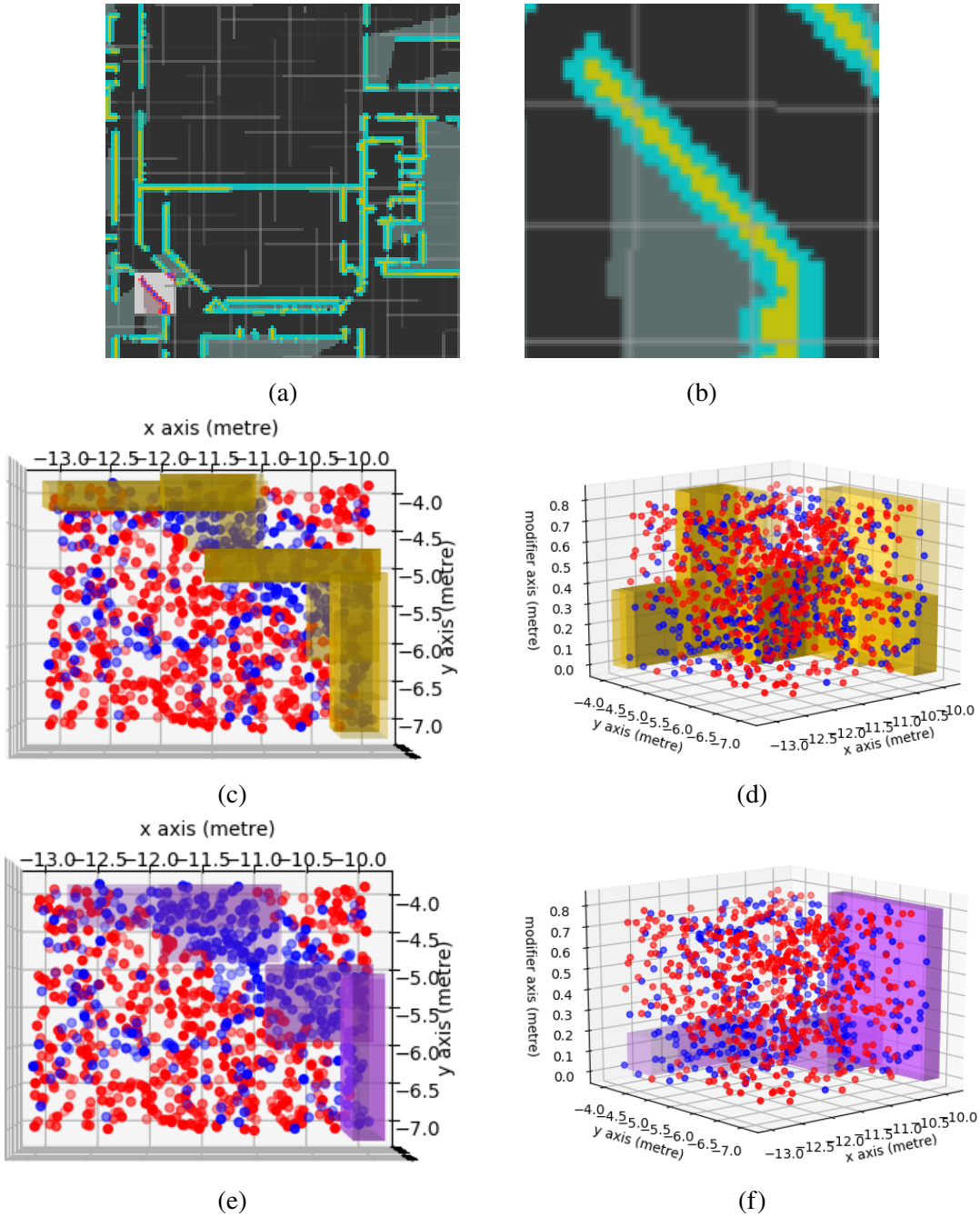


Figure 5.4: Two Types of Discovered IR Patterns - Scenario No.7, Mitosis Approach Global Perception Map: a; Local Zoom Map: b (targeted area shown with the green grid); *TP pattern*: yellow cuboids, *TN pattern*: purple cuboids; plan view: c, e; diagonal view: d, f. Red dots: instances that are labelled with obstacles; Blue dots: instances that are labelled with no obstacle.

Table 5.3: Confusion Matrix of *IR Patterns* (from the perspective of classifier's statistics)

		Predicted-reward	
		Open Spaces (*prd*=1000)	Obstacles (*prd*=0)
Classifier Prediction	Open Spaces (*act*=1)	TP Pattern (True Positive)	FP Pattern (False Positive)
	Obstacles (*act*=0)	FN Pattern (False Negative)	TN Pattern (True Negative)

Table 5.4: Confusion Matrix of *IR Patterns*

		True Occupancy	
		Open Spaces (True)	Obstacles (False)
Classifier Prediction	Open Spaces (Positive)	TP Pattern (True Positive)	FP Pattern (False Positive)
	Obstacles (Negative)	FN Pattern (False Negative)	TN Pattern (True Negative)

orientation (see Figure 5.4.a, and Figure 5.4.b.). Scenario No.28 was for an irregular territory (see Figure 5.5.b). Scenario No.4 (see Figure 5.5.h) and No.24 (see Figure 5.5.n) were for doorway areas.

Experiments show that the mitosis approach performed better than the standard approach to learn valid IR patterns in terms of *pattern robustness* and *pattern accuracy*. Firstly, correct patterns were more frequently achieved by the mitosis approach than the standard approach. The *pattern robustness* refers to the frequency of valid IR patterns, which can be harvested at the end of a learning process. An example is their performance comparison in navigation scenario No.7 (see Figure 5.4). Patterns aim to cover the corridor without a gap. The 45-degree orientation of the corridor creates a challenge for two approaches to generate accurate patterns, which will have an accurate coverage. The coverage requirement is similar to filling a trapezoidal space with cubes that can be skewed to align the axes. Thus, gaps between the cubes are difficult to avoid, and connected patterns without a gap are more accurate than patterns with gaps. Based on the performances of *TP pattern* in the total 30 trials repetition, the mitosis approach overcame this challenge 15 times, and the standard approach accomplished it 7 times.

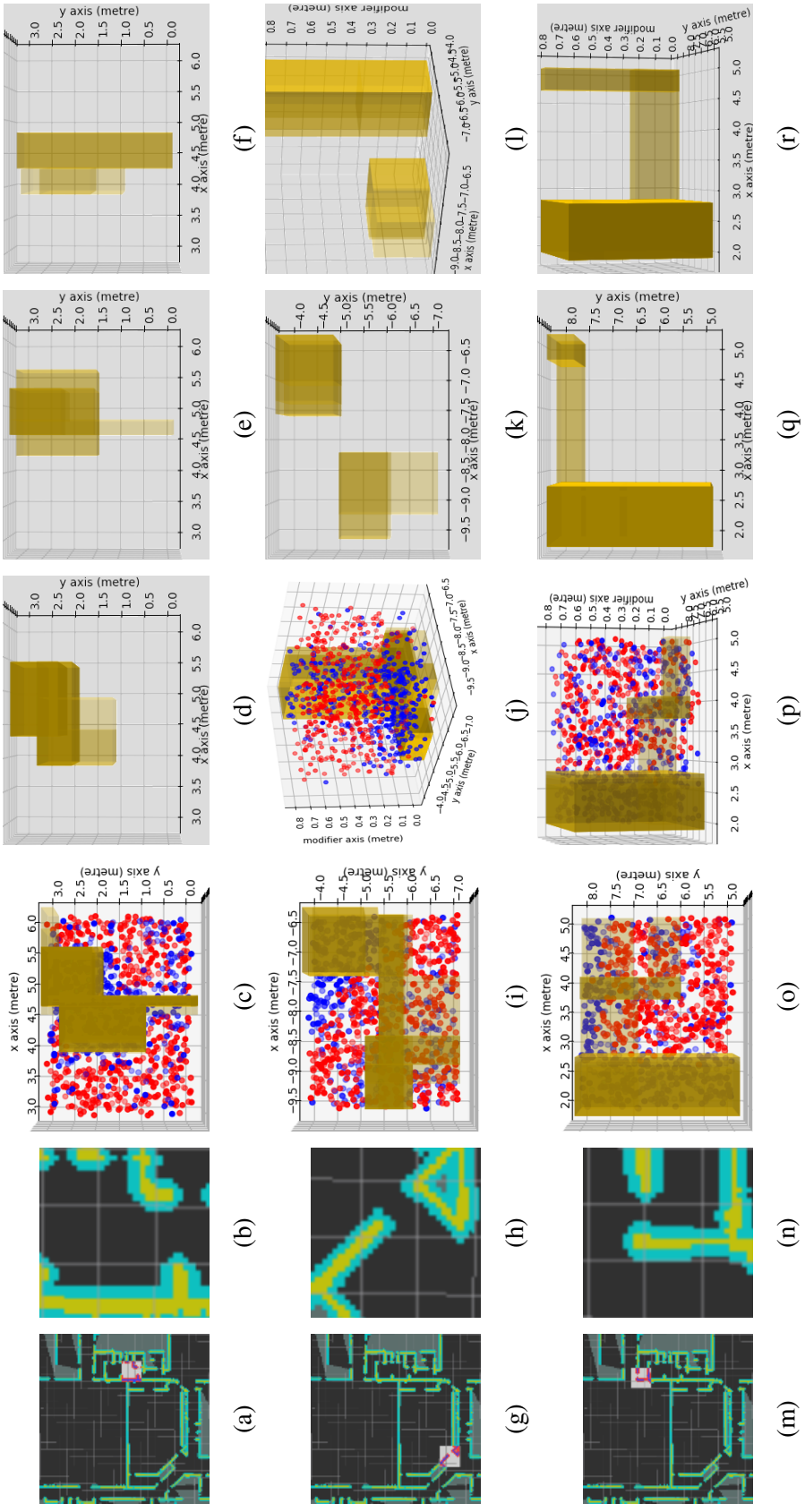


Figure 5.5: Discovered IR Patterns - Scenarios No.28, No.4 and No.24 per row respectively.

Note: unresolved gaps in paths learned by the standard approach compared with the novel approach, e.g. k vs i. Mitosis: white background; Standard: grey background. Row 1 shows single optimal path vs three local paths. Global Perception Map: Col. 1; Local Zoom Map: Col. 2; All columns plan views, except diagonal view: j, l, p, r.

Secondly, patterns that were achieved by the mitosis approach are more accurate than that of the standard approach. *Pattern accuracy* can be judged by how accurately a pattern covers the targeted area in the perception map, hence a global optimum solution is patterns that can accurately cover the targeted area. The mitosis approach was able to achieve a global optimum solution in complex territories, e.g. Scenario No.4, No.24 and No.28, where the standard approach failed (see discussions in the next two paragraphs). This suggests that the mitosis approach overcame two problems caused by the overgeneralization tendency of the standard *XCS* algorithm (see Section 4.2.2):

1 Mitosis approach provides an accuracy pressure that keeps accurate classifiers in their accurate states;

2 This accuracy pressure drives the evolutionary methods toward maximally general and accurate classifiers.

The first point was shown in irregular territories, such as Scenario No.28. In Scenario No.28, TN patterns of the mitosis approach reached the global optimum solution because of their accurate, full coverage (see Figure 5.5.c). In contrast, TN patterns of the standard approach only reached local optimum solutions because of their partial coverage (see TN patterns learned by the standard approach in three different trials in Figure 5.5.d, 5.5.e, and 5.5.f respectively). It is worth noticing that the combination of three local optimum solutions of the standard approach could provide the global optimum solution as in the mitosis approach. This also suggests that the standard approach has the potential to generate all the accurate rules that are necessary to represent a globally optimum solution. But, the standard approach failed to maintain some of these accurate rules in the later learning processes (see Section 4.2.1), hence this approach only achieved the local optimum solutions in this scenario. In contrast, the mitosis approach was able to maintain accurate classifiers because of the introduction of the accuracy pressure.

The second point was highlighted in doorway areas, such as Scenario No.4 and No.24. Again, the mitosis approach reached the global optimum solution and the standard approach failed (see i-l and o-r in Figure 5.5). Specifically, TN patterns of the mitosis approach contained the narrow doorways that the standard approach was less likely to identify them (see comparisons between (i) and (k), (j) and (l), (o) and (q), and (p) and (r) in Figure 5.5). These suggest that the generalized pressure in the standard approach is insufficient to learn accurate classifiers from such territories.

These also suggest that the accuracy pressure introduced by the mitosis approach can drive classifiers evolving toward maximally general and accurate classifiers.

Patterns also highlighted a complementary relationship. TP and TN patterns approach to the same ground truth pattern in open scenarios in opposite directions. A TP pattern approaches the ground truth pattern from its "upper boundary", because it tends to include neighbour spaces, which belong to boundaries of two types of spaces (see Figure 5.4.c and 5.4.d). Therefore, a TP pattern can be considered as a maximally general, accurate pattern. In contrast, a TN pattern becomes a "lower boundary" of the ground truth pattern, because it tends to exclude such neighbour spaces (see Figure 5.4.e and 5.4.f). Therefore, a TN pattern can be considered as a minimally specific, accurate pattern. Both of a maximally general, accurate pattern, and a minimally specific, accurate pattern can be represented as the robot's perceptions of the environment.

However, both the mitosis approach and the standard approach did not generate robust FP and FN patterns, which predict space that is occupied by obstacles. FP and FN patterns were relatively rarely generated in all the 31 navigation scenarios. This is because of inaccurate instances within scenarios. In these scenarios, the obstacle-occupied space contains a high proportion of inaccurate instances that advocate the open space. Due to the lack of a sufficient amount of the accurate instances, FP and FN patterns are much less frequently generated compared with TP and TN patterns.

A success for real-world application of this work depends on the noise and uncertainty in the world. If the noise has the same or less magnitude and the similar distribution as in the simulations, the system can function well. If the noise has larger magnitude or different distributions, then as long as the number of inaccurate instances that are caused by the noise is under a known accuracy limit, the system can still function because of SAOC pattern robustness. Once the known accuracy limit is exceeded, further experiments will be needed to investigate the effects of the noise.

5.3.3 Summary

This Section proposed an emotion inspired cognitive architecture for robotic adaptive path-planning without hand-coded prior knowledge. Firstly, *SAOC* knowledge is the basis of the proposed cognitive architecture, which was beneficial to navigation adaptation. Experiments showed that the architecture was able to generate valid paths in all

the 31 navigation scenarios. Secondly, the SAOC patterns can be learned by the novel algorithm, the *mitosis approach*, instead of hand-coded hyperparameters for targeted scenarios as prior knowledge. Learned *SAOC patterns* (*IR patterns*) are plain and interpretable for humans. The visualisations of *TP pattern* and *FP pattern* showed that these patterns accurately describe open space within the navigation scenarios. Finally, our mitosis approach performed better than the standard approach as the novel accuracy pressure improved performance in terms of *prediction accuracy*, *pattern robustness* and *pattern accuracy*.

5.4 Experiment Three: Deliberation-establishing

Psychological frustration is an experienced emotional response to the opposition that resists the fulfilment of an individual's goal. For a cognitive robot, an artificial frustration pattern can be categorised into an affective response to an event that causes the obstruction of the completion of the given task. An example event can be a physical roadblock that probabilistically occupies the targeted position of a mobile robot. An occupation is a negative event that can trigger a sensation of frustration during the cognitive robot's navigation task. The intensity of the sensation depends on the estimation of the future, potential result of the event. The intensity also increases as the perceived waste of time continues. Under the sensation of frustration, the robot can select an admissible action with the best, estimated result as its response to the event, then execute the action toward the result. After the execution, the outcome of the execution is provided for the robot to update its previous estimated result, improving the accuracy of the estimation and making the frustration pattern more sophisticated through an evolving process. The frustration hypothesis is that, under the guidance of a sophisticated frustration pattern, a cognitive robot can choose the best response among other alternatives, overcoming the frustration-causing event and moving toward task completion.

This section starts with the scenario set-up of the experiment, in which the frustration patterns emerges. A dynamic obstacle is introduced into the environment to trigger a Frustration event when the robot is interacting with the environment. The Deliberation node aims to learn its mappings to the best Strategy node as a response to the event. That is, the Frustration patterns can emerge from the encapsulated SAOC rule set of the Deliberation node after multiple iterations of the trials. The experiment shows that the

Frustration patterns as the result of the Deliberation node. 12 Frustration patterns are achieved by the node with interpretable symbolic meanings.

5.4.1 Scenario Set-up

This section introduces an experiment to support the above frustration hypothesis. The experiment concerns a robotic navigation task within a real-world office [5]. This real-world environment contains one beginning position and two targeted positions, and the task of a robot is to reach a targeted position in a multistep scenario. To simulate a human's occupation in the targeted position, a dynamic obstacle occurs, occupies the first targeted position for 15 seconds, then moves away. This occupation event has a 70% probability to occur in the trials of the experiments, triggering the sensation of frustration. This probability aims to create a balanced scenario distribution among various obstacle-occupied and obstacle-free scenarios (Table 5.5). This dynamic environment provides a multistep scenario for the mobile robot (the Pioneer) to learn the frustration pattern. Specifically, the Deliberation node in the secondary reinforcer layer (see the Deliberation node in 3.3.2.3) is assigned to learn this affective pattern during the node's engagement in an affective decision-making process of the task. The probability is set to 70% to create a balanced niche between different categories of the SAOC rules.

Table 5.5: Scenarios Distribution

	obstacle-occupied scenarios			obstacle-free scenarios		
Delay	0s	10s	20s	0s	10s	20s
Probability	23.3%	23.3%	23.3%	30%		

The Frustration patterns are encapsulated in a SAOC rule set that engages in the affective decision-making process in each trial of the task. As described in 3.3.2.3, Frustration patterns are fitness SAOC rules that contain three major parts: Stimuli part, the Action part and the Outcome part. The SAOC rules that interact with the environment during a trial of the task are illustrated as follows:

[5] Office 409, Alan MacDiarmid Building, VUW

(1) Stimuli part. The representation of the emotion of frustration is the condition of a SAOC rule to make decisions. This frustration representation subsumes two stimuli. The first stimulus comes from the Occupation node, which is a primary reinforcer that represents the sensation of the occupation event. When an obstacle is detected by the Occupation node, the value that is generated by the node and is embedded in the SAOC rules is 1, otherwise is 0 (see "C1V" in Table 5.6). Because this detection is not an exclusive function of the Occupation node, other primary reinforcer nodes (such as the Path node) can also achieve a similar detection. Yet, this Stimuli subsumes the Occupation node rather than the Path node because of the time consumption. The Path node takes about 10 seconds to generate the result, and the Occupation node takes less than 1 second because of its application of the CNN model (Section 3.3.1.5).

The second stimulus is a timer that increases the intensity of the sensation of the frustration. When an occupation event is detected, the timer starts and its increasing value represents the increasing intensity of the frustration. Before the timer and the intensity reach a threshold, the robot waits at the initial position. In this experiment, 0 second, 10 seconds, and 20 seconds are three optional thresholds of the waiting time (see "C2V" in Table 5.6). These three optional thresholds create different Outcomes when the agent chooses different Actions (see Outcome and Action below). A sub-hypothesis is that different thresholds of the frustration will bring various consequential outcomes by selecting different actions. Although the number of the options tested are limited to three by the time consumption that a real-world experiment takes, the approach itself is theoretically not subjected to this limitation.

(2) Action part. The response of the frustration is the action part of the SAOC rule, indicating what the decision needs to be made under the current condition. Two Strategies are available for the response: Persistence or Rescheduling (see 3.3.4). If Persistence is considered to be a better option than Rescheduling, the robot will focus on its navigation to the first targeted position in the current trial, even if the targeted position is constantly inaccessible. This strategy is embeded in the SAOC rules as a symbol of "one" in the action part (see "ACT" in Table 5.6). Otherwise, the robot will aim to move forward to the alternative position, even

if the roadblock moves away and the first targeted position becomes accessible. Thus, the action part of the SAOC rules will be filled with a symbol of "two". After the Deliberation node selects the Strategy, the robot will execute the selected Strategy till the end of the trial.

Other Strategies can be added as a potential response of the Deliberation node into the action part. A Strategy node, called "freeze", which can suspend the robot's executional commands, used to be the third options in the action part. Yet, this third option exponentially increases the robot's training time in this real-world scenario. Therefore, this node is removed from the action part for the time constraint of the experiment.

(3) Outcome part. The robot's performances are the Outcome part of the SAOC rules. The performances can be considered as the result of the execution of the selected Strategy. To estimate the effect of the selected Strategy, two elements are considered for the Outcome: the achievement of the goal and the associated time-consumption. The achievement is represented by the reward that the robot perceived through the primary reinforcer of the Reward node (see details of the Reward node in 3.3.1.6). When the Pioneer reaches any one of these targeted positions, it will receive a reward with an arbitrary value of 1000. Otherwise, when the Pioneer fails to achieve the targeted position, it will receive a reward with an arbitrary value of 0. In addition to the reward, a timer records the entire time-consumption of the task during the Pioneer's navigation performance, including the waiting time. Thus, the Outcome part is updated by a formula that equals the reward divided by the time-consumption. Through these two elements, a high fitness SAOC rule will represent a solution/pattern that can lead to a successful achievement with less time consumption than other low-fitness solutions.

High fitness SAOC rules, which are expected to emerge from the SAOC rule set, are termed as frustration patterns. 12 frustration patterns [6] emerge as the SAOC rule set evolves by an RL learning agent of the Deliberation node. The symbolic meaning of these patterns is shown in Table 5.7. As discussed in 3.3.2.3 and 4.2, the mitosis

[6] The number of the types of the SAOC rules equals to the combination of the Stimuli part and the Action part of these SAOC rules, and there are 12 combinations. (12 = 2 Stimuli (part 1) * 3 Stimuli (part 2) * 2 Actions)

approach of the XCS algorithm is applied by the RL learning agent that is underlying the Deliberation node, evolving the SAOC rule set. As the RL learning agent goes through trials and iterations of the task, frustration patterns can emerge as the SAOC rules approach their fittest states. By subsuming the primary reinforcers and the Strategies, this secondary reinforcer (the Deliberation node) allows the robot to make a long term decision, performing a "frustration-like" response.

5.4.2 Result of Deliberation-establishing

Frustration patterns become clear in the experienced SAOC rules as the RL agent runs through more than 150 iterations. The experienced SAOC rules of the Deliberation node are shown in the Table 5.6. All the rules which experienced less than 10 iterations are removed from the table, because they are defined as inexperienced rules in this experiment. Theoretically, at the fastest convergent rate, a SAOC rule (an XCS classifier) requires 16 iterations to reach its final accuracy value due to the RL agent's updating method (see Equation 4.1 and the related discussion in Section 4.3.2) [7]. If an iteration takes about 90 seconds, 16 continuous iterations cost 24 minutes for a SAOC rule to reach an accurate state in the real-world environment. Thus, in this case, an entire population/set of SAOC rule takes at least 9.6 hours[8] to achieve accurate states in the most ideal case. When we take the real-world noise, the algorithm's evolutionary processes, and the real-world event distribution into consideration, it takes much longer than 9.6 hours in reality. Therefore, we bootstrap the classifiers' accuracy in their first three iterations to reduce this time consumption (Equation 5.1). Although this bootstrapping encourages the early emerging of fit rules, it is still fair to say a classifier is an inexperienced one if it was experienced less than 10 iterations [9] .

[7]Specifically, the Bellman equation requires at least 16 continuous iterations to approach accuracy with the standard learning rate ($\beta = 0.2$) and the standard error threshold ($\varepsilon_0 = 5\%$)

[8]9.6 hours = 24 minutes * 12 (patterns) * 2 (positive and negative rewards)

[9]In XCS simulated problems, a classifier is considered as an inexperienced one when its experience is less than 25 iterations [76]. This work decreases this inexperienced threshold from 25 iterations to 10 iterations because of the constraint of time consumption to train a robot in the real world.

$$acc_{exp} = \begin{cases} (acc_{exp-1} * (exp-1) + acc_{current})/exp, & \text{if } exp <= 3 \\ \beta * acc_{current} + (1-\beta) * acc_{exp-1}, & \text{if } exp > 3. \end{cases} \quad (5.1)$$

where exp and acc are statistics of a classifier. exp is the experience of the classifier, acc is the accuracy of the classifier.

The result shows that 12 expected frustration patterns are achieved by the Deliberation node. Fit SAOC rules that are achieved by the Deliberation node are shown in Table 5.6. These high fitness SAOC rules cover 12 types of frustration patterns, which are enumerated in Table 5.7. The analysis of their statistics (i.e. "prd" [10]) can provide insights about their meanings and effects that are learned from the experiment:

(1) When there is no frustration event, hesitation brings no benefits, and selecting the strategy of Persistence always allows the robot to have a better performance than the strategy of Rescheduling. There are two underlying reasons:

Firstly, the waiting time of the hesitation is added to the final performances. This is clearly suggested by the comparison of the time consumptions between Patterns of 1, 3 and 5 (or Patterns of 2, 4 and 6). For example, Pattern 1 takes around 17s to complete a navigation task, and Pattern 3 takes around 28s. The major difference of the time consumptions is assumed to be from the waiting time.

Secondly, Persistence is always a better Strategy than Rescheduling. This is because that the first targeted position is closer to the beginning position than the alternative one. Therefore, Persistence allows the robot to navigate to the first position to complete the task, avoiding travelling an additional journey that takes about 7 seconds.

(2) When the frustration event happens in this experiment, then Rescheduling is the best Strategy among the alternatives. Pattern 8 allowed the robot to reach the alternative position and complete the task within the shortest time consumption among Pattern 7 to 12. This is reasonable because the obstacle that triggers the

[10] "prd" stands for "predicted reward", which is a standard statistics parameter of the XCS algorithm, see its definiton in Section 5.2.2. See values of the "prd" in the Table 5.6.

Table 5.6: Experienced SAOC rules (XCS classifiers)

ID	C1L	C1V	C1H	C2L	C2V	C2H	ACT	PRD	ERR	ACC	FIT	EXP	NICHE	NUM
18	-0.01	0	0.01	-0.78	0	0.78	one	473.5039	8.6723	1	0.5344	17	13.1181	10
2	-0.01	0	0.01	-0.78	0	0.78	one	473.3284	10.3532	1	0.4512	19	13.1069	8
3	-0.01	0	0.01	-0.78	0	0.78	two	345.1114	20.3183	1	0.5189	18	14.987	10
26	0	0	0.01	-0.78	0	0.78	two	345.4354	19.4608	1	0.2307	15	15.0635	5
24	-0.03	0	0.03	9.6	10	10.4	one	280.2623	26.0327	1	0.6822	15	3.27	1
42	-0.03	0	0.03	9.6	10	10.4	one	270.8018	53.2833	0.0826	0.0758	14	3.3549	1
85	0	0	0.97	9.6	10	10.4	one	280.6018	45.1247	1	0.1383	11	3.2902	1
62	-0.03	0	0.03	9.6	10	10.4	two	174.2181	12.6545	1	0.2474	11	20.7496	9
22	-0.02	0	0.02	19.25	20	20.75	one	136.6944	3.8208	1	0.3422	19	20.0229	7
23	-0.02	0	0.02	19.25	20	20.75	two	21.3316	70.4973	0.0357	0.8285	13	8.2014	9
4	0.99	1	1.01	-0.59	0	0.59	one	212.992	11.5416	1	0.4387	22	23.314	17
14	0.99	1	1.01	0	0	0.59	one	213.1158	11.1982	1	0.4125	21	23.3199	14
89	0.99	1	1.01	-0.59	0	0.59	one	211.5218	17.6825	1	0.0938	19	23.3384	5
105	0.99	1	1.01	0	0	1	one	213.0614	16.5392	1	0.0476	17	23.3496	3
16	0.99	1	1.01	0	0	0.59	two	238.7742	103.558	0.0113	0.3212	32	6.2463	2
5	0.99	1	1.01	-0.59	0	0.59	two	238.7965	103.6015	0.0112	0.1595	34	6.2454	1
40	0.99	1	1.01	-0.59	0	0.59	two	238.6401	103.8332	0.0112	0.1554	31	6.247	1
56	0.99	1	1.01	0	0	2.59	two	238.6567	103.8949	0.0111	0.1538	28	6.25	1
134	0.99	1	1.01	-0.59	0	0.59	two	239.7318	102.83	0.0115	0.173	23	6.2621	1
9	0.98	1	1.02	9.7	10	10.3	one	183.7456	47.4313	1	0.3162	40	25.3996	10
20	0.02	1	1	9.7	10	10.3	one	183.751	47.4121	1	0.1853	39	25.3996	6
88	0.98	1	1.02	9.7	10	10.3	one	184.6886	45.4307	1	0.1597	16	25.4954	6
81	0.02	1	1.02	9.7	10	10.3	one	183.7173	47.5225	1	0.1037	35	25.4023	4
78	0.98	1	1.02	9.7	10	10.3	one	184.4562	46.861	1	0.0566	20	25.4177	4
101	0.98	1	1.02	9.7	10	10.3	one	183.7299	47.5118	1	0.0838	34	25.403	3
108	0.98	1	1.02	9.7	10	10.3	one	183.4754	49.4954	1	0.0359	14	25.6411	2
32	0	1	1	9.7	10	10.3	two	110.0379	62.2388	0.0518	0.2862	43	23.6907	9
8	0.98	1	1.02	9.7	10	10.3	two	93.0323	62.5086	0.0512	0.3589	36	20.262	9
102	0	1	1	9.7	10	10.3	two	109.836	60.9082	0.0553	0.0628	13	24.2283	1
86	0.98	1	1.02	9.7	10	10.3	two	86.915	66.5502	0.0424	0.0262	11	20.7636	1
162	0	1	0.98	20	20	20	one	136.7826	4.6962	1	0.1719	13	20.4143	5
1	0.98	1	1.02	20	20	20	one	108.7281	48.532	1	0.1249	40	18.7119	4
28	0.98	1	1.02	20	20	20	one	108.717	48.5939	1	0.0911	36	18.713	4
10	0.98	1	1.02	18	20	20	one	108.7289	48.5291	1	0.0949	38	18.7123	3
93	0.98	1	1.02	18	20	20	one	108.7119	48.9937	1	0.0594	28	18.7183	3
0	0.98	1	1.02	20	20	20	two	41.0909	44.1464	1	0.4922	29	24.6486	14
12	0.98	1	1.02	20	20	20	two	41.1009	43.9625	1	0.3978	26	24.6523	10

Table 5.7: 12 Frustration Patterns

Pattern ID.	Description and Classifier IDs
1	No Frustration (C1V=0), No Hesitation (C2V=0), thus select the strategy of Persistence (ACT=one). Classifier IDs: 2,18. Time consumption: 17.8 +/- 0.4 s
2	No Frustration (C1V=0), No Hesitation (C2V=0), thus select the strategy of Rescheduling (ACT=two). Classifier IDs: 3,26. Time consumption: 24.3 +/- 1.1 s
3	No Frustration (C1V=0), Little Hesitation (C2V=10), thus select the strategy of Persistence (ACT=one). Classifier IDs:24,42,85. Time consumption: 28.1 +/- 1.5 s
4	No Frustration (C1V=0), Little Hesitation (C2V=0), thus select the strategy of Rescheduling (ACT=2). Classifier IDs: 62. Time consumption: 35.1 +/- 0.9 s
5	No Frustration (C1V=0), Medium Hesitation (C2V=20),thus select the strategy of Persistence (ACT=one). Classifier IDs: 22. Time consumption: 38.0 +/- 0.3 s
6	No Frustration (C1V=0), Medium Hesitation (C2V=20), thus select the strategy of Rescheduling (ACT=2). Classifier IDs: 23. Time consumption: 47.9 +/- 0.3 s
7	Frustration (C1V=1), No Hesitation (C2V=0), thus select the strategy of Persistence (ACT=one). Classifier IDs: 4,14,89,105. Time consumption: 32.4 +/- 0.7 s
8	Frustration (C1V=1), No Hesitation (C2V=0), thus select the strategy of Rescheduling (ACT=2). Classifier IDs: 5,16,40,56,134. Time consumption: 30 +/- 6.2 s
9	Frustration (C1V=1), Little Hesitation (C2V=10), thus select the strategy of Persistence (ACT=one). Classifier IDs: 9,20,78,81,88,101,108. Time consumption: 34.4 +/- 3.3s
10	Frustration (C1V=1), Little Hesitation (C2V=10), thus select the strategy of Rescheduling (ACT=2). Classifier IDs: 8,32,86,102. Time consumption: 40.1 +/- 4.8 s
11	Frustration (C1V=1), Medium Hesitation (C2V=20), thus select the strategy of Persistence (ACT=one). Classifier IDs: 1,10,28,93,106. Time consumption: 39.3 +/- 2.1 s
12	Frustration (C1V=1), Medium Hesitation (C2V=20), thus select the strategy of Rescheduling (ACT=2). Classifier IDs: 0,12. Time consumption: 46 +/- 3.9 s

frustration event is designed to occupy the first targeted position for 15 seconds. Rescheduling to the alternative targeted position only cost 7 more seconds to travel the extra journey. Therefore, instead of waiting for the obstacle to move away, Rescheduling is the best Strategy when the obstacle is detected.

In addition, Pattern 8 also illustrates the impact of the dynamic obstacle on the task performance. Compared to the performance of Pattern 2, Pattern 8 took more time on average with a large error. This is because the dynamic obstacle creates a narrow area where the path-planning model of the robot has to plan a curved path through. When the robot follows a curved path, it has a slower speed than when it goes straight. The repetition accuracy also drops when the robot makes a turns. The accuracies of the classifiers of Pattern 8 also indicate this impact. Their accuracies are the lowest among all 12 Patterns.

(3) Pattern 7 and Pattern 9 have similar performances in the task. This is because, in these two scenarios, the robot has to wait 15 seconds until the obstacle has moved away and the first targeted position becomes available again. These two patterns are complete solutions to Pattern 8 at the current experimental setup. This is because the time consumption, which Pattern 7 and 9 spend in waiting, almost equals to the time consumption that Pattern 8 spends in travelling for the extra distance of the alternative targeted position. If the alternative targeted position is set in a further position from the beginning position, these two patterns could outperform pattern 8.

(4) Performances of the patterns becomes similar when the event has little impact on the navigation scenario. Pattern 5 and Pattern 11 have similar performances because the waiting time is too long to eliminate the effects of the obstacle. That is, Pattern 5 and Pattern 11 require the robot to wait for 20 seconds, while the dynamic obstacle only occupies the targeted position for 15 seconds. The same reasoning is also applicable to Pattern 6 and Pattern 12.

5.4.3 Summary

The experiment was conducted on the secondary reinforcer of the Deliberation node, which aims to establish frustration patterns in a dynamic real-world environment. The

Deliberation node in the middle level of the hierarchy subsumes the low-level nodes, including primary reinforcers and strategy nodes, to construct appropriate components for the affective pattern. As a result, the node achieved 12 frustration patterns that allow a robot to extricate itself out of the negative impacts of a dynamic environment by adopting the best strategy.

5.5 Experiment Four: Emotion model

This experiment attempts to construct an emotion model that can necessitate decisions and respond to events in a robotic navigation scenario. Emotion states [11] can be elicited by events, and the emotion model can automatically learn appropriate affective responses to these emotion states. This emotion model is achieved by high-level subsumption operations that subsume the low-level nodes in ACMCA architecture. As reviewed from the research of the neuroevolutionary cognitive system (Section 2.2), human emotions are evidenced in the cognitive system and the brain's emotional-affective hierarchy [43]. This emotion model and its high-level subsumption operations allow ACMCA to construct complex, hierarchy knowledge from simple, diverse components.

Emotion theories (Section 2.3) attempt to provide interpretations of these cognitive processes. Three emotion theories, including Appraisal Theory, Constructive Theory, and Basic Emotion Theory, give their perspectives on emotion-related cognitive processes. These theories attempt to explain cognitive processes, such as how emotions are elicited and how they affect responses. Inspired by these explanations, this work proposes an emotion model that makes decisions by perceived events at the core affect state node (Section 3.3.3). This emotion model is designed as an event-emotion-response mapping structure, which subsumes events, emotions [12], and responses. Events are represented by primary reinforcers (Section 3.3.1) and secondary reinforcers (Section 3.3.2), emotions are core affect states (Section 3.3.3) elicited by the events, and the responses includes strategies (Section 3.3.4) and behaviours (Section 3.3.3). Various event-emotion-response mappings are encapsulated in the SAOC rules of a learning agent of the core affect state node. As the learning agent evolves by iterations, fit SAOC rules and event-emotion-response mappings can emerge. A learnt emotion model is

[11] core affect states, see Section 3.2.3

[12] also called emotion states or core affect states

the set of these fit SAOC rules, which are high-level, abstract, interpretable knowledge that establishes fit mappings between symbols of the low-level, concrete knowledge of primary reinforcer, secondary reinforcer, strategy and behaviour. Therefore, a learnt emotion model is reusable and transferable: an emotion model generated on a robot can be easily transferred to other homonomous and heteronomous robots that also deploy ACMCA.

To test the ability of ACMCA to learn such an emotion model, two events are introduced into the experiment: a routine navigation event and a special event. The routine navigation event activates standard navigation modules for a robotic navigation task. The special event interrupts the navigation event thus it requires auxiliary processes before the navigation event can be resumed. These two events are designed to provoke emotion states that necessitate decisions during the navigation task. Because emotion states are considered to be high-level states that summarise events, these two events do not directly elicit emotion states, but appraisals of events do as the Appraisal Theory suggests.

Following the Appraisal Theory, elicited core affect states (emotion states) can be directly assessed by their hedonic value and arousal value. The hedonic value indicates the pleasure level of a robot, demonstrating its attitude toward different stimuli during events. A stimulus is from a secondary reinforcer, the Reflex node (Section 3.3.2.1), producing the hedonic value that comes from the "pain" pattern for responding to unpredicted collisions. This stimulus is thus transferred into a normalized hedonic value for the current core affect state (see Equation 5.2). When a collision is detected by bumpers, the hedonic value equals to an arbitrary value of -1000. Otherwise, the hedonic value equals to an arbitrary value of 1000. In future, other stimuli that come from different nodes or the same stimulus from different time schedules (i.e. the future or the past) can be introduced by their normalized hedonic values to affect the core affect state.

$$reward_{pain} = stimulus_{pain} \tag{5.2}$$

$$reward_{vel} = \lfloor (45500 * vel_x^2 + 3400 * vel_w^2) \rfloor, \tag{5.3}$$

where vel_x is the linear velocity and vel_w is the angular velocity, weights of vel_x and

vel_w are to schedule $reward_{vel}$ to a range between 0 and 800. To emphasis the effect of the linear velocity, its weight is set to be above 10 times of the weight of the vel_w. Yet, weights of vel_x and vel_w, and the range can be customised to any scaled value.

$$reward_{emotion} = reward_{vel} + reward_{pain}, \tag{5.4}$$

The second appraisal that elicits the core affect states is the arousal value. The arousal value indicates the activation level of a robot. In the experiment, this arousal value comes from a primary reinforcer, the Speed node (Section 3.3.1.8), which records the robot's current velocity, suggesting the activation level of the robot. The Speed node transfers the current velocity into a scaled momentum of the robot as the arousal value of the current core affect state (See Equation 5.3). The momentum is scaled to facilitate the visualization of core affect states. Through the hedonic and arousal values, events and emotions are connected in the emotion model.

Emotion states in this emotion model are different from those of previous work. In previous work, emotion states are only discrete states without innate meanings (Section 2.6). The symbolic meanings of the emotion state come from the interpretation of the learnt reinforcer-emotion-modifier mappings. That is, these emotion states are labelled by external interpretations. In this work, emotion states (core affect states) have innate meanings that originate from stimuli, which follows the definition of the Appraisal Theory (Section 3.3.3). The meanings of the emotion states are not reliant upon on the interpretation of extra event-emotion-response mappings. Actually, the innate meanings of the core affect states should be coherent with the executable behaviours that are the results of the emotion model, because of the hypothesis of the Action-Outcome (AO) Contingence of the Constructive Theory (Section 2.3). The examination of this coherence can validate the hypothesis of the AO Contingence from a robotics perspective.

Responses will be mapped to emotion states in the emotion model. When a core affect state is elicited, the response that is associated with the core affect state is activated for a robot's executable behaviours. A response can be a specific commond to control robot's behaviour (i.e. the Reflex Velocity node (Section 3.3.5.1)) or an "umbrella" response (i.e. the Path-following node (Section 3.3.5.2)). Umbrella responses are critical for two reasons. Firstly, an umbrella response can subsume related topics that are necessary for an emotional response. For example, the path-following node includes various modules that are engaged in a navigation task, including obstacle-recognising modules,

map-service module, path-planning module, and motor control module. Thus, umbrella responses are necessary for a subsumption system. Secondly, umbrella responses allow the robot to focus on the action tendency as the Basic Emotion Theory suggests. The action tendency categories the robot's executable behaviours by emotion states. Each core affect state leads to a set of executable behaviours of the robot. As a high-level node, the Emotion model focus on establishing appropriate mappings from core affect states to sets of executable behaviours, leaving the construction of the sets of behaviours to the low-level nodes. This approach allows the emotion model to become a high-level, abstract knowledge that can be transferred among other homonomous or heteronomous ACMCA robots.

In this experiment, two emotional responses are optional for each core affect states. The first response is a navigation response, which should benefit the routine navigation event. The navigation response is an umbrella response that subsumes all responses related to a navigation task. To complete a navigation task, these responses include the map service, the path-planning module, and the motor control module. The second response is a reflex response, which should be applied for the special event. As the special event is designed to introduce unpredicted collisions, the reflex behaviours that are encapsulated in the reflex pattern provide experienced solutions for this event. Thus, this response will activate the output of the reflex pattern that was learned by the Reflex-learning node (see Section 5.2).

The emotion model evolves fitness mappings between core affect states and emotional response through multiple iterations. As discussed above, the event-emotion-response mapping is encapsulated in the SAOC rules of the core affect state node. Instead of presetting these mappings, the emotion model learns fitness mappings through a reinforcement learning approach (Section 3.3.3). In this experiment, the SAOC rules, which are inspired by the Constructive Theory, attempt to discover the contingencies between the events, the emotions, the responses and their sequential outcome. In training iterations, the performance of selected mappings will be provided to estimate the fitness of the emotional response. As the performance faces a trade-off between high velocity and little collisions, the performance is estimated by the combination of these two factors (see Equations 5.2 - 5.4). In equation 5.4, this work is biased to the collisions value by setting the maximum arbitrary value of hedonic value larger than the maximum arbitrary value of arousal value. That is, the $reward_{emotion}$ is usually negative in a scenario where

collisions frequently occur. The estimation of the performance is then provided to the XCS agent to update the mappings. The XCS agent that is deployed for the emotion model updates the statistical parameters of the mappings, including the parameter of fitness. As the emotion model iterates during the robot's navigation training process, fitness mappings emerge.

5.5.1 Scenario Set-up

The hypothesis is that the emotion model can learn to respond to various events through internal core affect states and effecting emotional responses through trial and errors. This section introduces an experiment to test this hypothesis. Two events, two responses, and three environments are included in the experiment.

The two events of the experimental scenario are a routine event and a special event. The routine event is a navigation event, which requires the robot to pursue the targeted position through its navigation responses. Ideally, an appropriate response should subsume all the navigation-related processes of perception, decision-making and execution.

The special event is that the robot detects an unpredicted collision, which can be categorised as an uncertainty factor of the environment. The response to this event should guide the robot away from continuous collisions, which happens if the learnt response affects positively. Therefore, a desirable response to the special event should subsume the reflex pattern, which learns the solutions for collisions by the Reflex node (Section 3.3.2.1) in the secondary reinforcer layer.

The navigation response and the reflex response are two responses that are available for the above two events. Instead of applying the standard navigation response, a simulated navigation response is proposed as the appropriate response for the routine event. This is because the standard navigation modules will update the map service in each iteration, and thus create different scenarios during the emotion model's learning iterations. To create a scenario that will be repeated in each of the iterations, the simulated navigation response randomly selects a direction to move forward in this experiment. As long as the simulated navigation response can generally simulate navigation behaviours, the difference between them is less important, because the emotion model attempts to achieve high-level, abstract knowledge.

The reflex response is another symbolic response. It subsumes the reflex patterns

that select a reflex-action as an affected behaviour. As discussed in Section 3.3.2.1, the reflex patterns were achieved by the learning agent of the reflex node. That is, execution of the reflex-action requires cooperations of two learning agents in the different levels of ACMCA.

Three real-world environments with different frequencies of the special event are tested in this experiment. There are an obstacle-free environment, a corridor and a narrow corner (Figure 5.1). The obstacle-free environment serves as the first scenario for the experiment. To create an obstacle-free environment, the emotion model is trained when the robot is suspended and lifted up from the ground. This scenario can determine the form of trained emotion model when only the routine event engages in this scenario.

The second environment is a common corridor (i.e. level-4 of the AM building, VUW). The corridor is an open space where a robot might engage with unpredicted collisions in daily life. In the experiment, the robot has a chance to collide with the walls thus triggering the special event as it carries out the routine event. As the emotion model experiences both the routine event and the special event with a reasonably balanced ratio, the emotion model shows the ability to cooperate with other learning agents from different parts of the hierarchy to complete a navigation task.

The third environment is a corner in the office of AM-409. The corner is a narrow area of 1.5 metres by 1.5 metres, where the special event occurs frequently. As the emotion model is exposed to the environment where the special event could be overwhelming, the emotion model has the potential to show different characteristics from previous test scenarios.

5.5.2 Result of Emotion Model

Results show that the emotion model establishes a fitness mapping between events and responses through internal core affect states in these three environments. Core affect states that summarise the events of the environments are elicited in the core affect space. The elicited core affect states are mapped to the emotional responses by the emotion model. The mappings that were automatically established by the emotion model are fit, indicating that the solutions can be constructed to facilitate the robot decision-making process in the complex scenarios.

Results are demonstrated in the core affect space with the predicted rewards of the

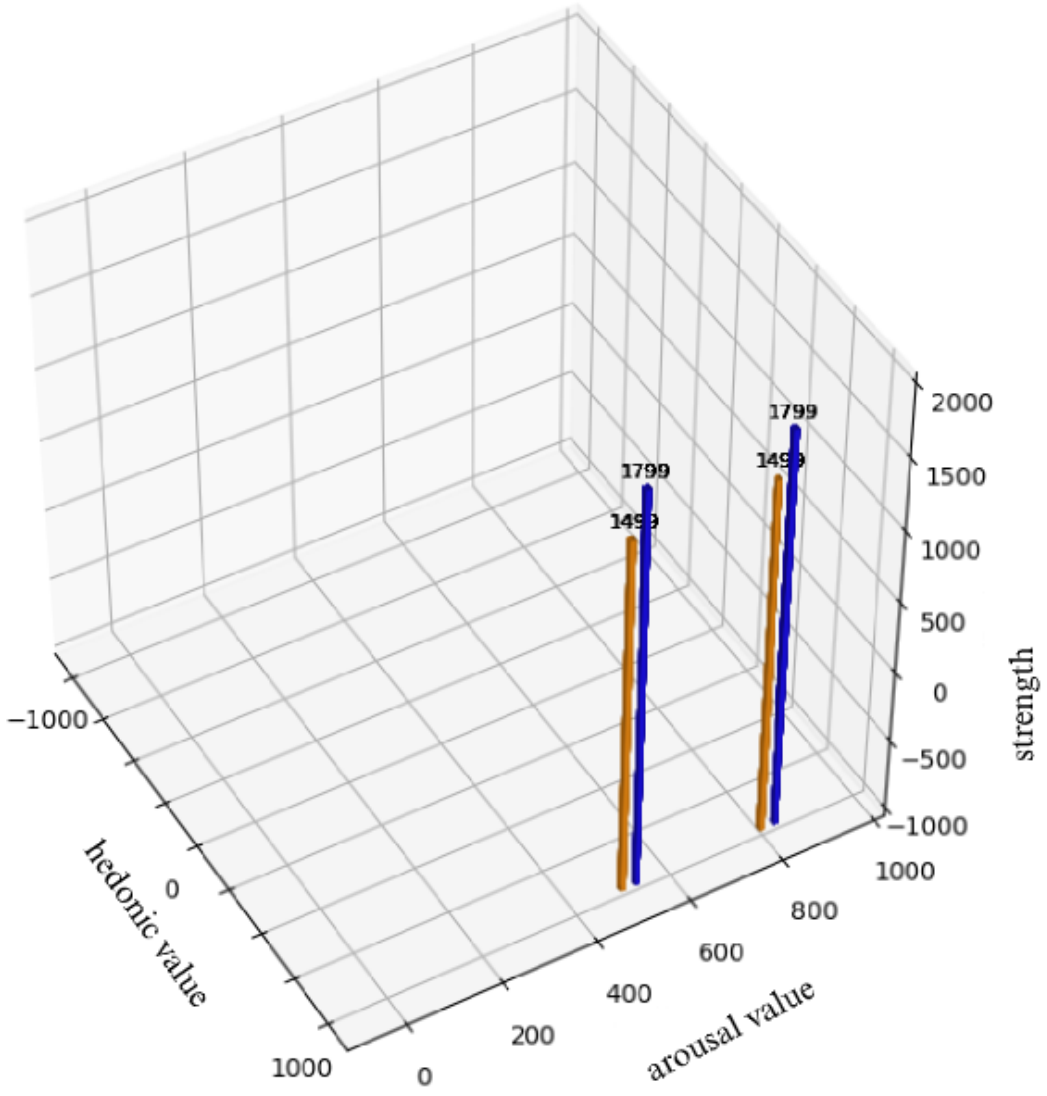


Figure 5.6: Suspension environment

The hedonic-and-arousal plane represents the core affect space (Section 3.3.3). Each bar represents a response to a core affect state. Thus, the location where each bar stands represents a core affect state, and the height of each bar represents the strength of the response that links to the core affect states. Only core affect states of "happiness" are elicited. The blue bar represents the navigation response, and the yellow bar represents the reflex response.

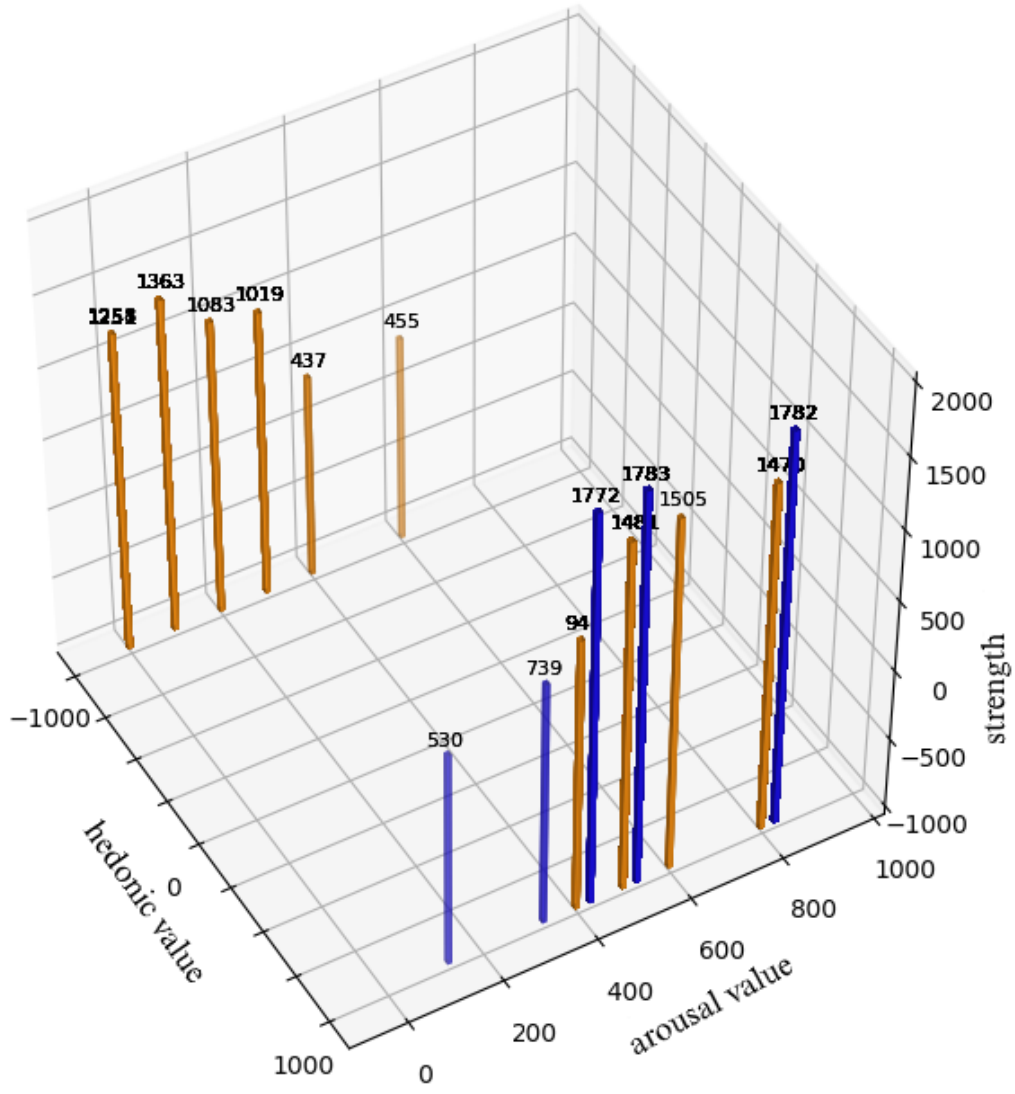


Figure 5.7: Corridor environment

The number of emerged core affect states increase as the scenario get complex. Core affect states of “happiness”, “fear”, and “calm” are elicited. The reflex response dominates the area where core affect states have negative hedonic values.

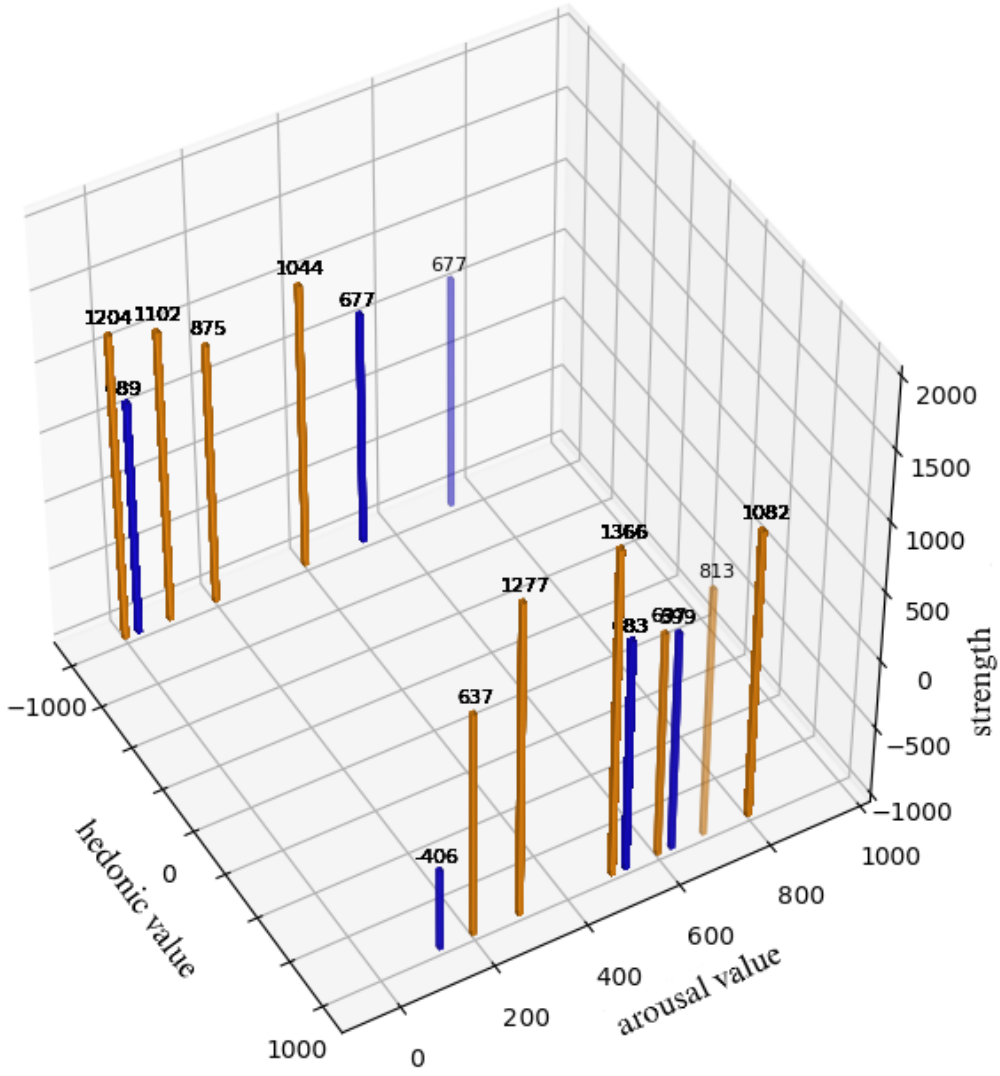


Figure 5.8: Corner environment

Most of the core affect states are biased to the reflex response in this scenario.

mappings (see Figures 5.6, 5.7, and 5.8). As discussed above, the core affect space is represented by the two-dimension space of the hedonic and arousal values. These core affect states can be interpreted by their symbolic meanings. Each appearance of a bar in Figures 5.6, 5.7, and 5.8 indicates an eliciting of a core affect state. The location of a bar is defined by its hedonic and arousal values in the plane, allowing the symbolic interpretation of a core affect state. For example, core affect states that have high hedonic value and high arousal value can be categorised as states of "happiness" by the Appraisal Theory (see the definition of "happiness", "fear", and "calm" in Section 3.3.3). In contrast, those states that have low hedonic value and low arousal value can be categorised as states of "fear". Core affect states that have high hedonic value and low arousal value can be categorised as states of "calm".

The elicited core affect states of happiness, fear and calm suggest a summary of the current events the robot experiences at the given moment. When a robot is in a collision-free environment, only states of happiness are elicited (see Figure 5.6). Only two core affect states are elicited in this environment due to lack of the collisions and the ideal velocities of the robot. The core affect states become diverse when the robot is exposed to environments where collisions happen. Because collisions in these environments impact the ideal velocities of the robot, different hedonic values and arousal value are generated. Other states of "fear" and "calm" are experienced by the robot in the environments of the corridor and the corner. This result provides an example to support an appraisal hypothesis that the complexity of the environment leads to the diversity of core affect states.

These emotion states of happiness, fear and calm are associated with potential consequences of the effected emotional responses. Each core affect state activates emotional responses through mappings, which also estimate the potential consequence of the activated emotional response. The blue bar represents the navigation response and the yellow bar represents the reflex response in the Figures. The estimated value of the potential consequence is kept in the predicted-reward parameter, which represents the emotion model. The predicted-reward of an emotional response predicts the consequential reward that this response can result in. By comparing the strength (/magnitude) of the predicted-reward, the emotion model can advocate the strongest one among these two emotional responses. All the predicted-rewards of the navigation response and the reflex response of all core affect states make up a reward surface of the core affect space

that the robot explored in the environment.

The results show that the emotion model learns reward surfaces from the robot's interactions with all three environments. In the first scenario, the reward surface approaches its theoretical value in the collision-free environment (see Figure 5.6). Both two predicted-rewards of the navigation responses (see the two blue bars in Figure 5.6) approach the upper boundary of 1800 [13]. Similarly, two predicted-rewards of the reflex responses (see the two yellow bars) approach their upper boundary of 1500 [14]. The navigation response has a stronger predicted-reward than the reflex response when core affect states are elicited in this case. This is because the navigation response is considered to be the major response, which can lead to the task completion, and the reflex response is the auxiliary response that facilitates the task completion. Therefore, the navigation response is the better choice than the reflex response that the emotion model will advocate in the collision-free environment as expected.

The reward surface shows a segmentation in the second scenario when the special event is introduced into the corridor environment. The navigation event dominates half of the core affect space and the reflex event dominates the other half, separating by the hedonic value (see Figure 5.7). When core affect states have a positive hedonic value, the navigation responses are the better choice. The navigation responses have stronger predicted values than the reflex response because the navigation response leads to a higher velocity than its alternative response on average. The situation reverses when core affect states have a negative hedonic value. The reflex response enables a better performance because the subsumed reflex pattern can guide the robot away from continuous collisions. Compared with the first control group, elicited core affect states become more diverse as the scenario gets more complex in the second scenario. The emotion model can respond to the events by advocating the optional emotional response.

The reward surface shows that the reflex response is a better choice than the navigation response in the third scenario where the special event happens frequently. The reflex response dominates the entire core affect space (see Figure 5.8), including core affect states with both positive and negative hedonic values. In the area with negative hedonic

[13] This upper boundary can be calculated by applying path-planning velocities in Table 3.3 into Equations 5.6, 5.7, and 5.8.

[14] This upper boundary can be calculated by applying reflex velocities in Table 3.2 into Equations 5.6, 5.7, and 5.8.

values, the reflex response is activated through the subsumption of the reflex pattern in the emotion model. In the area with positive hedonic values, the navigation response is no longer dominate in this area, compared to its domination in the second scenario. This is because the limited space of the corner environment limits the performance of the navigation response. That is, when the robot becomes stuck in the corner, the navigation response is inclined to elicit further future collisions. Thus, the navigation response is inferior to the reflex response even in the navigation event. Therefore, the reflex response is the better option, which the emotion model will advocate in the full-of-collisions environment.

5.5.3 Summary

Inspired by the emotion theories, the experiment was conducted to train the emotion model to learn optional emotional responses to complex scenarios. The emotion model was trained to respond to the two events in the three scenarios. The event-emotion-response mappings are the solution that was learned by the emotion model. Through the visualization of these mapping in the reward surface, the results show that the emotion model learnt correct event-emotion-response mappings in these environments. Compared to the mappings learnt in these environments, the emotion model has also demonstrated its adaptation to the given environment as the emotion model can advocate the optional emotional response for the robot's execution. Although the three inspiring emotion theories focus on different perspectives, this experience provides a valid example that attempts to unify these theories and apply them in a real-world robotic application.

5.6 Experiment Five: Combined Reward Assignment

The previous four experiments show ACMCA's ability to achieve a complex solution through solution components within a multilayer architecture. The fourth experiment, the emotion model, further indicates that learning agents should cooperate to achieve an optimum solution. However, coevolving agents require a credit assignment method that can appropriately assign a final reward to agents that contribute to it. This next experiment introduces maze problems to investigate the credit assignment method in multistep problems. This method can allow multiple separate agents to coevolve for

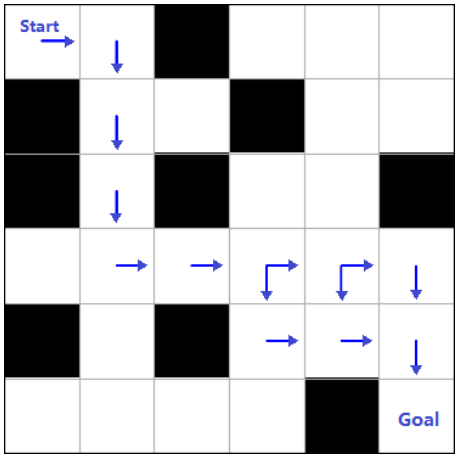
optimum solutions in a robot and lead to the life-long learning behaviours of a robot.

5.6.1 Scenario Set-up

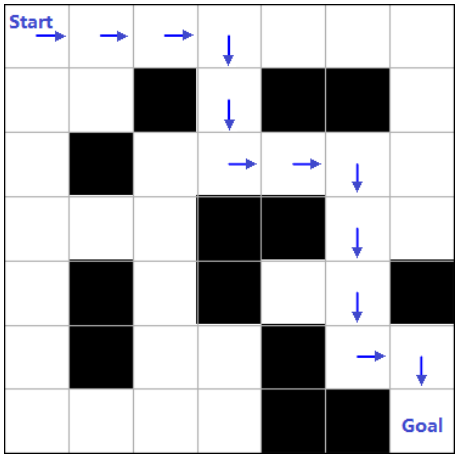
Maze problems provide suitable environments to test an algorithm's performance when searching for global optimal policies. Maze environments include a large policy space but with known and interpretable optimal paths as policies. This work includes experiments in three maze environments: the Maze 4, 5, and 6 (see Figure 5.9). These maze environments have been frequently applied to demonstrate XCS performance on multistep problems [69, 109, 115]. Maze environments contain free paths, obstacles, a starting position and a targeted position. Free paths are shown as white cells that can be occupied by an agent. Obstacles are black cells.

The navigation task requires a robot to start at the left, top cell of the maze, targeting to the right, bottom cell. In each step, the robot activates an XCS agent to select one direction from four options (left, right, up, and down) to move forward as its action. A greater number of directions, e.g. eight, can be used but this increases the search space and more importantly makes visualisation of the results more cluttered so is difficult to interpret the effects of novel methods. Actions toward any black cell or boundary in a step would cause a collision. In these cases, the robot will stay at the same place until the next step.

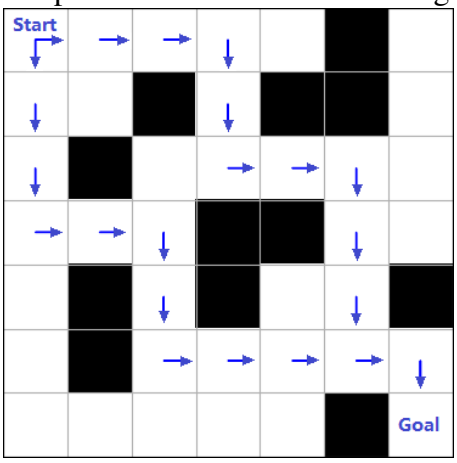
Policies can be learned through robot interactions with a maze. Generally, a policy is any state-action-state succession. In the maze problem, a "state" is the robot's current perception of the environment, and an "action" is a direction that the robot moves forward. Therefore, policies can establish a sequence of actions that are intended to lead the robot to achieve its target position. Policies are often rewarded by the worth of the robot's performances (i.e a worth value of 1000 when the robot achieves the target cell) such that state-action mappings in policies are assigned with values of worth. Therefore, a global optimal policy is a policy with the minimum number of sequential actions that reaches its target. Each action in a global policy has the maximum value of worth among the other three actions available at a given state (see the worth of an action in Equation 4.6). A global optimal policy is a policy with the minimum number of sequential actions that reaches its target. Global optimal policies, which are shown as sequences of blue arrows in Figure 5.9, are the objective that the XCS algorithms attempt to learn.



(a) Maze 4
three global optimal policies



(b) Maze 5
one global optimal policy



(c) Maze 6
two global optimal policies

Figure 5.9: Three Maze Environments
Start position is at the left, top cell of each maze environment, and targeted position is at the right, bottom cell. Global optimal policies/rules are shown with blue arrows.

Experiments were conducted to test the XCSCR's ability to search for global optimal policy in multistep problems. In each maze problem, 100 trials were conducted for each XCS, the adapted XCS algorithm and the proposed algorithm. In each trial, the number of iterations (epochs) it took for an algorithm to achieve a global optimal policy for the first time is recorded. In each iteration/epoch, an XCS agent takes steps to complete a

maze navigation task. In each step of the interaction, the XCS agent interacts with the maze environment through an XCS iteration loop (see Section 4.3.2). The XCS agent is implemented by the XCSCR, the adapted XCS algorithm, and the standard XCS. These XCS algorithms have common parameters with the same settings from Butz [76]. Other parameters that are modified are specified in the method section 4.3.

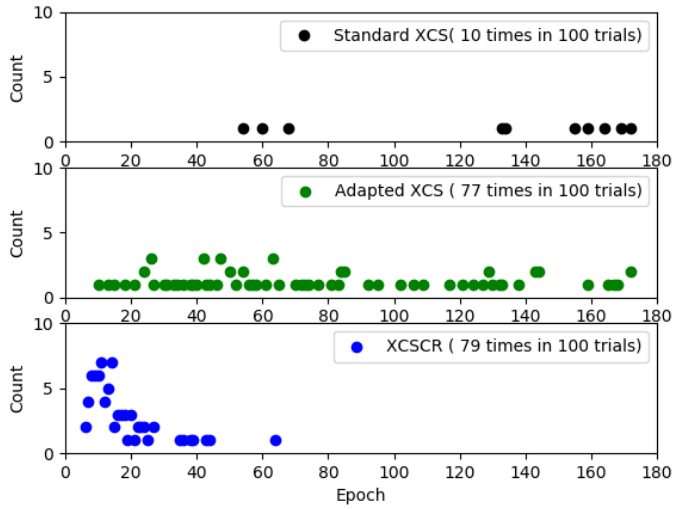
Parameter Settings for XCS algorithms:.

- *N*, the maximum size of the classifier population: 900.
- β, the default learning rate for r_p, ϵ, fit, and θ_{step} : 0.2.
- ϵ_0, the accuracy threshold, which equals 0.05 percentage of the absolute value of the maximum reward: 5,
- γ, the discount fact for the standard XCS multistep problems: 0.8.
- θ_{GA}, the GA threshold: 20.
- θ_{del}, the deletion threshold : 25.
- χ, the crossover probability: 0.8.
- μ, the mutation probability: 0.04.
- p_{explr}, the exploration probability: 0.2

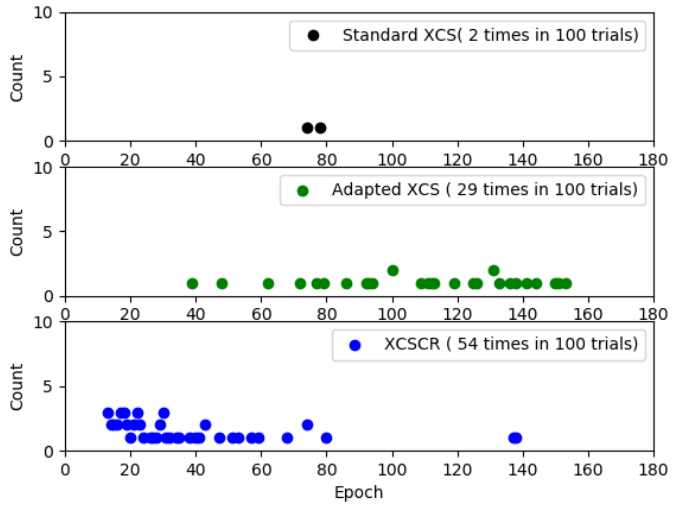
5.6.2 Results of Maze Problems

The result shows that XCSCR performed better than the previous adapted XCS and the standard XCS in all of the three maze environments. The global optimal policy are shown as blue arrows in Figure 5.9. XCSCR achieved the global optimal policy more often than the other two algorithms (Figure 5.10). In Maze 4, XCSCR achieved the global optimal policy 79 times, compared to 77 times of the adapted XCS and 7 times of the standard XCS. In Maze 5, XCSCR achieved the global optimal policy 54 times, compared to 29 times of the adapted XCS and 2 times of the standard XCS. In Maze 6, XCSCR achieved the global optimal policy 65 times, compared to 25 times of the adapted XCS and zero times of the standard XCS within 180 epochs.

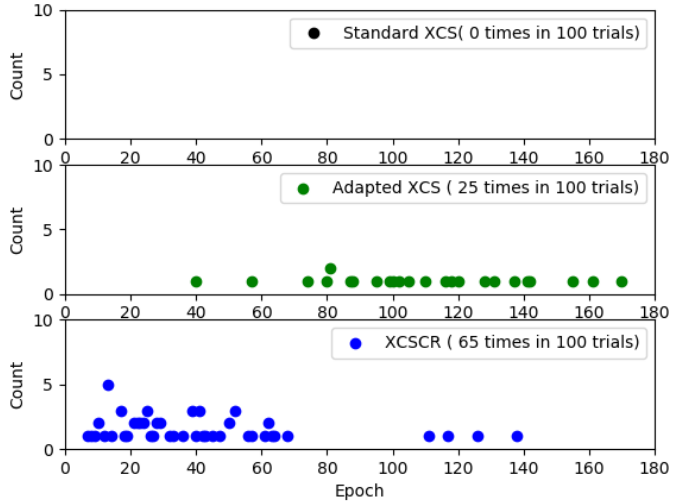
In addition, XCSCR achieved the global optimal policy earlier than the other two algorithms in these three mazes. The distributions of optimum policies of the XCSCR



(a) Maze 4



(b) Maze 5



(c) Maze 6

Figure 5.10: Distribution of Global Optimal Policies in the Maze Problems
The x coordinate of a dot indicates when the global optimal policies first discovered. The y coordinate of a dot indicates the count of times in 100 trials. The higher count at fewer epochs, the better.

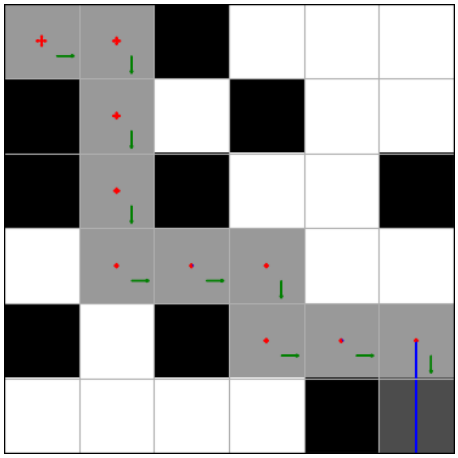
are concentrated on early epochs, while the optimum policies of the adapted XCS and the standard XCS are distributed among the entire epoch-axis (see Figure 5.10). In Maze 4, major achievements of XCSCR happened before the fortieth epoch, while achievements of the adapted algorithm and standard XCS were distributed evenly. In Maze 5, major achievements of XCSCR happened before the seventieth epoch, while achievements of the adapted algorithm majorly were distributed after the seventieth epoch. In the Maze 6, major achievements of XCSCR also happened before the seventieth epoch, while achievements of the adapted algorithm majorly were distributed after the seventieth epoch. Therefore, XCSCR changed the distribution of achievement of the global optimal policy, and XCSCR achieved the global optimal policy more frequently and earlier than the adapted XCS and the standard XCS.

5.6.3 Analyses and Discussion of Maze Problems

Analysis of the agent's learning process illustrates XCSCR's effects on generating policy. The short-term reward mechanism contributes to a good policy emerging in an early learning phase. Since the long-term, positive rewards are scarce at this phase, the worth of rules were estimated by punishments, especially through the short-term reward filter (see Figure 4.3.c). Thus, a policy is based on comparisons on the negative worth of rules (see Figure 5.11.a). In addition, magnitudes of negative worth decrease from the starting cell to the targeted cell in a policy, while magnitudes of positive worth are increasing. This suggests the value of the worth is related to the vastness of the searching space. The further the cell is from the targeted cell, the more vast a searching space will be, and the higher likelihood that rules receive negative rewards.

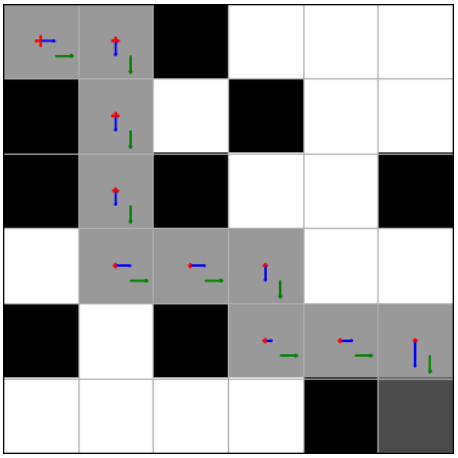
The long-term reward mechanisms tend to increase the stability of a policy as iterations progress. The long-term reward filter, which combines the three novel reward mechanisms with the standard one, can propagate long-term positive rewards to rules. For example, in the first four steps of Figure 5.11.b, the values of the worth of rules, which belong to the policy, turn from negative to positive. In contrast, the values of the worth of rules that suggest a different action from the optimum policy remain negative.

As the training encounters sufficient iterations, the magnitude of the worth of the positive rules increases (see blue arrarys in Figure 5.11). The long-term positive rewards were evenly assigned to rules in different steps. The magnitudes of the worth of these



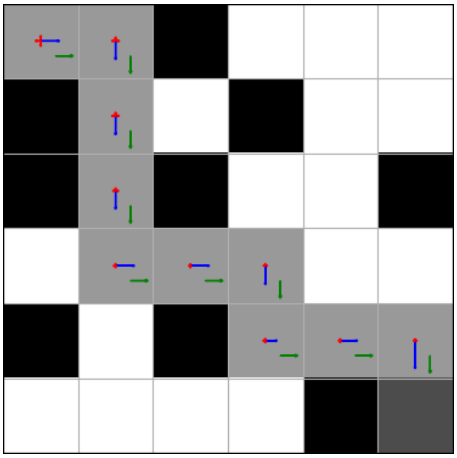
(a) Iteration = 44

The policy is bootstrapped by negative rule worth during the initial learning phase.



(b) Iteration = 79

The policy is bootstrapped by positive rule worth as the training progresses.



(c) Iteration = 111

The positive worth become equivalent after sufficient iterations.

Figure 5.11: Policy Learning Process in the Maze 4

Blue and red arrows: worth of rules, the direction of an arrow suggests the direction of action in the rule, magnitude suggests the value of the worth, red suggests negative worth and blue suggests positive worth. Green arrows: policy, sequential rules with the best worth, the direction of an arrow suggests the direction of the best rule, the magnitude is fixed. Light grey cells suggest they have been visited by the agent in this epoch of a trail. The dark grey cell suggests the location where the agent ended up at the end of the epoch.

positive rules, which advocate for a policy, became more evenly spread in Figure 5.11.c than they are in Figure 5.11.b.

The changing worth of rules in the XCSCR learning iterations also indicates that XCSCR was able to increase the quality of policy in terms of stability. The stability of a policy depends on the difference between the worth of rules advocating different actions in the same state. A policy will increase its stability as the differences increase. In the experiments, the differences along a policy were increased by XCSCR learning (see Figure 5.11), suggesting that the quality of the policy was increasing.

5.6.4 Summary

This section conducted experiments for XCSCR, an XCS algorithm with a combined reward method, to learn global optimal policies in multistep problems. The XCSCR adopts four novel mechanisms to the reward method for better usages of long-term and short-term rewards. Experiments were conducted in three standard maze environments (the Maze 4, 5, and 6) to test the XCS algorithms' performances on their search global optimal policies. Results show XCSCR performed better than the standard XCS and the adapted for robotics XCS methods. In all the three mazes, the XCSCR enabled global optimal policies to emerge earlier and more frequently than the other two approaches. Analyses also provide interpretable insights about the policy learning process. The insights illustrate effects of the four novel mechanisms, which allows XCSCR to increase the stability of the policy throughout the training iterations.

5.7 Chapter Summary

This chapter conducted five experiments for the five-layer-and-three-level cognitive system, Affective Computing Multiplayer Cognitive Architecture (ACMCA), to test its capability of learning solutions for a robotic navigation task. These five experiments were reflex-learning, IR-tuning, deliberation-establishing, emotion model, and combined reward assignment.

The first three experiments were conducted among low-level nodes. The solution components were constructed by the subsumption operations on the secondary reinforcers of ACMCA. These secondary reinforcers showed their capability of constructing low-level solution components that affect a robot's performance from the behaviour, strategy, and model perspectives.

The fourth experiment was conducted among high-level nodes. The solution component, emotion model, was constructed by the subsumption operations on the core affect state node of ACMCA. The core affect state node showed its capability of constructing the high-level solution (e.g. the emotion model) through the established low-level solution components. The emotion models that were constructed by the core affect state node are adapted to different scenarios. This indicates that ACMCA are able to construct complex, hierarchy solution from simple, diverse components.

The fifth experiment was conducted to test the ability of the XCSCR (XCS with a Combined Reward method) to search global optimal policy in multistep problems. This method allows a robot to assign the final credit to previous, contributed policies, aiming to explore ACMCAâĂŹs potential implementations of the life-long learning scenario. This method could allow multiple separated agents to coevolve for optimum solutions in a robot and lead to the life-long learning behaviours of a robot.

The first four experiments cost a different length of time to evolve various SAOC (Stimulus-Action-Outcome Contingency) rules. The time consumptions of these experiments vary primarily based on the components of the SAOC rules. Namely, time consumptions of Stimulus, Action, and Outcome that are embedded in the rules. Generally, the time consumption of Action costs the majority of the time consumption in each learning trial, because the Action will be advocated for the robot's execution. For example, in the experiment of IR-tuning, the dynamic setting of IR (Inflation Radius) for the path planning module takes about 5 seconds. Thus, an iteration of the XCS agent

costs about 7 seconds, including time consumption on the perception and the synchronization among distributed ROS nodes. In the experiment of deliberation-establishing, the Action in its SAOC requires the robot to navigate in the office. Each trail takes 4 minutes on average for the robot to reach the goal position and come back to the starting position. Thus, this experiment runs longer than the earlier one.

Chapter 6

Conclusions

6.1 Summary of Work

This work proposed a five-layer-and-three-level cognitive system, termed an Affective Computing Multiplayer Cognitive Architecture (ACMCA), which learns solutions for a robotic navigation task. The five layers are the primary reinforcer layer, the secondary reinforcer layer, the core affect state layer, the strategy layer, and the behaviour layer. These five layers follow the traditional decomposition of a mobile control system, and each layer contains various computing nodes that perform as functional modules that construct the system. The architecture of the cognitive system provides a three-level hierarchy that encapsulated the learned solution within its five-layer architecture. This satisfied the first objective of this work.

This work decomposed the solution into diverse components within a hierarchy structure by emotion theories. Inspired by psychological hypotheses, diverse solution components cooperate to fulfil a cognitive system's capabilities of reconceptualizing situations, dispositions, dynamics, and invariance in personality structure [1]. In this work, solution components are encapsulated by 17 computing nodes as functional modules in the five layers. These nodes are distributed among five layers in ACMCA by their categories: primary reinforcer, secondary reinforcer, core affect state, strategy and behaviour. All these nodes and their interactions can provide clear symbolic interpreta-

[1]The hypothesis is that there is invariance in personality structure, which is underlying the variability of behaviour across situations [37].

tions for their solution components in terms of emotion theories. Therefore, the second objective of this work was satisfied.

This work proposed a contingency-based subsumption approach to construct the cognitive system. The contingency-based subsumptions can establish connections between computing nodes within the hierarchy structure. Nodes in higher-level can subsume lower-level ones by the proposed underlying XCS algorithms, which learn high-fitness solution components with diverse symbolic contingencies. Such subsumption operations are conducted on the three secondary reinforcer nodes and the core affect state node. The reinforcer nodes establish contingencies of diverse stimuli perceived, and the core affect state node establishes emotional responses to environmental stimuli. As a result, the cognitive system was constructed various contingencies had been established by the proposed machine learning techniques. That is, the third objective of this work was achieved.

This work proposed two novel variants of XCS algorithm as the machine learning techniques that facilitate robotic applications. They are the mitosis approach of XCS and the XCS with a Combined Reward method (XCSCR). The mitosis approach introduces an accurate pressure into the algorithms evolutionary process. As the machine learning technique that underlies the contingency-based subsumption operations, the mitosis approach improves the algorithmsâĂŹ performance in robotic applications where noisy interferences exist. The XCSCR enables the policy to emerge earlier and more frequently than the existing benchmark approaches in multistep problems. Had been tested in the maze environments (the fifth experiment), the XCSCR explored ACMCAâĂŹs potential implementation in the life-long learning scenario. Both of these two variants of the XCS algorithm satisfied the fourth objective of this work.

This work test the proposed ACMCA in both simulated and real-world robotic platforms. Four experiments were conducted to test the robustness of the proposed approach. They are *reflex-learning*, *IR-tuning*, *deliberation-establishing* and *emotion model*. In experiments, three secondary reinforcers (e.g. the Reflex node, the Tuning node, and the Deliberation Node) and an emotion model had emerged. These also indicate that ACMCA were constructed through Reinforcement Learning (RL) approach. Therefore, the fifth objective of this work was achieved.

These experiments also suggest that Various Stimuli-Action-Outcome Contingencies (SAOCs) can be achieved by ACMCA. Contingencies are represented by connections

between various computing nodes that encapsulate various Stimuli, Actions, and Outcomes in different layers, suggesting ACMCA's firing sequences as its responses to the task. In this work, four types of SAOC are achieved and encapsulated in the three secondary reinforcers and the emotion model, suggesting ACMCA's learning and reasoning capability in real-world scenarios. The four types of SAOC and their symbolic interpretations are summarised as follows:

The first type of SAOC is the reflex-pattern. The reflex-pattern describes reflex-like responses based on collisions and the stimulus of the "pain" during the robot's navigation process. The automatic learning process of this pattern engages the Touch node, the Reflex-Velocity node, the Pain node, and the Reflex node. Specifically, the Reflex node subsumes the other three nodes, bringing solution components to unpredicted collisions with the underlying XCS agent. The learned reflex-patterns allow the robot to respond in an appropriate manner to the unpredicted "pain-causing" stimuli instantly, thus avoiding continuous damage.

The second type of SAOC is the IR-pattern. The IR-patterns provide an adaptive approach for path-planning modules. The IR-patterns automatically tune the adaptive hyperparameter, Inflation Radius (IR), to the applied path-planning module, allowing it to generate a valid path. Otherwise, the path-planning module would fail if the hyperparameter is inappropriate. The automatic tuning process happens at the Tuning node, which achieves the patterns by subsuming the Goal node, the Path node, and the IR node. The achievement of the IR-patterns suggests that ACMCA is capable of allowing a functional module to achieve its admissible performance, such as adaptive path-planning, without hand-coded prior knowledge for this module.

The third type of the SAOC is the frustration-pattern. The frustration-pattern enables a robot to extricate itself from the negative environmental impacts by adopting its best strategy. Triggered by the dynamic obstacles in the environment, the frustration-patterns reschedule strategies to achieve early task completion. The frustration-patterns are encapsulated in the Deliberation Node, which subsumes multiple nodes: the Occupation node, the Delay node, the Reward node, the Persistence node, and the Rescheduling node. Although the frustration-patterns were automatically established by the XCS agent, these interpretable artificial patterns can support the psychology-based hypothesis that the personal emotions indicate an individual's physical coping ability that was established through previous experience. In addition, these three SAOCs, the reflex-pattern, the IR-

pattern, and frustration-pattern, also support the Constructive Theory that conversions from primary reinforcers to secondary reinforcers provide fundamental processes for emotional responses.

The fourth type of SAOC is the emotion model. The emotion model was adapted to the given environment so it could advocate the optional emotional response for the robot's execution. Inspired by the Appraisal Theory, when the emotion model is activated by events, the emotion states, termed as core affect states, are elicited in the two-dimensional core affect space, summarising the perception with straightforward interpretations. The elicited core affect states thus activate respective responses, which lead to the robot's subsequential behaviour, based on the Basic Emotion Theory. This emotion model in the high-level is an affective computing model that cooperates with other lower-level solution components, increasing the robot's flexibility.

In sum, this work proposed that ACMCA as a novel emotion inspired multilayer architecture, which can produce task solutions through contingency-based subsumption operations and underlying appropriate machine learning algorithms, allows a robot to complete admissible tasks through evolutionary processes.

6.2 Contributions

ACMCA is a novel emotion-inspired, contingency-based, multilayer robotic cognitive system, which has been tested on a real-world robot to learn solutions for navigation tasks. This work contributes to evolutionary robotics, cognitive science and emotion theory, and Learning Classifier Systems (LCSs). These contributions are summarised below:

(1) ACMCA is a novel five-layer cognitive architecture that can contribute to evolutionary robotics. The architecture of an ACMCA provides a three-level hierarchy that encapsulates a learned solution within its five-layer architecture. Among the five layers, nodes fulfil functional modules that provide diverse solution components with symbolic meanings, leading to the construction of the solution with a symbolic interpretation. The high-level nodes can learn their solution components through the underlying machine learning techniques when the robot interacts with the environment through trial and error. As a result, ACMCA can learn sym-

bolically interpretable solutions toward task completion in real-world scenarios. This work provides a benchmark of a five-layer cognitive architecture, in which homogeneous computing nodes in layers can cooperate to achieve a robotic task in complex scenarios.

(2) ACMCA is a novel emotion-inspired robotic cognitive system that can provide insights for psychological theories, cognitive systems and Artificial Intelligence. The construction in work is inspired by three emotion theories: Constructive Theory, Appraisal Theory, and Basic Emotion Theory. The construction of secondary reinforcers supports Constructive Theory as stimuli are estimated by a sophisticated cognitive system. The construction of the core-affect-state supports Appraisal Theory in terms of how emotions are elicited. The construction of the emotion model supports Basic Emotion Theory regarding how the emotional responses are executed. This work extends the previous work [17, 18], which applies Constructive Theory in the construction of a robotic cognitive system for the navigation task for the first time. Following the previous work, this work combines these three mainstream emotion theories in the construction of the system, providing a novel benchmark for future construction of the emotion-inspired robotic cognitive system.

(3) ACMCA is a novel contingency-based subsumption system that contributes to robotic control systems. Traditionally, the classical behaviour-based subsumption systems are constructed by layers of behaviours. In this work, the system is constructed by layers of Stimuli-Action-Outcome Contingencies (SAOCs). As Stimuli, Actions, and their consequential Outcomes are encapsulated in low-level nodes, subsumptions of these low-level nodes by high-level nodes can establish various SAOCs that construct the system. Based on the established SAOCs, the system can select its action to respond to the detected stimulus and predict the consequential outcome of the action. In the three-level hierarchy, contingency-based subsumptions can operate at the middle level nodes (e.g. secondary reinforcers) and at the highest level node (e.g. the core affect state node). These subsumption operations can construct affective responding patterns from the secondary reinforcers and an emotion model from the core affect state node, automatically constructing a control system that instructs the robot's reactions to complete a

complex task. This work provides a benchmark of a subsumption system that is capable of planning and prediction, which the behaviour-based subsumption systems find difficult to incorporate.

(4) This work proposes two underlying novel XCS algorithms which extend the usage of XCS algorithms for real-world robotic implementations. Two variants of XCS algorithms, the mitosis approach and the XCSCR, are proposed to solve the problems that occur during real-world implementations. The mitosis approach allows the XCS learning agents to learn accurate knowledge even in noisy scenarios. The XCSCR could allow learning agents to co-evolve in life-long learning scenarios by extending the mitosis approach from single-step scenarios to multistep scenarios, These two XCS algorithms provide novel benchmarks for future variants of XCS algorithms in robotic applications.

The following publications were produced during this thesis:

Zhang, Z., Browne, W.N. and Carnegie, D.A., 2018, June. Emotion Inspired Cognitive Architecture for Robotic Adaptive Path Planning. In Australasian Conference on Robotics and Automation 2018.

Zhang, Z., Browne, W.N. and Carnegie, D.A., 2019, June. XCS with Combined Reward Method (XCSCR) for Policy Search in Multistep Problems. In 2019 IEEE Congress on Evolutionary Computation (CEC) (pp. 2982-2989). IEEE.

Zhang, Z., Browne, W.N. and Carnegie, D.A. Affective Computing Multilayer Cognitive Architecture (In process of writing).

6.3 Future Work

Three hypotheses can be tested in future work:

The first potential future work can focus on the hypothesis that increasing diversity of nodes can produce increasing diversity of the learnt solutions. The current work includes 17 nodes in the five-layer hierarchy of ACMCA, and the results show that ACMCA are capable of learning solutions for complex scenarios. This future work can increase the number of nodes in each layer. Because each node represents a different

functional module and a unique solution component, including additional diverse nodes into ACMCA should generate more diverse solutions than the current ones for the same scenarios.

The second potential future work can continue to focus on the hypothesis that applying emotion theories in the construction of a robotic cognitive architecture will bing benefits to robotic tasks. The current work applies the major principles of Constructive Theory, Appraisal Theory, and Basic Emotion Theory in the construction of ACMCA. Psychological mechanisms (i.e. empathy mechanism) of emotion theories provide inspirations for future work.

The third potential future work can focus on the hypothesis that introducing evolutionary processes to the construction of a cognitive system can lead to a completely automatic construction of the system. The current work constructs the system through contingency-based subsumption operations, which allow high-level nodes to subsume low-level ones. A potential future work can introduce an evolutionary process into the contingency-based subsumption operation. The evolutionary process can create, select, mutate, and/or replace nodes during the subsumption operation. XCSCFs (XCS with code-fragment actions/conditions [66]) is a competitive underlying algorithm for this evolutionary process. The code-fragment part of the algorithm can focus on the evolutionary hierarchy, allowing the robot to evolve different types of contingencies through its interactions with the environment. As a result, the evolutionary process should allow the cognitive system to discover new contingencies that increase the robot's performance in the environment.

6.4 Final Summary

This thesis proposed the Affective Computing Multilayer Cognitive Architecture for a mobile robot that can learn diverse task-complete solutions. Inspired by cognitive studies and emotion theories, ACMCA introduces a novel representation of a task solution, which is constructed by diverse components of solutions in the five-layer-and-three-level hierarchy of ACMCA. These 17 diverse components of task solutions are achieved automatically by ACMCA. The subsumption operations can establish three contingencies and one emotion model between the subsumed components by multiple RL agents which deploy proposed XCS algorithms. These three emotion patterns and emotion

model can consistently improve the robot's navigation performance with interpretable explanations. These two variants of XCS algorithms can amend shortfalls of the standard XCS approach in real-world robotic implementations. It has been demonstrated that the diverse solutions learned by ACMCA can improve the navigation performance of the robot by increasing flexibility and reducing continuous collisions and navigation times.

Bibliography

[1] "How robots are helping to fight the coronavirus outbreak." https://spectrum.ieee.org/automaton/robotics/robotics-hardware/robots-helping-to-fight-coronavirus-outbreak. Accessed: 2020-3-29.

[2] S. Nolfi, J. Bongard, P. Husbands, and D. Floreano, *Evolutionary Robotics*, pp. 2035–2068. Cham: Springer International Publishing, 2016.

[3] S. Nolfi, D. Floreano, and D. D. Floreano, *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT press, 2000.

[4] R. J. Alattas, S. Patel, and T. M. Sobh, "Evolutionary modular robotics: Survey and analysis," *Journal of Intelligent & Robotic Systems*, vol. 95, no. 3-4, pp. 815–828, 2019.

[5] A. Faíña, F. Bellas, F. López-Peña, and R. J. Duro, "Edhmor: Evolutionary designer of heterogeneous modular robots," *Engineering Applications of Artificial Intelligence*, vol. 26, no. 10, pp. 2408–2423, 2013.

[6] R. Pfeifer and C. Scheier, *Understanding intelligence*. MIT press, 2001.

[7] F. Silva, M. Duarte, L. Correia, S. M. Oliveira, and A. L. Christensen, "Open issues in evolutionary robotics," *Evolutionary Computation*, vol. 24, no. 2, pp. 205–236, 2016. PMID: 26581015.

[8] M. R. Dawson, *Understanding cognitive science*. Blackwell Oxford, 1998.

[9] "What is cognitive science?." http://www.cs.oswego.edu/~blue/xhx/books/cogsci2/text1/section01/main.html. Accessed: 2020-2-10.

[10] "Discriminative stimuli | in chapter 05: Conditioning | from psychology: An introduction by russ dewey." https://www.psywww.com/intropsych/ch05-conditioning/stimulus-control.html#discriminative. Accessed: 2020-2-10.

[11] D. McColl, A. Hong, N. Hatakeyama, G. Nejat, and B. Benhabib, "A survey of autonomous human affect detection methods for social robots engaged in natural hri," *Journal of Intelligent & Robotic Systems*, vol. 82, no. 1, pp. 101–133, 2016.

[12] S. Spaulding and C. Breazeal, "Frustratingly easy personalization for real-time affect interpretation of facial expression," in *2019 8th International Conference on Affective Computing and Intelligent Interaction (ACII)*, pp. 531–537, IEEE, 2019.

[13] H. W. Park, I. Grover, S. Spaulding, L. Gomez, and C. Breazeal, "A model-free affective reinforcement learning approach to personalization of an autonomous social robot companion for early literacy education," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 687–694, 2019.

[14] F. Schrodt, J. Kneissler, S. Ehrenfeld, and M. V. Butz, "Mario becomes cognitive," *Topics in cognitive science*, vol. 9, no. 2, pp. 343–373, 2017.

[15] E. Jacobs, J. Broekens, and C. Jonker, "Emergent dynamics of joy, distress, hope and fear in reinforcement learning agents," in *Adaptive learning agents workshop at AAMAS2014*, 2014.

[16] T. M. Moerland, J. Broekens, and C. M. Jonker, "Fear and hope emerge from anticipation in model-based reinforcement learning.," in *IJCAI*, pp. 848–854, 2016.

[17] H. Williams, C. Lee-Johnson, W. N. Browne, and D. A. Carnegie, "Emotion inspired adaptive robotic path planning," in *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pp. 3004–3011, IEEE, 2015.

[18] H. Williams, *Human inspired robotic path planning and heterogeneous robotic mapping*. PhD dissertation, Victoria University of Wellington, 2016.

[19] T. M. Moerland, J. Broekens, and C. M. Jonker, "Emotion in reinforcement learning agents and robots: a survey," *Machine Learning*, vol. 107, no. 2, pp. 443–480, 2018.

[20] J. Panksepp, "Affective consciousness: Core emotional feelings in animals and humans," *Consciousness and cognition*, vol. 14, no. 1, pp. 30–80, 2005.

[21] I. Kotseruba, O. J. A. Gonzalez, and J. K. Tsotsos, "A review of 40 years of cognitive architecture research: Focus on perception, attention, learning and applications," *arXiv preprint arXiv:1610.08602*, pp. 1–74, 2016.

[22] J. E. Laird, "Extending the soar cognitive architecture," *Frontiers in Artificial Intelligence and Applications*, vol. 171, p. 224, 2008.

[23] P. Langley, J. E. Laird, and S. Rogers, "Cognitive architectures: Research issues and challenges," *Cognitive Systems Research*, vol. 10, no. 2, pp. 141–160, 2009.

[24] M. Schlesinger and B. McMurray, "The past, present, and future of computational models of cognitive development," *Cognitive Development*, vol. 4, no. 27, pp. 326–348, 2012.

[25] D. Klahr, P. Langley, R. Neches, and R. T. Neches, *Production system models of learning and development*. MIT press, 1987.

[26] S. Nason and J. E. Laird, "Soar-rl: integrating reinforcement learning with soar," *Cognitive Systems Research*, vol. 6, no. 1, pp. 51 – 59, 2005. Special Issue of Cognitive Systems Research - The Best Papers from ICCM2004.

[27] R. L. Lewis, "An architecturally-based theory of human sentence comprehension," tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1993.

[28] F. Bellas, R. J. Duro, A. Faiña, and D. Souto, "Multilevel darwinist brain (mdb): Artificial evolution in a cognitive architecture for real robots," *IEEE Transactions on autonomous mental development*, vol. 2, no. 4, pp. 340–354, 2010.

[29] F. Bellas, A. Faiña, G. Varela, and R. J. Duro, "A cognitive developmental robotics architecture for lifelong learning by evolution in real robots," in *The*

2010 International Joint Conference on Neural Networks (IJCNN), pp. 1–8, IEEE, 2010.

[30] J. P. Borst and J. R. Anderson, "A step-by-step tutorial on using the cognitive architecture act-r in combination with fmri data," *Journal of Mathematical Psychology*, vol. 76, pp. 94 – 103, 2017. Model-based Cognitive Neuroscience.

[31] J. R. Anderson, *How can the human mind occur in the physical universe?* Oxford University Press, 2009.

[32] 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, no. 4, p. 1036, 2004.

[33] C. L. Dancy, "Why the change of heart? understanding the interactions between physiology, affect, and cognition and their effects on decision-making," 2014.

[34] J. R. Anderson, *Rules of the mind*. Psychology Press, 2014.

[35] N. Taatgen, "Modeling parallelization and flexibility improvements in skill acquisition: From dual tasks to complex dynamic skills," *Cognitive Science*, vol. 29, no. 3, pp. 421–455, 2005.

[36] J. Panksepp, "Empathy and the laws of affect," *Science*, vol. 334, no. 6061, pp. 1358–1359, 2011.

[37] W. Mischel and Y. Shoda, "A cognitive-affective system theory of personality: reconceptualizing situations, dispositions, dynamics, and invariance in personality structure.," *Psychological review*, vol. 102, no. 2, p. 246, 1995.

[38] D. Kortenkamp, R. Simmons, and D. Brugali, "Robotic systems architectures and programming," in *Springer Handbook of Robotics*, pp. 283–306, Springer, 2016.

[39] E. Rolls, *The Brain and Emotion*. Oxford University Press, 1999.

[40] E. T. Rolls, *Emotion and decision-making explained*. OUP Oxford, 2013.

[41] E. T. Rolls and F. Grabenhorst, "The orbitofrontal cortex and beyond: from affect to decision-making," *Progress in neurobiology*, vol. 86, no. 3, pp. 216–244, 2008.

[42] E. T. Rolls, "Limbic systems for emotion and for memory, but no single limbic system," *Cortex*, vol. 62, pp. 119–157, 2015.

[43] J. Panksepp and L. Biven, *The archaeology of mind: Neuroevolutionary origins of human emotions*. WW Norton & Company, 2012.

[44] R. S. Lazarus and C. A. Smith, "Knowledge and appraisal in the cognitionâĂŤemotion relationship," *Cognition & Emotion*, vol. 2, no. 4, pp. 281–300, 1988.

[45] R. G. Parsons and K. J. Ressler, "Implications of memory modulation for post-traumatic stress and fear disorders," *Nature neuroscience*, vol. 16, no. 2, p. 146, 2013.

[46] K. Skelton, K. J. Ressler, S. D. Norrholm, T. Jovanovic, and B. Bradley-Davino, "Ptsd and gene variants: New pathways and new thinking," *Neuropharmacology*, vol. 62, no. 2, pp. 628 – 637, 2012. Post-Traumatic Stress Disorder.

[47] K. R. Scherer, T. Bänziger, and E. Roesch, *A Blueprint for Affective Computing: A sourcebook and manual*. Oxford University Press, 2010.

[48] K. R. Scherer, "Emotions are emergent processes: they require a dynamic computational architecture," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 364, no. 1535, pp. 3459–3474, 2009.

[49] D. Sander and K. Scherer, *Oxford companion to emotion and the affective sciences*. OUP Oxford, 2014.

[50] J. Panksepp, "Affective consciousness: Core emotional feelings in animals and humans," *Consciousness and Cognition*, vol. 14, no. 1, pp. 30 – 80, 2005. Neurobiology of Animal Consciousness.

[51] J. Panksepp, T. Fuchs, and P. Iacobucci, "The basic neuroscience of emotional experiences in mammals: The case of subcortical fear circuitry and implications for clinical anxiety," *Applied Animal Behaviour Science*, vol. 129, no. 1, pp. 1 – 17, 2011.

[52] "Apa dictionary of action readiness." `https://dictionary.apa.org/action-readiness`. Accessed: 2020-2-10.

[53] E. T. Rolls, "What are emotional states, and why do we have them?," *Emotion review*, vol. 5, no. 3, pp. 241–247, 2013.

[54] "Discriminative stimuli | in chapter 05: Conditioning | from psychology: An introduction by russ dewey." `http://www.intropsych.com/ch05_conditioning/discriminative_stimuli.html`. Accessed: 2020-2-10.

[55] R. Brooks, "A robust layered control system for a mobile robot," *IEEE journal on robotics and automation*, vol. 2, no. 1, pp. 14–23, 1986.

[56] R. C. Arkin and T. Balch, "Aura: Principles and practice in review," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 9, no. 2-3, pp. 175–189, 1997.

[57] B. Siciliano and O. Khatib, *Springer handbook of robotics*. Springer, 2016.

[58] M. MendonÃğa, B. Angelico, L. Arruda, and F. Neves, "A dynamic fuzzy cognitive map applied to chemical process supervision," *Engineering Applications of Artificial Intelligence*, vol. 26, no. 4, pp. 1199 – 1210, 2013.

[59] L. V. R. Arruda, M. Mendonça, F. Neves, I. R. Chrun, and E. I. Papageorgiou, "Artificial life environment modeled by dynamic fuzzy cognitive maps," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 10, no. 1, pp. 88–101, 2016.

[60] P. Domingos, *The master algorithm: How the quest for the ultimate learning machine will remake our world*. Basic Books, 2015.

[61] A. Pétrowski and S. Ben-Hamida, *Evolutionary algorithms*. Wiley Online Library, 2017.

[62] T. Back, *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[63] T. Back and H.-P. Schwefel, "An overview of evolutionary algorithms for parameter optimization," *Evolutionary Computation*, vol. 1, no. 1, pp. 1–23, 1993.

[64] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[65] R. J. Urbanowicz and W. N. Browne, *Introduction to learning classifier systems*. Springer, 2017.

[66] M. Iqbal, W. N. Browne, and M. Zhang, "Evolving optimum populations with xcs classifier systems," *Soft Computing*, vol. 17, no. 3, pp. 503–518, 2013.

[67] T. A. M. O'Hara, *Learning Classifier Systems with Neural Network Representation*. PhD thesis, Citeseer, 2006.

[68] L. Bull and T. O'Hara, "Accuracy-based neuro and neuro-fuzzy classifier systems," in *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pp. 905–911, Citeseer, 2002.

[69] S. W. Wilson, "Classifier fitness based on accuracy," *Evolutionary computation*, vol. 3, no. 2, pp. 149–175, 1995.

[70] T. Kovacs, *Strength or accuracy: credit assignment in learning classifier systems*. Springer Science & Business Media, 2012.

[71] M. Roozegar, M. Mahjoob, M. Esfandyari, and M. S. Panahi, "Xcs-based reinforcement learning algorithm for motion planning of a spherical mobile robot," *Applied Intelligence*, vol. 45, no. 3, pp. 736–746, 2016.

[72] P. L. Lanzi, "Learning classifier systems from a reinforcement learning perspective," *Soft Computing*, vol. 6, no. 3-4, pp. 162–170, 2002.

[73] E. Bernadó-Mansilla and J. M. Garrell-Guiu, "Accuracy-based learning classifier systems: models, analysis and applications to classification tasks," *Evolutionary computation*, vol. 11, no. 3, pp. 209–238, 2003.

[74] A. Orriols and E. Bernado-Mansilla, "Class imbalance problem in ucs classifier system: fitness adaptation," in *2005 IEEE Congress on Evolutionary Computation*, vol. 1, pp. 604–611 Vol.1, 2005.

[75] T. Ebadi, M. Zhang, and W. Browne, "Xcs-based versus ucs-based feature pattern classification system," in *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, pp. 839–846, 2012.

[76] M. V. Butz and S. W. Wilson, "An algorithmic description of xcs," *Soft Computing*, vol. 6, no. 3-4, pp. 144–153, 2002.

[77] J. Hurst, L. Bull, and C. Melhuish, "Tcs learning classifier system controller on a real robot," in *International Conference on Parallel Problem Solving from Nature*, pp. 588–597, Springer, 2002.

[78] M. Studley and L. Bull, "X-tcs: accuracy-based learning classifier system robotics," in *2005 IEEE Congress on Evolutionary Computation*, vol. 3, pp. 2099–2106, IEEE, 2005.

[79] J. Hurst, L. Bull, and C. Melhuish, "Zcs and tcs learning classifier system controllers on real robots," *UWE Learning Classifier Systems Group Technical Report 02*, vol. 2, 2002.

[80] G. D. Howard, L. Bull, and P.-L. Lanzi, "Towards continuous actions in continuous space and time using self-adaptive constructivism in neural xcsf," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pp. 1219–1226, 2009.

[81] P. L. LANZI *et al.*, "A cognitive architecture based on a learning classifier system with spiking classifiers," 2015.

[82] P. O. Stalph and M. V. Butz, "Learning local linear jacobians for flexible and adaptive robot arm control," *Genetic programming and evolvable machines*, vol. 13, no. 2, pp. 137–157, 2012.

[83] J. Kneissler, P. O. Stalph, J. Drugowitsch, and M. V. Butz, "Filtering sensory information with xcsf: improving learning robustness and robot arm control performance," *Evolutionary computation*, vol. 22, no. 1, pp. 139–158, 2014.

[84] S. W. Wilson, "Classifiers that approximate functions," *Natural Computing*, vol. 1, no. 2-3, pp. 211–234, 2002.

[85] O. Unold, E. Rogula, and N. Kozłowski, "Introducing action planning to the anticipatory classifier system acs2," in *International Conference on Computer Recognition Systems*, pp. 264–275, Springer, 2019.

[86] N. Kozlowski and O. Unold, "Investigating exploration techniques for acs in discretized real-valued environments," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, pp. 1765–1773, 2020.

[87] W. Stolzmann and M. Butz, "Latent learning and action planning in robots with anticipatory classifier systems," in *International Workshop on Learning Classifier Systems*, pp. 301–317, Springer, 1999.

[88] "Pioneer 3 operations manual." `https://www.inf.ufrgs.br/~prestes/Courses/Robotics/manual_pioneer.pdf`. Accessed: 2020-2-10.

[89] "Pioneer3dx-p3dx-reva." `https://www.generationrobots.com/media/Pioneer3DX-P3DX-RevA.pdf`. Accessed: 2020-2-10.

[90] "about ros." `https://www.ros.org/about-ros/`. Accessed: 2020-2-10.

[91] "Ros packages." `http://wiki.ros.org/Packages`. Accessed: 2020-2-10.

[92] J. A. Russell, "Core affect and the psychological construction of emotion.," *Psychological review*, vol. 110, no. 1, p. 145, 2003.

[93] R. Lowe, "The feeling of action tendencies: on the emotional regulation of goal-directed behavior," *Frontiers in psychology*, vol. 2, p. 346, 2011.

[94] J. Panksepp, T. Fuchs, and P. Iacobucci, "The basic neuroscience of emotional experiences in mammals: The case of subcortical fear circuitry and implications for clinical anxiety," *Applied Animal Behaviour Science*, vol. 129, no. 1, pp. 1–17, 2011.

[95] R. Feldman, "Maternal-infant contact and child development: Insights from the kangaroo intervention," *Low-Cost Approaches to Promote Physical and Mental Health*, pp. 323–351, 2007.

[96] R. E. Grunau, "Early pain in preterm infants: a model of long-term effects," *Clinics in perinatology*, vol. 29, no. 3, pp. 373–394, 2002.

[97] J. Walinga and C. Stangor, "Introduction to psychology-1st canadian edition," 2014.

[98] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: An astounding baseline for recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2014.

[99] S. D'Mello and A. Graesser, "The half-life of cognitive-affective states during complex learning," *Cognition & Emotion*, vol. 25, no. 7, pp. 1299–1308, 2011.

[100] J. A. Russell and L. F. Barrett, "Core affect, prototypical emotional episodes, and other things called emotion: dissecting the elephant.," *Journal of personality and social psychology*, vol. 76, no. 5, p. 805, 1999.

[101] E. T. Rolls, *The brain, emotion, and depression*. Oxford University Press, 2018.

[102] A. Amsel, "Frustration theory: Many years later.," *Psychological bulletin*, vol. 112, no. 3, p. 396, 1992.

[103] J. Breuer and M. Elson, "Frustration–aggression theory," *The Wiley handbook of violence and aggression*, pp. 1–12, 2017.

[104] I. M. Alvarez, W. N. Browne, and M. Zhang, "Human-inspired scaling in learning classifier systems: Case study on the n-bit multiplexer problem set," in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, Denver, CO, USA, July 20 - 24, 2016* (T. Friedrich, F. Neumann, and A. M. Sutton, eds.), pp. 429–436, ACM, 2016.

[105] M. V. Butz, T. Kovacs, P. L. Lanzi, and S. W. Wilson, "Toward a theory of generalization and learning in XCS," *IEEE Trans. Evolutionary Computation*, vol. 8, no. 1, pp. 28–46, 2004.

[106] T. Kovacs and M. Kerber, "What makes a problem hard for xcs?," in *International Workshop on Learning Classifier Systems*, pp. 80–99, Springer, 2000.

[107] T. Kovacs, "Towards a theory of strong overgeneral classifiers," in *Foundations of Genetic Algorithms 6*, pp. 165–184, Elsevier, 2001.

[108] S. D. Whitehead and L.-J. Lin, "Reinforcement learning of non-markov decision processes," *Artificial Intelligence*, vol. 73, no. 1-2, pp. 271–306, 1995.

[109] P. L. Lanzi, "An analysis of generalization in the xcs classifier system," *Evolutionary Computation*, vol. 7, no. 2, pp. 125–149, 1999.

[110] M. Nakata, P. L. Lanzi, and K. Takadama, "Xcs with adaptive action mapping," in *Proceedings of the 9th international conference on Simulated Evolution and Learning*, pp. 138–147, Springer-Verlag, 2012.

[111] O. Nachum, M. Ahn, H. Ponte, S. Gu, and V. Kumar, "Multi-agent manipulation via locomotion using hierarchical sim2real," *arXiv preprint arXiv:1908.05224*, 2019.

[112] I. Mordatch, N. Mishra, C. Eppner, and P. Abbeel, "Combining model-based policy search with online model learning for control of physical humanoids," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 242–248, IEEE, 2016.

[113] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 12627–12637, 2019.

[114] "Ros dynamic reconfigure." `http://wiki.ros.org/dynamic_reconfigure`. Accessed: 2020-2-10.

[115] X. Cheng, W. N. Browne, and M. Zhang, "Decomposition based multi-objective evolutionary algorithm in xcs for multi-objective reinforcement learning," in *2018 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8, IEEE, 2018.