ACTION EXECUTION, ITS ESTIMATION AND LEARNING FOR A SYSTEMS LEVEL COGNITIVE ARCHITECTURE

by

Daqi Dong

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

Major: Computer Science

The University of Memphis

August, 2016

Acknowledgments

I am grateful for the generous support of the Faculty Research Grants (2011) of the University of Memphis, without which this thesis would not have been possible.

I wish to thank my academic advisor, Dr. Stan Franklin. His daily guidance has considerably added to my graduate experience; his trust and confidence have continually supported my academic explorations; and the example of his remarkable work consistently inspires me to pursue excellence. I would also like to thank the other members of my committee, Dr. Dipankar Dasgupta, Dr. Vinhthuy Phan, and Dr. Vasile Rus, for the directions they have provided both to my study and to this thesis.

The same gratitude goes out to my colleagues who have worked and continue to work in our Cognitive Computing Research Group. This thesis is has grown out of our group's collective work. Specific thanks go to Steve Strain, who provided edits, comments, and suggestions to the thesis.

Significant acknowledgement is also due the Department of Computer Science, the Institute for Intelligent Systems, and the University of Memphis where I am grateful for the opportunity to study. These institutions offer the highest quality support to students pursuing high-level development as academic researchers.

Finally, great thanks to my family for their love and support throughout my life, and especially for their encouragement of my academic pursuits.

Abstract

Daqi Dong. Ph.D The University of Memphis. August/2016. Action Execution, Its Estimation and Learning for a Systems Level Cognitive Architecture. Major Professor: Stanley P. Franklin.

An agent or robot achieves its goals by interacting with its environment, cyclically choosing and executing suitable actions. Cognitive architectures are considered the control structures of the agent, helping it decide what to do next, while the designs resemble how minds work, be they human, animal, or artificial.

An action execution process is a critical part of an entire cognitive architecture, because the process of generating executable motor commands is not only driven by low-level environmental information, but is also initiated and affected by the agent's high-level mental processes. I give a review of the cognitive models of the action execution process as implemented in a set of popular cognitive architectures, and conclude with some general observations regarding the nature of action execution.

Next, I present a cognitive model—the Sensory Motor System (SMS)—for an action execution process, as a new module of the LIDA (for "Learning Intelligent Distribution Agent") systems-level cognitive model. A sensorimotor system derived from the subsumption architecture has been implemented into the SMS; and several cognitive neuroscience hypotheses have been incorporated as well.

Inspired by the hypothesis that humans estimate their movements based on their knowledge of the dynamics of the environment, and on actual sensory data (Wolpert, Ghahramani, & Jordan, 1995), I create a model of the estimation process of action execution using SMS in LIDA. Also, based on a recent study in neuroscience (Herzfeld, Vaswani, Marko,

& Shadmehr, 2014), I introduce a new factor—memory of errors—into this model of estimation. The historical errors help humans determine the stability of the environment, so as to decide the degree to which knowledge of the environment may affect the estimation.

Learning is significant for for allowing an agent to act more intelligently. I present a new model of sensorimotor learning in LIDA, one that helps an agent properly interact with its environment using past experiences. Following Global Workspace Theory, the primary basis of LIDA, this learning is cued by the agent's conscious content, the most salient portion of the agent's understanding of the current situation. Furthermore, I add a dynamic learning rate to control the extent to which newly arriving conscious content may affect the learning.

Finally, I introduce an extension of the SMS. This extension allows, and explains, the use of the sensory data, the prime, perceived before a participant starts his or her movement, by the SMS during action execution. Furthermore, this extension allows the replication by a LIDA-based agent, of some human experiments (T. Schmidt, 2002) studying the priming process in motor control.

Table of Contents

List of Tables

List of Figures

1. Introduction

Humans seem very curious about their bodies, their minds, and especially their intelligence. In the field of artificial intelligence (AI), the original aim was to reproduce human-level intelligence. In robotics, people would like to create human-like robots. In addition to the hardware bodies, a robot does need a controller determining what to do next. A human-level controller allows the robot to be human-like.

I consider the robot that owns both the body and the controller to be an agent. Furthermore, I require the agent to be autonomous. An autonomous agent acts independently in its environment with an agenda, and over time its actions may affect what it senses in the future (Franklin & Graesser, 1997). A general question motivating my research activities is "how do minds work, be they human, animal, or artificial?" Following the LIDA Model, I define a mind as a control structure for an autonomous agent (Franklin, Madl, D'Mello, & Snaider, 2014). In my work, cognitive architectures are used as a concrete tool to explore cognitive representations and processes of minds, from the perspective of computer science, the data structures and algorithms of control structures.

Action and action execution

Action plays an essential role in creating a human-like agent: The agent interacts with its environment by acting to achieve its goals. But how does that an action happen? For example, when I am doing my daily driving, I know I am driving but I do not know exactly what I am doing at every moment. How about the force applied by my fingers to the steering wheel, the oxygen-level in my blood, or my mental states? I am not aware of much of that either during the driving or afterwards. I only remember some "screen shots" of the driving after I have arrived. It is interesting to see that humans do not know much about how their actions have been done, in

the cases of driving, swimming, or even gripping a cup, while they do know that they can do these things and, at a relatively abstract level, what they are doing.

From different fields of study such as psychology, neuroscience, and cognitive science, researchers have provided evidence and formulated hypotheses to explain how human action works. Marc Jeannerod, citing the work of Searle (1983), built upon the concept that covert action representation is followed by overt, real execution of action. In detail, "…the conceptual content, when it exists (i.e., when an explicit desire to perform the action is formed), is present first. Then, at the time of execution, a different mechanism comes into play where the representation loses its explicit character and runs automatically to reach the desired goal" (Jeannerod, 2006, pp. 4-5). Jeannerod suggests that action representation (preparation) and action execution are two different processes. A similar idea of distinguishing action execution from action preparation (selection) is proposed by Milner and Goodale as well. In their work on the two visual systems (1992; 2008), they proposed two cortical systems, the ventral and dorsal streams, providing "vision for perception" and "vision for action" respectively. Regarding the roles of the two streams in the guidance of action, the perceptual mechanism in the ventral stream identifies a goal object, and helps to select an appropriate course of action, while the dorsal stream "is critical for the detailed specification and online control of the constituent movements that form the action" (Milner & Goodale, 2008, p. 775). Additional studies regarding human action, especially certain neuroscientific evidence, can be found in a set of review papers (Castiello, 2005; Grafton, 2010; Wolpert, Diedrichsen, & Flanagan, 2011).

Dr. Stan Franklin and I have proposed as well that human action presents two aspects: "what to do" and "how to do it" (2014b). On the one hand, action is driven by the agent's intention. This means the agent selects the action via internal motivation as a result of mental

processes, rather than generating a simple reflex in response to a stimulus. Thus, the agent understands what it will do before the action execution begins. However, this understanding of the action is not executable in the real world, because the needed low-level environmental information is not yet involved; executing an action in the real world requires us to conceive of an agent's action as occurring within its environment (Franklin & Graesser, 1997). On the other hand, the action's execution may not be understandable to the agent, because the environmental elements involved are low-level raw data without explicit meaning, while that which is understandable must have some form of meaning for the agent. As an example, the agent does not directly understand the raw stimulus data retrieved by its sensors from the environment. Rather, the data must be transformed into higher level meaning by a perception process; that is, the transformation produces an understandable representation of the sensed data. Action execution performs a transformation similar to that of perception, but in reverse: converts an understandable action into low-level movements.

We have further proposed and computationally implemented a new model, the Sensory Motor System (SMS), for how a human maintains one facet of action: "how to do it" (Dong & Franklin, 2015b). The SMS is a cognitive model of the action execution process. Action execution refers to a situation in which an agent executes a selected goal-directed action in the real world so as to produce pertinent movement. The SMS transforms the selected action into an executable low-level action sequence, a sequence of motor commands, and executes them through appropriate use of the agent's actuators in the environment. This transformation is assisted by the sensory data perceived online.

One important data structure used in the SMS is the motor plan. A set of motor commands is prepared inside a motor plan, and the plan generates motor commands in an order

driven by the arrival of sensory data. Our motor plan is implemented based on the subsumption architecture (Brooks, 1986, 1991), which behaves like a reactive structure that passively produces output upon the arrival of input. The subsumption architecture fulfills the required features of action execution as we model it, including (1) the bottom-up sensory data directly driving the executable action, (2) the decomposition from an understandable action to executable motor commands, and (3) the absence of an understandable action's "explicit character" as mentioned by Jeannerod above. But on the other hand, the subsumption architecture does not reflect the process of specification for the movement parameters nor does it interact with high-level goal-directed actions. We have implemented these in our SMS, an important extension of the subsumption architecture. I introduce the details of the subsumption architecture, its extension, and the fundamental concepts of the SMS in Chapter 4.

Action execution for cognitive architectures

For the last several decades, due to the difficulty of achieving human level intelligence, the majority of AI researchers have focused on what has been called "narrow AI", where the AI system is highly constrained to specific tasks. But recently, a movement in AI research called artificial general intelligence (AGI) has been initiated (Goertzel & Pennachin, 2007; Wang, Goertzel, & Franklin, 2008). It aims to return to the original goal of AI, to construct computer systems with human-like general intelligence. AGI research treats intelligence as a whole; it carries out the engineering practice according to an outline of a system comparable to the human mind in a certain sense. A parallel movement appeared a little later under the rubric of BICA (Biologically Inspired Cognitive Architectures) as well. BICA focuses on the integration of various research efforts from different disciplines to address the challenge of creating a

computational equivalent of the human mind. Both AGI and BICA address their AI dreams by approaching things at a systems level, and proceeding to model the human mind.

Actually, the use of systems level cognitive architectures has been championed by several researchers in the past as well. Artificial intelligence pioneer Allen Newell strongly supported the need for systems-level theories and architectures, claiming that "You can't play 20 questions with nature and win" (1973). Langley, Laird, and Rogers (2009) argued as follows: "Instead of carrying out micro-studies that address only one issue at a time, we should attempt to unify many findings into a single theoretical framework, then proceed to test and refine that theory." They are calling for a broad-based, systems-level architecture.

Cognitive architectures are designed to be the basis for creating autonomous agents that can solve a wide variety of problems using a wide range of knowledge; they define and organize "the primitive computational structures that store, retrieve, and process knowledge" to pursue the agent's goals (J. Laird, 2012). A collection of cognitive architectures has been reviewed in recent studies (Duch, Oentaryo, & Pasquier, 2008; Goertzel et al., 2010; P Langley et al., 2009). Regarding actions and their execution processes, a brief summary has been made in the following lines:

> A cognitive architecture must also be able to execute skills and actions in the environment. In some frameworks, this happens in a completely reactive manner, with the agent selecting one or more primitive actions on each decision cycle, executing them, and repeating the process on the next cycle. This approach is associated with closed-loop strategies for execution, since the agent can also sense the environment on each time step. The utilization of more complex skills supports open-loop execution, in which the agent calls upon a stored procedure across many cycles without checking the environment. However, a flexible architecture should support the entire continuum from fully reactive, closed-loop behavior to automatized, open-loop behavior, as can humans. (P Langley et al., 2009)

I give a review regarding the action execution process implemented in different cognitive architectures in Chapter 3.

In our work, not only do we model action execution itself, we have also addressed the relationship between action execution and other cognitive processes. We have developed the Sensory Motor System (SMS) as a new module for a systems level cognitive architecture, LIDA[1]. LIDA is a conceptual, systems level model of human mental processes. It had integrated perception, attention, and action (selection) previously (Franklin et al., 2014), and now we have added the SMS to fulfill its action execution part (Dong & Franklin, 2015b). In the current LIDA, its Sensory Memory provides sensory data to drive the process of action execution implemented by the SMS, while LIDA's Action Selection module provides the selected goal-directed action (the selected behavior in LIDA) to the SMS to execute. I describe the details of LIDA in Chapter 2.

Furthermore, we have implemented *estimation* and *learning* of action execution in LIDA (Dong & Franklin, 2015a; Dong, Franklin, & Agrawal, 2015).

Humans estimate their movements based on both their knowledge of the dynamics of the environment and actual sensory data (Wolpert & Ghahramani, 2000; Wolpert et al., 1995). Wolpert and colleagues have incorporated this understanding into a model that simulates this estimation using the Kalman filter (Kalman, 1960). Inspired by their work, we have modeled the *estimation* process embedded within action execution in LIDA (Dong et al., 2015). An internal model has been added into the SMS and the Kalman Filter has been extended using the idea of memory of errors (Herzfeld et al., 2014) for estimating action effects. I introduce this estimation work in Chapter 5.

[1] For historical reasons LIDA stands for Learning Intelligent Distribution Agent.

In LIDA, by the competitive process specified in Global Workspace Theory (Baars, 1988, 2002), a LIDA-based agent decides what portion of the perceived present situation should be attended to and broadcast to the rest of the system to modulate learning (Franklin et al., 2014). Particularly, this attended present situation, called the current conscious content, is broadcast to the SMS to assist its sensory motor learning (Dong & Franklin, 2015a). I introduce learning in SMS in Chapter 6.

We have created different LIDA-based agents using different software robots, such as youBot and a two-wheel robot, to implement our SMS. A software environment, Webots, is used in our experiments as well. These computational implementations and experiments have previously verified and improved the capabilities of the models we have created herein. I review them in Chapter 4.

Finally, we have tested the LIDA Model to explain and predict an unconscious priming effect on motor control as reported from a human experiment (T. Schmidt, 2002). The model failed in both the explanation and the prediction before our improvement. Therefore, we improved (refined) the LIDA Model by extending its Sensory Motor System (SMS) so as to model, and thereby explain the empirical data. A software agent was created using this improved model, which allowed the replication of the empirical data concerning motor priming. I introduce our design of the extended SMS and the relevant computational simulations in Chapter 7.

Contributions of this work

An agent achieves its goals by interacting with its environment, cyclically choosing and executing suitable actions. An action execution process is a reasonable and critical part of an entire cognitive architecture, because the process of generating executable motor commands is

not only driven by low-level environmental information, but is also initiated and affected by the agent's high-level mental processes.

Many cognitive architectures exist, though none, as yet, at human-level intelligence (Samsonovich, 2010). Especially, not very many cognitive architectures consider action execution a standard component: In an online Comparative Table of Implemented Cognitive Architectures[2], from more than two dozen posted architectures, we find that less than half of them have implemented action execution in a relatively complete form (Dong & Franklin, 2014a).

Even of those cognitive architectures in which action execution has been implemented, none has fully addressed the features of action execution discussed above. For example, in the Adaptive Control of Thought-Rational (ACT-R) architecture (*ACT-R 6.0 Tutorial*, 2012), "there is typically no direct communication between the perceptual and motor modules.[3] The data passed to the motor module always comes from the high-level declarative memory." (Dong & Franklin, 2015b). And in Soar (J. Laird, 2008, 2012), although it uses the term "motor commands" to describe its final output data, these commands cannot be directly executed in the external world. Soar only transforms selected high-level actions into general low-level actions, and an external program is always necessary to handle their final "real" execution. I give a detailed discussion regarding action execution implemented in different cognitive architectures in Chapter 3.

In our work, we have designed the Sensory Motor System (SMS) as a sub module of the systems level cognitive model LIDA, thereby rendering it capable of communicating with other cognitive modules naturally in a closed cognitive loop, from sensors to actions. The design of

[2] It is available at http://bicasociety.org/cogarch/

[3] There is only very limited direct connectivity between perceptual and motor modules. Spatial information in particular is communicated directly.

LIDA is biologically inspired, aiming to model human-level minds, and the SMS follows LIDA's design philosophy, pursuing the model of human-level action execution. The features of action execution introduced above have been well covered in our addition of the SMS into LIDA. Furthermore, many new hypotheses and understandings regarding human minds and action execution have been involved.

Here we provide a way to explore a specific cognitive module, action execution, and its relationship with other relevant cognitive modules. This allows us to further explore how a mind works, especially regarding its action execution part.

Specifically in the design of SMS, we have considered the subsumption architecture from a new viewpoint, namely, that its capabilities fulfill the hypothesis regarding the online control role of the dorsal stream. Also, we have modified the original subsumption architecture as inspired by certain hypotheses from cognitive neuroscience so as to combine a reactive structure with a goal-directed action.

We have modeled two distinct cognitive processes occurring in action execution, *estimation* and *learning*. This makes the SMS behave in a manner closer to human-level action execution. Also, the concept of memory of errors has been borrowed from recent studies in neuroscience (Herzfeld et al., 2014) and applied into our models to improve their similarities to certain human behaviors and their computational performance. I give full descriptions of the estimation, learning, and memory of errors in Chapters 5 and 6.

We have published our work in peer-viewed journals. Some of the papers are incorporated in the literature review (Chapter 3) and the three Chapters (4-6) reporting results.[4]

[4] In all publications, Dong proposed the original idea, wrote the draft of paper, designed the experiment, developed the software, and analyzed the data. Ideas provided by Franklin and Agrawal were reflected into the final version by Dong after the discussion with these co-authors. All publications were supervised by Franklin.

Also, we have submitted a new paper that is incorporated in Chapter 7.[5] I list these papers and their summarized contributions just below, presented as a summary of my research contributions to Computer Science and Cognitive Modeling:

- Chapter 3: Dong, D., & Franklin, S. (2014). The Action Execution Process Implemented in Different Cognitive Architectures: A Review. Journal of Artificial General Intelligence, 5(1), 47-66.

Contributions: (1) A review of the action execution process as implemented in different cognitive architectures. The common characteristics of action execution were identified, and its comprehensive representations and functional procedures were summarized.

- Chapter 4: Dong, D., & Franklin, S. (2015). A New Action Execution Module for the Learning Intelligent Distribution Agent (LIDA): The Sensory Motor System. Cognitive Computation, 7(5), 552-568.

Contributions: (2) The completion of a systems level cognitive architecture, LIDA, by fulfilling its action execution part, particularly by newly designing and implementing the Sensory Motor System (SMS). The design is both biologically inspired and computationally implementable. (3) The subsumption architecture has been newly applied in cognitive modeling, and has been extended with variables dynamically specifiable at runtime. (4) The execution of a grip action has been implemented using a LIDA-based agent incorporating the SMS. A software robot, youBot, and a simulated environment, Webots, were configured and involved as well. This grip implementation allows experimental verifications of the models.

[5] In this paper, Dong provided the design of the extended SMS, developed its software simulation, and wrote the relevant sections. Agrawal prepared the experimental environments, replicated the human experimental results using the extended SMS, and wrote the rest of the paper. All work was supervised by Franklin.

- Chapter 5: Dong, D., Franklin, S., & Agrawal, P. (2015). Estimating Human Movements Using Memory of Errors. Procedia Computer Science, 71, 1-10.

Contributions: (5) An internal model has been implemented in the SMS to produce an estimation of the effect of action execution. The estimated result was determined by comparing it with experimental results of humans. (6) The original implementation of the internal model (Wolpert & Ghahramani, 2000; Wolpert et al., 1995) was novelly extended with the concept of memory of errors (Herzfeld et al., 2014), resulting in improved similarity to humans in our model.

- Chapter 6: Dong, D., & Franklin, S. (2015). Modeling Sensorimotor Learning in LIDA Using a Dynamic Learning Rate. Biologically Inspired Cognitive Architectures, 14, 1-9.

Contributions: (7) Sensory motor learning has been implemented in LIDA following global Workspace Theory (Baars, 1988, 2002). This is the second implementation of learning in LIDA, the first being the modeling of attentional learning by Faghihi and colleagues (2012). (8) A dynamic learning rate has been added into the reward updating process of the learning. We have set up a software simulated experiment, where a LIDA-based agent looks for boxes and pushes them around. Better execution of the push action has been obtained by using sensory motor learning with the dynamic learning rate.

- Chapter 7: Agrawal, P., Dong, D., & Franklin, S. (submitted in 2016). Modeling Motor Priming in a Systems Level Cognitive architecture. Cognitive Science.

Contributions: (8) Empirical data concerning unconscious motor priming has been explained by an extension to the LIDA Model. In particular, LIDA's Sensory Motor System

(SMS) has been extended so as to model, and thereby explain the data. A software agent is created based on the improved LIDA incorporating the extended SMS, which allowed the replication of the empirical data concerning motor priming.

Structure of this work

The rest of this dissertation is organized as follows: Chapter 2 introduces the context of our work and the LIDA Model, especially its action part. Chapter 3 presents a review regarding a set of cognitive architectures that have implemented the action execution process. The fundamental concept of the Sensory Motor System (SMS) of LIDA, a model of action execution for a systems level cognitive architecture, is introduced in Chapter 4. I introduce the models for estimation and learning of action execution, using the SMS in LIDA, separately in Chapter 5 and 6. In Chapter 7, I describe the model of priming of action execution. Chapter 8 discusses the conclusions, and outlines directions for future research.

2. Background and Context

We are working on cognitive modeling of human mental processes and relevant behaviors. This area of computer science also has interdisciplinary ramifications, including implications for psychology, neuroscience, cognitive science, and others.

The simulation of real-world human behavior provides an opportunity to create robots that mimic different classes of movement, as well as the relationship between human movement and the physical world. Physics offers rich empirical studies regarding these motions. The simulation of the human mind, modeling the various internal processes and representations of human cognition, naturally directs us to the field of psychology. An emphasis on the modeling of human action execution hybridizes two complementary perspectives. From one point of view, humans execute actions using actuators that produce physical movement of body parts; from another, the action is an output of human mental activities initiated internally. This leads us to the field of cognitive neuroscience.

Our approaches resemble those of traditional computer science. We have certain requirements to satisfy and specific computational problems to resolve; we design and implement appropriate data structures and algorithms (architectures); we test our results from different levels and viewpoints as when releasing a software product. However, distinctly, since the similarity between the subjects of study and their simulations is an important criterion for the evaluation of a model, we borrow hypotheses from other disciplines regarding system requirements, as well as replicate the experiments of such studies using our cognitive models, and compare simulated results to human data as a means of model verification.

Although we neither study nor experiment on humans directly, a cognitive model of the human mind—more specifically, a computational implementation—provides many potential supports for human activities such as education, health care, or entertainment. It is similar to

computational simulations of physical or social phenomena, for instance, meteorological, cosmological or economic models.

The LIDA Model

The LIDA Model is a systems level cognitive model (Franklin et al., 2016). It implements and fleshes out a number of psychological and neuropsychological theories, but is primarily based on Global Workspace Theory (Baars, 1988, 2002). The model is grounded in the LIDA cognitive cycle (see Figure 1). The simulated human mind can be viewed as functioning via a continual, overlapping sequence of these cycles. Each cognitive cycle consists of three phases: 1) the LIDA agent first senses the environment, recognizes objects, and builds its understanding of the current situation; 2) by a competitive process, as specified by Global Workspace Theory (Baars, 1988, 2002), it then decides what portion of the represented situation should be attended to and broadcasted to the rest of the system; 3) finally, the broadcasted portion of the situation supplies information allowing the agent to choose an appropriate action to execute, and modulates learning.

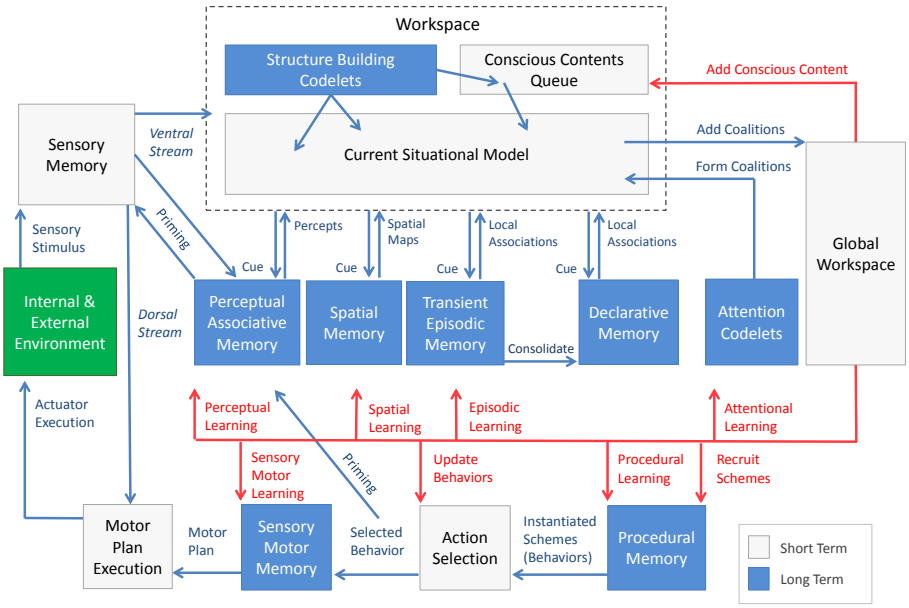


Figure 1. LIDA Cognitive Cycle Diagram

Figure 1 gives an intuitive feel for the relationship among different modules. These LIDA modules are introduced below ordered according to the three phases of the LIDA cognitive cycle: understanding, attention, and action/learning. I describe them in a linear way in order to make the process more easily understood. But actually, each LIDA module acts independently and asynchronously with other modules in LIDA. I introduce more about action execution part in *action/learning* phase while the details of other modules can be found in (Franklin et al., 2016).

Understanding phase

The incoming stimuli are sensed by the agent's sensors from the external and internal environment. Sensory Memory (SM) gets these sensory data (stimuli) and a set of low-level features of the stimuli are made out of them. These low-level features are passed to Perceptual Associative Memory (PAM) where higher-level features, such as objects, events, categories, actions, feelings, etc. are recognized. These recognized entities make up the percept that passes to the Workspace, where the agent's perceived current situation is updated in the Current Situational Model (CSM). The percept serves as a cue to Spatial Memory, Transient Episodic Memory, and Declarative Memory, to recall the remembered contents from these memory systems that were associated with the elements of the cue.

Structure building codelets build high-level and more abstract understanding of the current situation. These codelets are small, special purpose programs, each of which has some particular type of structure it is designed to build. The Conscious Contents Queue stores a few past conscious contents that helps the agent to update its understanding of current situation as well.

Attention phase

Attention Codelets are also special purpose programs that are concerned with certain portions of the content maintained in CSM, and form them into coalitions. The codelets bring these coalitions into the Global Workspace (GW) to compete to have the agent's attention. The winner of the competition, the most salient coalition, has its content become the so-called contents of the consciousness (Baars, 1988), to which the agent selectively attends in the current cognitive cycle. The contents of consciousness issued by GW are broadcast to the rest of the system.

Action/Learning phase

The broadcast conscious contents helps (1) agent decide what to do next and (2) for modulating different modes of learning.

Procedural Memory stores schemes, the templates of possible actions including their contexts and expected results (Drescher, 1991). It also stores an activation value that attempts to measure, for each such scheme, the likelihood that an action taken within its context produces the expected result. The schemes whose contexts and results match the broadcast conscious content well are instantiated into behaviors, which are the instances of the templates with their variables instantiated to the current situation.

These behaviors are passed to the Action Selection (Maes, 1989), where a single behavior, the selected action in LIDA, is selected to execute.

Ideas concerning action execution have been briefly proposed in the LIDA Model, mainly expressed by two modules: Sensory Motor Memory and Motor Plan Execution depicted in the bottom left corner of Figure 1. However, the complete concept of action execution and its computational implementation have not yet been specified. We work on this here.

The original Sensory Motor Memory and Motor Plan Execution modules have been implemented by the SMS in detail (See Chapter 4). Two of other LIDA modules, Action Selection and Sensory Memory, provide relevant information—a selected behavior and the sensory data through a dorsal stream channel[1]–as inputs to the SMS. The selected behavior is a data structure resulting from the preceding Action Selection in the LIDA Model. It is comprised of three components: a context, an action[2], and a result. With some reliability, the result is expected to occur when the action is taken in its context. The SMS sends out motor commands to agent's actuators to generate its output in the environment.

Different types of learning are modeled in LIDA, in Figure 1 the learnings are illustrated using the channel arrows starting from Global Workspace (GW) to different modules. The global broadcast of the contents of the consciousness assists the learning. Particularly in our work, we have implemented sensory motor learning, which is represented by the channel from GW to the Sensory Motor Memory (See Chapter 6).

[1] In LIDA, the dorsal stream channel directly passes sensory information, some interpretation of sensory data, from the sensory memory to the action execution process.

[2] In this context, the term "action" refers to a component of a behavior. This differs from the general usage, such as in the phrase "action execution". In this document, we use "action" in the general sense, while "action of a behavior" refers to a particular component of that behavior.

3. The Action Execution Process Implemented in Different Cognitive Architectures: A Review

Introduction

This review focuses on cognitive models of action, or more specifically, of the action execution process, as implemented in several popular cognitive architectures. We examine the representations and procedures inside the action execution process, as well as the cooperation between action execution and other high-level cognitive modules. We finally conclude with some general observations regarding the nature of action execution. This work has been published in 2014 (Dong & Franklin).

Cognitive architectures are designed to be the basis for creating general, autonomous agents that can solve a wide variety of problems using a wide range of knowledge; they define and organize "the primitive computational structures that store, retrieve, and process knowledge" to pursue the agent's goals (J. Laird, 2012). Here we examine such cognitive models of actions, and especially of the action execution process as it is implemented in different cognitive architectures. The emphasis is placed on three questions: 1) What are the comprehensive representations and functional procedures of action execution? 2) How do action preparation/selection and action execution cooperate? and 3) What kind of specific designs are useful for generating actions that both achieve the agent's goals and execute appropriately using actuators in the environment?

The cognitive model of action in different cognitive architectures might be implemented variously because of the model's tasks. For example, an action implemented in a simulated chess match could be an abstract move, such as moving a piece one square to the left, or the low-level actions implemented in a simulated tennis match which are necessary for controlling the player's muscles (actuators) during the game. A chess action is driven only by the agent's internal goal; it

is not affected by the environmental situation—except the abstract representation of the position of the pieces—nor does it require the maintenance of specifications for its actuators. On the other hand, actions in tennis are generated on the basis of both the player's internal goals as well as the present environmental situation. Tennis, unlike chess, requires an action execution process that enables the agent to act in an uncertain and dynamic environment[1]. The action execution process is the focus of this review.

Even for the cognitive architectures in which action execution has been considered, the architecture may or may not completely implement it as a standard component. Some architectures implement a complete action execution process, such as ACT-R (J. Anderson, 2007) and LIDA (Franklin et al., 2014); other architectures only transform prepared/selected high-level actions into general low-level actions, and leave the action execution process to a domain-dependent external program, such as Soar (J. Laird, 2012). We review both types of architectures' action execution models below.

The next section describes a set of popular cognitive architectures with models for action execution. For each of these architectures, we first give a brief introduction and overview of the architecture's major components and functions; then we examine its specific implementation of action execution; specifically, we may compare its action execution part to that of LIDA. The following section concludes with a comparative summary of the action execution processes reviewed herein.

[1] A task environment is uncertain if 1) the agent sensors do not detect all aspects that are relevant to the choice of action, or 2) the next state is unable to be determined by the current state and the executed action; and the environment is dynamic if it can change while an agent is deliberating (Russell & Norvig, 2009).

Action Execution Processes of Cognitive Architectures

In this section, we review the action execution processes of different cognitive architectures in the ensuing subsections. In each of the subsections, after an introduction to the architecture, we examine the representations and procedures inside the action execution process, as well as the cooperation between action execution and other high-level cognitive modules.

4D/RCS

The review content of 4D/RCS mentioned here is mainly in response to a paper by Albus and Barbera (2005). We cite the paper for this whole subsection, unless other explicit citations or quotations are mentioned.

Real-time Control System (RCS) is a cognitive architecture designed to enable multiple levels of intelligent behaviors, achieved by a multi-layered hierarchy of sensory-interactive intelligent control process nodes. The most recent version, 4D/RCS, embeds the 4-D approach (Dickmanns, 1992, 2000), a machine vision technology, within the RCS control architecture.

Each node in the architecture contains sensory processing (SP), world modeling (WM), value judgment (VJ), behavior generation (BG), and a knowledge database (KD) (Albus & Barbera, 2005). A SP process receives input from sensors; SP and WM processes cooperate to filter, attribute, and classify the input data as a perception process; WM processes create and update the recognized states of the world in the KD; a BG process accepts tasks and plans, and executes behaviors to accomplish those tasks; a VJ process evaluates the results of tentative plans, and saves evaluation results in the KD.

Process nodes act hierarchically. The BG processes form a command tree: each input task is decomposed into a plan consisting of subtasks for subordinate BG processes. Information maintained in the KD is shared between WM processes in nodes above, below, and at the same

level within the same sub tree. Sensory data flow up the SP hierarchy typically forms a graph; and these data are populated by the WM in the KD at each level.

A WM predicts what will change in the world as the result of an action, and what will stay the same, giving its solution to the frame problem. Specifically, the location and direction of motion of objects in the world are represented in an image or map, and a simple comparison between one frame and the next distinguishes what changes from what does not in a dynamic environment.

A BG process receives tasks from a supervising BG process as input. The receiving BG process has a planner that decomposes each task into a set of coordinated plans for subordinate BG processes. During this period, tentative plans are proposed by the BG planner; the VJ evaluates the probable results of those plans as predicted by the WM; and a plan selector in the BG planner will choose the plan with the greatest value as the current plan. "For each subordinate there is an executor that issues commands, monitors progress, and compensates for errors between desired plans and observed results. The Executors use feedback to react quickly to emergency conditions with reflexive actions. Predictive capabilities provided by the WM may enable the executors to generate pre-emptive behavior" (Albus & Barbera, 2005).

In each node, the content of the selected current plan is moved from the planner into an "executor plan buffer" that initiates and guides the upcoming execution. This buffer is an interface between the planner and executor processes, and also the interface between deliberative and reactive processes.

At the top level of the architecture, the task is defined by the agent's goal that is established typically by a human operator outside of the agent. At each successive level in the hierarchy, tasks from the level above are decomposed into subtasks that are sent to the

subordinate levels below. Finally at the bottom level, decomposed task commands are sent to actuators to generate movements.

In each node of 4D/RCS, the execution is driven by a selected plan so as to reflect the requirements of the agent's goal in a bottom-up fashion that is reactive to the sensory input. Thus, at the lower levels of the architecture, the process nodes generate goal-directed reactive behaviors, while at the higher levels, the process nodes enable decision-making. 4D/RCS has implemented both the action preparation/selection and the action execution processes hierarchically; it allows a more gradual, and thus smoother, transformation from the agent's motivations, represented by a top-level task, to low-level actions that are directly applied to the agent's actuators.

ACT-R

Adaptive Control of Thought-Rational (ACT-R) is a cognitive architecture, a theory for simulating and understanding human cognition based on numerous facts derived from psychological experiments (Budiu, 2013). ACT-R consists of two types of modules: memory modules and perceptual-motor modules. Action preparation (selection) and action execution are implemented in ACT-R by using these two types of modules separately. An agent's motivation is achieved by choosing a proper action in memory modules, and the action is appropriately executed in the motor modules.

There are two types of memory modules in ACT-R: declarative memory and production memory. Declarative memory, represented in structures called chunks, maintains knowledge that people are aware of, and can share with others through a set of buffers. Procedural memory, encoded in production rules, represents knowledge outside of their awareness that is expressed in their behavior rules (ACT-R 6.0 Tutorial, 2012). A production rule is a condition-action pair.

The condition specifies a pattern of chunks that must be in declarative memory's buffers for the production to apply; a production fires if its condition matches the chunks in the buffers. The action specifies some actions, all of which are to be taken when the production fires (ACT-R 6.0 Tutorial, 2012). "[A] critical cycle in ACT-R is one in which the buffers hold representations determined by the external world and internal modules, patterns in these buffers are recognized, a production fires, and the buffers are then updated for another cycle" (J. R. Anderson et al., 2004).

ACT-R's perceptual-motor modules provide an elementary cognitive layer by which to couple the environment with the high-level cognition layer, including declarative memory and production memory (Byrne & Anderson, 2001). Perceptual-motor modules embedded in ACT-R 6.0 was heavily influenced by Kieras and Meyer's EPIC system (1996). Their major difference is that, only one production rule fires each time in ACT-R, while EPIC allows multiple rules fire, a parallel cognitive processing.

The motor module in ACT-R 6.0 is developed based on EPIC's manual motor processor (module). It is designed for modeling a simulated hand to operate a virtual keyboard and mouse (Bothell, n.d.). The motor module "receives commands from the production system that specify a movement style (e.g., PUNCH, as in punch a key) and the parameters necessary to execute that movement (e.g., LEFT hand and INDEX finger)" (Byrne & Anderson, 2001). The movement is generated through three phases: preparation, initiation, and execution. Below we describe each phase in turn.

In the preparation phase, the motor module builds a list of "features" which guide the actual movement; the features include the movement's style and parameters. For an example, in the movement of "punch the key below the left index finger", three features of PUNCH, LEFT,

and INDEX are involved in the preparation. (Byrne & Anderson, 2001). The amount of time that preparation takes depends on the number of features that need to be prepared—the more that need to be prepared, the longer it takes.

The motor module maintains a history of the last set of features that it prepared. The actual number of features that need to be prepared depends upon two things: the complexity of the movement to be made and the difference between that movement and the previous movement. On one end of the scale, the motor module is simply repeating the previous movement, then all the relevant features will already be prepared and do not require preparation. On the other end, a request could specify a movement that requires the preparation of full features, which have not been made in previous movements.

By default, the first 50ms after the preparation is movement initiation (Bothell, n.d.). After that, the movement may be executed if the motor module is not already executing a movement (Byrne & Anderson, 2001). "If a movement is currently being executed, then the newly prepared movement will be queued and will not be executed until the current movement and all other movements in the queue have been executed" (Byrne & Anderson, 2001).

In ACT-R, "[t]he world with which a model interacts is called the device. The device defines the operations which the perceptual modules can use for gathering information and the operators available to the motor modules for manipulating the device." (Bothell, n.d.). The executed movement sent out from the ACT-R motor module is passed to the related device module in order to carry out the execution in the real world.

There is typically no direct communication between the perceptual and motor modules in ACT-R. There is only very limited direct connectivity between perceptual and motor modules[2]. The data passed to the motor module always comes from the high-level declarative or production

[2] Spatial information in particular is communicated directly.

memory. This is almost the only significant conceptual difference between LIDA's SMS and ACT-R's motor module.

BECCA

This subsection heavily relies on a paper by Rohrer (2012). We cite it for the whole subsection unless we explicitly cite or quote otherwise.

A Brain-Emulating Cognition and Control Architecture (BECCA) is developed to address the problem of natural world interaction (NWI). NWI is the set of all tasks by which an agent pursues its goal in an unstructured physical environment. BECCA consists of an automatic feature creator and a model-based reinforcement learner to capture structure in the environment, and to maximize rewards respectively. BECCA issues action commands to a world module, and receives back a reward signal and observations in the form of sensory input and basic features. The world module is not the part of the standard BECCA architecture. Rather, it maintains simulations of the world, agent embodiment, actuators, and so on (see details later in this section).

The feature creator identifies patterns in the input. Sensory inputs are formed into groups based on how often they are co-active, and patterns within each group are identified as features and added to a feature space. The creator also maps the input into that feature space at each time step; the strongest feature voted by the projected input is activated. Features can be built hierarchically into higher-level features. Low-level features progressively activate high-level features, and the final set of activated features is passed to the reinforcement learner as a feature simulation.

In the reinforcement learner, a feature activity vector maintains the recent incoming feature simulations. A salience filter selects a single feature from the vector for attention[3], and the working memory maintains a brief history of attended features.

The reinforcement learner forms a model of the world and uses that model to select actions that will maximize the amount of reward. "The model consists of a list of feature-space transitions in the form of cause-effect pairs, each with an associated count and reward value. At each time step, the previous working memory is compared to the list of causes and the attended feature is compared to the list of effects. If a similar pair exists within the model, its count is incremented, and its reward value is adjusted toward the current reward. If there isn't a sufficiently similar pair, the previous working memory and attended feature are added as a new cause-effect pair" (Rohrer, 2012). In this way, the model is formed and updated. "The count associated with each transition establishes its frequency of observation, and the reward value represents the expected reward associated with making that transition" (Rohrer, 2012).

In the feature space, a transition represents a path segment that, when linked to other segments, may take a BECCA agent to its desired state from the current one. To predict the likely effect of a transition, the agent ranks the expected transition by 1) the matching strength between the current state and the cause of the transition, and 2) the count of the transition, and selects the transition with the highest rank.

Many effects are conditional on the actions selected by the agent. If an expected transition has both high similarity to the current state and high reward, and involves an agent's action on it, that action will be selected and executed.

[3] From the viewpoint of LIDA, this selection acts as the beginning of an agent's consciousness, and the most salient feature is the content of consciousness.

There are two types of action selection in BECCA, deliberative and reactive. Their major difference regards the content of the current state that is used in the prediction of transitions and in the action selection. In deliberative action selection, only the working memory (the recently attended features) is used to seed predictions and action selections from the model, while the entire feature activity vector is used in reactive action selection. The final selected action results from a nonlinear sum of the actions selected by the two selection modes. The deliberative action is also fed back to the working memory so that the action's effect can be recorded as part of the previous working memory content when the model is trained on the following time step.

A world module maintains 1) the environment—the simulation of the real world with which humans interact; 2) the physical embodiment of the agent—the virtual actuators and sensors of the hardware and the mechanisms that couple them; 3) preprocessing between sensors/actuators and the BECCA agent; and 4) a reward calculator providing reward value to the model of reinforcement learner[4]. From the viewpoint of action, a preprocessing step occurs between the selected action and the actuators; this step may include the incorporation of coordinated multi-actuator motions, fixed motion primitives, and heuristic goal pursuit subroutines.

An action processing mechanism unique to BECCA is its two-step action selection process: 1) Certain transitions, cause-effect pairs, are predicted (selected), and 2) an action is selected from the predicted transitions if the effect of the transition with the highest expectation relies on that action. These three components, cause, action, and effect, may be represented and organized differently in other architectures. For example, in many other architectures such as ACT-R and Soar (see its section below), a production rule is used to represent a condition-action

[4] A BECCA agent is not autonomous because its action decision is not driven by an agenda or motivation inside but by artificial rewards created outside of the agent.

pair, corresponding to cause and action in BECCA; while in LIDA, all of three components are encapsulated together: a data structure called a scheme includes context, action, and result, corresponding to BECCA's cause, action, and effect respectively.

Another issue is that the action execution process is outside the standard BECCA architecture. The commands for controlling actuators are not generated by the BECCA agent, but rather by the preprocessing of a world module. Also, the world module maintains the domain knowledge; this allows the BECCA agent to remain unchanged between many different domains (tasks). A similar strategy is implemented in Soar as well: its operator application doesn't really perform the action on actuators but an external program is always necessary to handle the final execution (performance). In contrast, some architectures do involve the action execution process in their architecture, such as ACT-R and LIDA.

CERA-CRANIUM

We cite a paper (Arrabales, Ledezma, & Sanchis, 2009) for this entire subsection of CERA-CRANIUM. Other citations or quotations will be explicitly noted in the text.

The Conscious and Emotional Reasoning Architecture (CERA) is a cognitive architecture structured in layers, providing a flexible framework with which to integrate different cognitive models of consciousness. CERA offers a basic hypothesis that conscious contents emerge as a result of competition and collaboration between specialized processors (functions). The Cognitive Robotics Architecture Neurologically Inspired Underlying Manager (CRANIUM) provides services through which CERA can execute thousands of asynchronous but coordinated concurrent processes.

CERA is structured in four layers, which are as follows:

1) The sensory-motor services layer comprises a set of communication services that provide a uniform access interface for the agent's physical sensors and actuators.

2) The physical layer is responsible for the low-level representations and preparations of the agent's sensors and actuators. Actuator commands are finally regulated at this level.

3) The mission-specific layer maintains complex perceptions and behaviors that are combined from and decomposed to the single sensory-motor content. Complex behaviors represent the agent's missions; one mission typically involves several goals.

4) The core layer includes a set of modules that perform higher cognitive functions. CERA is designed to allow customized core modules, such as attention, preconscious management, memory management, and self-coordination.

A CRANIUM workspace implements a set of specialized processors and a shared access working memory for the processors. Each of these processors is designed to perform a specific function, cooperating and competing with other functions. A CERA agent has two hierarchically arranged CRANIUM workspaces. The low-level workspace is located in the CERA physical layer and the high-level is located in the CERA mission specific layer. Based on the two CRANIUM workspaces, the perception flows are organized bottom-up in packages called single percepts, complex percepts, and mission percepts. Meanwhile in the same workspaces, a top-down action[5] flow includes mission behaviors, simple behaviors, and single actions; behaviors are iteratively decomposed until a sequence of atomic actions is obtained.

The core layer's operations are problem domain dependent. The operations are directed by the problem's interests, meta-goals, instead of mission-specific goals coming from lower

[5] The term "action" is an abstract concept; it refers to a class of action-kind concepts organized in different levels, including mission behaviors, simple behaviors, and single actions.

layers. Meta-goals shape the overall resulting behaviors. At any given time, a number of possible behaviors are generated in the CRANIUM workspaces; however, only those behaviors that are directed to the same locations as the represented meta-goals are likely to be selected and finally executed.

There are three types of processors related to the action process implemented in the CRANIUM workspaces.

1) Action planners transform the input behavior into the corresponding sequence of atomic actions that are submitted for eventual execution, so as to achieve the behavior's missions.

2) Action preprocessors prepare the atomic actions generated from action planners. Action preprocessors build so-called "single action constructs" to provide specific contextual data for actions. "Proprioceptive sensory data is also included in order to adapt actions to the current position of the actuators" (Arrabales et al., 2009).

3) Reactive processors are typically located in the CERA physical layer. They provide a quick response to stimuli that are considered harmful or highly undesired for the agent. These processors build simple behaviors to diminish or prevent negative consequences when unsafe or undesired situations are detected, without the participation of upper cognitive processes.

In summary, the CERA-CRANIUM action process supports both action selection—the selection between behaviors driven by the domain independent meta-goals—and the action execution—the decomposition from a behavior to a sequence of atomic actions, implemented by the action planners and the final execution preparation in the physical layer. CERA-CRANIUM

also establishes the interface between the agent and its environment in the sensory-motor services layer, providing the agent with the necessary environmental specifications.

CLARION

CLARION stands for Connectionist Learning with Adaptive Rule Induction ON-line. The purpose of this architecture is to capture all the essential cognitive processes within an individual cognitive agent (Sun, 2003, 2006). CLARION consists of a number of subsystems, including the action-centered subsystem (the ACS), the non-action-centered subsystem (the NACS), the motivational subsystem (the MS), and the metacognitive subsystem (the MCS). The ACS implements the action decision making of an individual cognitive agent (Sun, 2003). The MS motivates an agent to choose its actions by means of the rewards or gains which the agent seeks to maximize. The MS influences the working of the ACS by providing the context in which the goal and the rewards of the ACS are set (Sun, 2006).

The ACS consists of two levels of representation: the top level for explicit and the bottom level for implicit knowledge. The implicit knowledge generally does not have associated semantic labels, and is less accessible. Accessibility refers to the direct and immediate availability of mental content to the major operations that act on it. The bottom level is a direct mapping from perceptual information to actions, implemented in backpropagation neural networks[6] involving distributed representations, whose representational units in the hidden layer are capable of accomplishing tasks, but are generally not individually meaningful (Sun, 2003). Furthermore, the ACS might be composed of multiple instances of the backpropagation neural network, and a selection process is proposed to choose one of them. In contrast, the explicit knowledge is more accessible, manipulable, and has conceptual meaning (Sun, 2006). At the top

[6] Learning of implicit knowledge (the backpropagation network) transpires at the bottom level. "In this learning setting, there is no need for external teachers providing desired input/output mappings. This (implicit) learning method may be cognitively justified" (Sun, 2006).

level, a number of explicit action rules are stored, which are usually in the following form: current-state-condition → action (Sun, 2003). An agent can select an action in a given state by choosing an applicable rule. The output of a rule is an action recommendation, which is similar to the output from the bottom level (Sun, 2003).

The two levels implemented in CLARION's ACS operate independently: each of them makes action decisions in parallel based on the current state. The action sent out from both top and bottom levels are all performable. The final output action of the ACS is a combination of the output actions from the top and bottom levels. On the other hand, CLARION also models an interaction between the top and bottom levels, as well as between explicit and implicit knowledge. The input state or the output action to the bottom level is structured using a number of input or action dimensions; each of the dimensions has a number of possible values. At CLARION's top level, an action rule's condition or action is represented as a high-level node which is connected to all the specified dimensional values of the inputs or actions at the bottom level (Sun, 2003).

The overall algorithm of CLARION's action decision making consists of a structure that goes from perception to actions, and ties them together through the top and bottom levels of the ACS's cognitive processes as follows: "Observing the current state of the world, the two levels of processes within the ACS (implicit and explicit) make their separate decisions in accordance with their own knowledge, and their outcomes are somehow 'combined'. Thus, a final selection of an action is made and the action is then performed" (Sun, 2003). This decision making mechanism covers both action selection (preparation) and action execution. The top level of the mechanism provides the agent's internal goal for action execution, and the bottom level provides real-time environmental information. In contrast, in LIDA, we hypothesize that the two facets of

action, understandable and executable, cannot coexist in one presentation of action: action selection provides explicit desires of the action, and then action execution implicitly perform it. Both explicit and implicit representations of action act linearly but not in parallel, and neither of them can be both understandable and executable.

Note that the specifics of the agent's actuators are not involved in the representation of output actions (motor commands). This means the output performable actions of CLARION are independent of the motors of the robot's actuators; this is different than the executable motor commands mentioned in the introduction.

Additionally, learning has been applied in CLARION's ACS in three distinct ways: 1) the learning of implicit knowledge at the bottom level; 2) bottom-up learning, or learning explicit knowledge at the top level by utilizing implicit knowledge acquired in the bottom level; and 3) top-down learning, the assimilation at the bottom level of explicit knowledge from the top level (which must have previously been established through bottom-up learning) (Sun, 2003).

EPIC

This review of EPIC mainly relies on an EPIC overview paper (Kieras & Meyer, 1997). We cite the paper for this entire subsection, unless explicit citations or quotations are claimed in the text.

Executive Process-Interactive Control (EPIC) is a cognitive architecture created for modeling human task performance. EPIC has three processing modules: 1) sensory processors, 2) motor processors, and 3) a cognitive processor that represents a general procedure as a set of production rules to perform a complex multimodal task. During the execution of a procedure, EPIC specifies both the production-rule programming for the cognitive processor as well as the relevant perceptual and motor processing parameters.

Specifically, there are visual and auditory processors that accept multimodal stimuli as perceptual input. This input is stored in the corresponding working memory located in the cognitive processor.

The motor processors produce a variety of simulated movements for the hands, eyes, and vocal organs. From the cognitive processor, an action's command is sent to a motor processor that consists of the movement type (name) and certain parameters.

There are two steps for a complete movement: a preparation and an execution. In the preparation, the motor processor transforms the movement type (name) into a set of movement features and generates them. "The time to generate the features depends on how many features can be reused from the previous movements (repeated movements can be initiated sooner), and how many features have been generated in advance" (Kieras & Meyer, 1997). The ensuing execution step begins with an initiation phase, followed by the actual physical movement. In addition to reusing the features remaining from previously executed movements, the movement features may be prepared in advance. "If the task permits the movement to be anticipated, the cognitive processor can command the motor processor to prepare the movement in advance by generating all of the required features and saving them in motor memory" (Kieras & Meyer, 1997).

Rather than the voluntary movements produced by various motor processors, as mentioned above, the oculomotor processor may produce the involuntary (reflexive) eye movements, either saccades or small smooth adjustments in response to the visual situation.

In the cognitive processor, there is a set of production rules that specify which actions are performed in certain situations to accomplish a task. The format for a rule is <rule-name> IF <condition> THEN <action>. The rule condition will test the contents of the production system

working memory. The rule action will then add or remove information from the working memory, or send a command to the motor processors. Motor working memory stores information about the current state of the motor processors.

The cognitive processor operates cyclically. During each cycle, the contents of working memory are first updated with the perceptual input information and the previous cycle's modifications; then the contents of the production system working memory are updated based on the rules that fire, and the action commands of the firing rules are sent to the motor processors. A unique feature of EPIC is that it will fire all rules in which conditions match the contents of working memory, and will execute all of the corresponding actions; in other words, the EPIC cognitive processor allows parallel cognitive processing.

The EPIC model builder should provide 1) the task environment, either physical or simulated, which includes the characteristics of relevant objects external to an EPIC agent, 2) a set of tasks which specify the environmental events, 3) task-specific sensory data encodings (representations), and 4) the task procedures represented as production-rules.

In summary, regarding the action process in EPIC, action selection and action execution have been implemented by production rules firing in the cognitive processor, and movement preparation and execution in the motor processors separately.

GLAIR

We cite a paper (Shapiro & Bona, 2010) for this subsection, unless other citations or quotations are explicitly mentioned in the text.

Grounded Layered Architecture with Integrated Reasoning (GLAIR) is a multi-layered cognitive architecture for embodied agents. In GLAIR, the highest layer is the Knowledge Layer (KL), which contains the agent's beliefs, and performs reasoning and selects acts. The middle

layer is the Perceptuo-Motor Layer (PML), which grounds the KL symbols in perceptual structures and primitive actions. The lowest layer is the Sensori-Actuator Layer (SAL), which contains the controllers of the sensors and actuators of the hardware or software agent.

The KL contains the beliefs of the agent; with respect to action, it includes 1) plans for carrying out complex acts and for achieving goals, 2) beliefs about the preconditions and effects of acts, and 3) policies about when, and under what circumstances, acts should be performed.

The PML is responsible for the communication between the KL and the SAL by three top-down sub layers: the PMLa, the PMLb, and the PMLc. The PMLa grounds the KL symbols, providing primitive actions; the PMLc abstracts the sensors and actuators into basic behavioral repertoire of the robot. The PMLb translates and communicates between the PMLa and the PMLc.

GLAIR agents execute a sense-reason-act cycle. The original focus of the GLAIR design is on reasoning, but not problem solving or goal-achievement such as ACT-R. Its basic driver is based on reasoning: either thinking about some perceptual input, or answering some question. If the input (typically a natural language utterance) is a statement or a question, the GLAIR agent will output the proposition of the statement or the answer to the question respectively. A later added acting component allows a GLAIR agent to obey a command, to perform an act and to achieve a goal. When the input is a command, the agent will perform the indicated act, implementing an action process that is the focus of this review.

An act consists of an action and one or more arguments. For an example, “the term find (Bill) denotes the act of finding Bill (by looking around in a room for him), composed of the action find and the object Bill” (Shapiro & Bona, 2010). Acts may be classified as either external, mental, or control. External acts affect the outside world. Mental acts affect the agent’s

beliefs and policies. Control acts are the control structures used to support ground computational processes such as inference operations so as to maintain the GLAIR acting system.

GLAIR acts may also be classified as primitive, defined, or composite. Primitive acts are the basic acts predefined in the PMLa. Composite acts consist of primitive acts. A defined act is the abstracted identifier of a plan; if a GLAIR agent is to perform a defined act, it "deduces it" to a plan and performs it. Such a plan is an act, which can be either a primitive, composite, or defined. It is assumed that a plan is "closer" to primitive acts than a defined act. A defined act may have different plans depending on circumstances. The use of conditional plans has allowed a GLAIR agent to select among alternative procedures to perform.

The procedure for performing an act consists of several steps:

1) To attempt to achieve the preconditions of the act and, if it is a defined act, to prepare a set of candidate plans that can be used to perform the act.

2) If the act is a defined act, only its most suitable plan is tried, after which the agent will automatically consider it successful.

3) Effects of the act are derived before the act is performed; and after that, the agent will consider all the effects of the act to hold.

In GLAIR, the act is represented in the same formalism as other declarative knowledge such as the agent's beliefs. However, the declarative knowledge and the act are maintained by the KL and the PMLa layers separately. In this way, the declarative and procedure knowledge are represented with the same formalism but operated in different levels.

The PMLa layer grounds the KL by providing primitive actions, transforming high-level actions to low-level. Although the actions maintained in PMLa are primitive, they are independent of the implementation of the agent's body. It is the PMLc layer which directly

abstracts the actuators[7] of the robot into the basic behavioral repertoire of the robot body. The primitive actions in the PMLa are translated to these basic behavioral repertoires—the basic execution units of GLAIR—through PMLb.

Two steps occur during the process of action execution: 1) actions are initially selected in the KL driven by reasoning results and translated into their primitive format in the PMLa layer; and 2) the primitive actions are translated to actuator-dependent basic behavioral repertoires in PMLc through PMLb, and then those basic units are sent to SAL for execution.

ICARUS

We cite a paper (Pat Langley & Choi, 2006) for this entire subsection. Other citations or quotations will be notated explicitly.

ICARUS is a cognitive architecture for physical agents that has been influenced by results from cognitive psychology. ICARUS's most basic mechanism, conceptual inference, operates by matching long-term conceptual structures against short-term perceptual data and beliefs. Based on the inference, ICARUS operates processes for goal selection and skill execution.

In order to perceive the states of the external environment, ICARUS incorporates a perceptual buffer (short-term memory) that describes aspects of the environment. The element stored in this memory responds to a particular object, and characterizes the object's specifications at the current time step. ICARUS also includes a conceptual memory, which contains long-term structures that are the classifications of the environmental state. During each cycle of conceptual inference, objects are perceived first into the perceptual buffer, where they begin to match against long-term conceptual classifications. The system updates its belief

[7] In the original paper (Shapiro & Bona, 2010), the authors use the term "effectors" instead of "actuators" as we do here.

memory based on the results of this matching. The elements in the belief memory describe relations among objects. ICARUS repeats this inference process, updating its beliefs about the environment over time.

In order to take action in the environment, ICARUS has a performance mechanism that concerns goals the agent wants to achieve, skills the agent can execute to reach them, and intentions about which skills to pursue.

Specifically, ICARUS includes a goal memory that contains a list of the agent's objectives. The goal is a set of concept instances that the agent wants to achieve, and the goal memory takes the same form as belief memory. An agent needs to select only one goal at a time among multiple elements in goal memory. On each time, it chooses the goal with the highest priority that is not yet achieved.

ICARUS has a long-term skill memory that contains skills it can execute in the environment and use to accomplish goals. Each skill has a head and a body. The head states the skill's objective, and the body specifies the necessary perceptual data and beliefs of a skill. Multiple skills may have the same head; they provide different ways to achieve the same goal under different conditions. Once the agent has chosen a goal, it selects a skill to achieve the goal based on a matching between the skill body's specifications and the agent's current perceptual data and beliefs[8].

The skills are organized hierarchically. "Primitive skills" refers to actions that the agent can execute directly in the environment, while non-primitive skills are goals/sub-goals that the agent might seek to achieve. Primitive skills correspond to the executable motor commands mentioned in the introduction. During the execution of a non-primitive skill, the agent must find

[8] In LIDA, a similar matching occurs in the process of recruiting schemes. Schemes are selected based on a matching between the agent's conscious contents, the most salient current situation, and the scheme's context and result contents.

a path downward from its goal to one or more terminal primitive skills in the hierarchy. Once the agent has selected a skill path for execution, it invokes the actions referred to the primitive skill or skills in the path.

If the applicable skill is not found, an impasse appears, and the agent invokes its problem solver for achieving the goal. The agent decomposes the goal into sub-goals iteratively until find the skills for achieving them. The skills applicable for all sub-goals are finally selected and then executed to achieve the goal. A new skill is learned for the goal by structuring the applicable skills for those sub-goals. The similar impasse resolving mechanism has been implemented in Soar (see its section below) as well.

LIDA

For historical reasons LIDA stands for Learning Intelligent Distribution Agent. The LIDA Model (Franklin et al., 2014) is a conceptual, systems level model of human mental processes, used to develop biologically-inspired intelligent software agents and robots. It implements and fleshes out a number of psychological and neuropsychological theories, but is primarily based on Global Workspace Theory (Baars, 1988, 2002).

The LIDA Model is grounded in the LIDA cognitive cycle. Each cognitive cycle consists of three phases: 1) the LIDA agent first senses the environment, recognizes objects, and builds its understanding of the current situation; 2) by a competitive process, as specified by Global Workspace Theory (Baars, 1988), it then decides what portion of the represented situation should be attended to, and broadcast to the rest of the system; 3) finally, the broadcast portion of the situation supplies information allowing the agent to choose an appropriate action to execute, and modulates learning (Franklin et al., 2014). The simulated human mind can be viewed as functioning via a continual, overlapping sequence of these cycles.

The dual aspects of action are represented in the LIDA Model as the distinct processes of action selection and action execution. Specifically, the sensory data retrieved in LIDA influences the action process at two "levels". At one level, sensory data is filtered through the understanding and attention phases, and then helps recruit appropriate actions in the action selection process; the selected result is used to initiate certain processes operating in the concomitant action execution process, ultimately generating executable low-level actions. At the other level, the sensory data is sent through a dorsal stream channel directly to the action execution process for assisting the execution (See Figure 1).

The concept of scheme has been borrowed to implement LIDA's action selection. A scheme is a data structure representing the procedural knowledge stored in LIDA's Procedural Memory. It is composed of three components: a context, an action[9], and a result. With some reliability, the result is expected to occur when the action is taken in its context. In LIDA's action selection process, one or more schemes are recruited based on the most salient current situation. And then, the schemes' context and result components are bound with additional information from the current situation, so that the recruited schemes are instantiated into behaviors. A behavior has a data structure similar to a scheme, but the components of context and result have been instantiated with concrete values. Finally, a behavior is selected based on the agent's motivation and its understanding of the current situation.

The process of action execution has been recently added to LIDA, modeled by the Sensory Motor System (SMS) (Dong & Franklin, 2014b). Two other LIDA modules, Action Selection and Sensory Memory, provide relevant information—a selected behavior and the sensory data through a dorsal stream channel, respectively–as inputs to the SMS. The SMS sends

[9] In this context, the term "action" refers to a component of a scheme. This differs from the general usage, such as in the phrase "action execution". In this document, we use "action" in the general sense, while "action of a scheme" refers to a particular component of that scheme.

out motor commands to an agent's actuators to execute its selected action in the environment. Within the SMS, three data structure types have been proposed—the motor command (MC), the motor plan (MP), and the motor plan template (MPT)—and three types of processes have been modeled: online control, specification, and MPT selection.

A motor command (MC) is applied to an agent's actuator. Every MC has two components: a motor name, and a command value. The motor name indicates which motor of an actuator the MC specifically controls, while the command value of a MC encodes the extent of the command applied to the motor.

An MP acts like a MC generator that generates MCs based on the sensory data transmitted via the dorsal stream. An MP is implemented based on the principles of the subsumption architecture (Brooks, 1991), a reactive structure. In the subsumption architecture, 1) the sensory data is linked to directly thus determining the selection of motor commands that drive the actuators; 2) it decomposes a robot's control architecture into a set of task-achieving behaviors; and 3) it does not maintain any internal model of the world[10], and is without any explicit representations. The MP generates motor commands as the output of the SMS to the environment (using actuators), while environmental data directly influence the generation process through the dorsal stream channel from Sensory Memory. These cyclically occurring processes are called the online control process of the SMS.

An MPT is an abstract MP that resides in an agent's long-term memory (Sensory Motor Memory in LIDA). It has a set of motor commands (MCs) that are not yet bound with the command values, whereas after a specification process, the motor commands are bound with specific values, instantiating the MPT into a concrete MP. Both sensory data from the dorsal

[10] Although no central world state is one of the essences of the subsumption architecture, implicit understanding and expectation of the environment has been built into the architecture by its layered structure.

stream and the selected behavior determine the specification process (Dong & Franklin, 2014b). MPTs and MPs have very similar structures, so they are designed with nearly the same data structure. Their major differences are 1) an MPT is persistently stored in a long-term memory, while an MP is short-term, and created anew each time it is used; and 2) typically an MP's command values have been specified, while those of an MPT have not.

As the SMS's initial process, A MPT selection acts to select and initiate a MPT by an incoming selected behavior before the MPT is specified into a concrete motor plan. MPT selection chooses one MPT from the set of those associated with the selected behavior. It connects action selection to action execution. Currently this selection is built in by the agent designer (Dong & Franklin, 2014b).

A sensory motor learning has been implemented in LIDA's SMS. See the learning in Chapter 6.

Soar

Soar is a cognitive architecture in pursuit of general intelligent agents (J. Laird, 2008). "The design of Soar is based on the hypothesis that all deliberate goal-oriented behavior can be cast as the selection and application of operators to a state. A state is a representation of the current problem-solving situation; an operator transforms a state (makes changes to the representation); and a goal is a desired outcome of the problem-solving activity" (J. E. Laird et al., 2012).

Soar has separate memories for descriptions of its current situation and its long-term knowledge. It represents the current situation in its working memory, which is Soar's short-term memory and maintains the sensory data, results of intermediate inferences, active goals, and

active operators (J. E. Laird et al., 2012). The long-term knowledge specifies how to respond to different situations in the working memory so as to solve a specific problem.

All of Soar's long-term knowledge is organized around the functions of operator selection and operator application, which are organized as a processing cycle as described below (J. Laird, 2008; J. E. Laird et al., 2012).

1) Elaboration. Knowledge with which to compute entailments of short-term memory, creating new descriptions of the current situation that can affect operator selection and application indirectly.

2) Operator Proposal. Knowledge with which to propose operators that are appropriate to the current situation based on features of the situation tested in the condition of the production rules.

3) Operator Comparison (Evaluation). Knowledge of how to compare candidate operators, to create preferences for some proposed operators based on the current situation and goal.

4) Operator Selection. Knowledge with which to select an operator based on the comparisons. "If the preferences are insufficient for making a decision, an impasse arises and Soar automatically creates a substate in which the goal is to resolve that impasse. … The impasses and resulting substates provide a mechanism for Soar to deliberately perform any of the functions (elaboration, proposal, evaluation, application) that are performed automatically/reactively with rules." (J. Laird, 2008)

5) Operator Application. Knowledge of how the actions of an operator are performed on the environment, to modify the state.

Four of the above functions require retrieving long-term knowledge that is relevant to the current situation: Elaborating, Operator Proposal, Operator Comparison, and Operator Application. These functions are driven by the knowledge represented as production rules (J. E. Laird et al., 2012). A production rule has a set of conditions and a set of actions. The production's actions are performed if its conditions match working memory; that is, the production fires (J. E. Laird et al., 2012). The other function, Operator Selection, is performed by Soar's decision procedure, which is a fixed procedure that makes a decision upon the knowledge that has been retrieved (J. E. Laird et al., 2012).

An operator contains preconditions and actions; its action differs from a production rule's action. The operator action is an output for the agent to its internal or external environment, while actions of a production rule generally either create preferences for operator selection, or create/remove working memory elements (J. E. Laird et al., 2012).

When Soar interacts with the environment, it must make use of a mechanism that allow it to effect changes in that environment; the mechanism provided in Soar is called output functions (J. E. Laird et al., 2012). During the operator application process, Soar productions could respond to an operator by creating a structure on the output link, a substructure which represents motor commands for manipulating output. Then, an output function would look for specific motor command in this output link, and translate this into the format required by the external program that controls the agent's actuators (J. E. Laird et al., 2012). "[In the external program,] functions that execute motor commands in the environment use the values on the output links to determine when and how they should execute an action" (J. E. Laird et al., 2012). This means that it is Soar's external program, not its output functions, that specifies how to execute the action in detail (when and how).

In the case of Soar's output, motor commands, which cannot be directly performed (executed) on the external world, an external program is always necessary to handle the final "real" execution (performance) for Soar. Soar does not cover the representation of environmental information related to action. This allows it to maintain generality with a clear standard, without the necessity of considering every possible domain that the Soar agent might live in. Note the term "motor commands" in Soar expresses completely different concepts than in other architectures, such as LIDA, although it is used to represent the final output data in both cases. In LIDA, motor commands are executable, while in Soar they are not. By saying that motor commands are "executable", we mean that these commands 1) are able to be applied to the agent's actuators directly, and 2) are maintained in an order appropriate to both the agent's internal goal and the current environment's dynamics.

Conclusions

We realize that the action execution processes implemented in the cognitive architectures described above have many similar representations and procedures though they use different structures. We conclude with some general observations regarding the nature of these representations and procedures, followed by a summary.

Inside Action Execution

Each cognitive architecture having a representation at an explicit level of knowledge, typically also needs a process that transforms high-level knowledge into motor-level commands; that is, action execution. For example, a task (sub task) is transformed into task commands in 4D/RCS, a production rule's action into movements in ACT-R and EPIC, a behavior into atomic actions in CERA-CRANIUM, an act into basic behavioral repertoires (the basic execution units) in GLAIR, a non-primitive skill into primitive skills in ICARUS, and a behavior into a sequence

of motor commands in LIDA. Some other architectures, such as BECCA and Soar, prepare the actions for external programs to finish the action execution. These architectures accomplish only the initial phase of the action execution process. CLARION has a unique action decision mechanism in its two levels of representation. As we have discussed above, this mechanism covers both action selection (preparation) and action execution (performance), though its action performance does not maintain the specifics of the actuators within the representations of the output actions.

The Cooperation between Action Selection and Execution

A goal-directed action resulting from action selection concomitantly initiates the initial action execution process. This process may be implemented in two different ways. One possibility is to decompose the selected goal-directed action into primitive actions, in which case the action's data structure is gradually broken down from high-level to low-level without qualitative changes, such as tasks in 4D/RCS, behaviors and atomic actions in CERA-CRANIUM, actions in CLARION, skills in ICARUS, and operators in Soar. The other option is to map the selected goal-directed action to another type of action representation—the action's data structure has been qualitatively changed—that enables the generation of the ensuing low-level actions, such as a production rule's action mapping to a movement style in ACT-R and EPIC, a transition mapping to an action in BECCA, and a behavior mapping to a Motor Plan Template (MPT) in LIDA. GLAIR combines the natures of these two options: it first decomposes acts into primitive ones, and then translates them into actuator-dependent basic behavioral repertoires.

Environmental Information for Action Execution

During the action execution process, additional environmental information is usually supplied to specify and adjust the final command values for execution. For example, in both ACT-R and EPIC, a preparation phase operates during the action execution process to build and specify a list of "features"; the features include the movement's style—the name of a low-level action's identifier—and the values of its parameters. In CERA-CRANIUM's action execution, action preprocessors provide specific contextual data for preparing atomic actions. In LIDA, the sensory data sensed through its dorsal stream channel is sent directly to the action execution process, so that a Motor Plan Template (MPT) is instantiated into a Motor Plan (MP) that generates the final motor commands.

This additional information might be directly sensed from the environment through sensory processes; in this case, a direct communication between the perceptual and motor modules is implemented to assist the action execution. For example, in CLARION's bottom level, perceptual information directly maps to actions, and in LIDA, sensory data may be sent to the motor system directly through a dorsal stream channel. On the other hand, this environmental information might come from high-level cognitive modules that store the current state of the environment. For example in ACT-R, the data passed to the motor module comes from high-level declarative or production memory.

Summary

Based on the above review, we can identify certain common characteristics for action execution. It provides an elementary cognitive layer by which a cognitive architecture couples the environment with its high-level cognitive modules, including action selection. Action execution finalizes an agent's intention, so that it generates a cognitive architecture's output.

Action execution involves domain specifications—including environmental specifications and those of the agent's actuators—sufficient to enable execution to be actuated in the environment. However, it does not contain so much abstract knowledge, which is the province of high-level cognitive modules.

On the other hand, action execution may vary in concrete implementations, with respect to its representations, procedures, and the means of cooperation with high-level cognitive modules. Also, action execution itself may or may not be considered a standard module of a cognitive architecture, so that architectures may differ in their degree of completion.

4. A New Action Execution Module for LIDA: The Sensory Motor System

Introduction

I and Franklin have proposed two facets of action: "what to do" and "how to do it" (2014b). Following this hypothesis, the dual aspects of action are represented in the LIDA Model as the distinct processes of action selection and action execution. Action selection has been described in previous work (Franklin et al., 2014). Here we specify the action execution in the form of the Sensory Motor System (SMS), a cognitive model for an action execution process in LIDA. The SMS responds by transforming a desired understandable action, a selected behavior in LIDA, into an executable low-level action sequence, a sequence of motor commands, and executing them.

The next section describes the subsumption architecture (Brooks, 1986, 1991), which is used as the SMS's prototype. The following section introduces the SMS concepts and its high-level designs. Two data structure types have been proposed—the Motor Plan Template (MPT), and the Motor Plan (MP)—and three types of processes have been modeled: online control, specification, and MPT selection. The next section introduces the simulation of a specific action execution process, gripping. One aspect of grip, grip aperture, has been simulated and compared to the human performance. Furthermore, we compare the SMS of LIDA with the action (execution) process implemented in three other cognitive architectures: ACT-R, Soar, and CLARION. We conclude with a discussion of the benefits of modeling a natural action execution process with the SMS for LIDA, followed by a conclusion for SMS development.

The subsumption architecture

The subsumption architecture is a parallel and distributed computation formalism for connecting sensors to actuators (Brooks, 1986, 1991), a type of reactive structure for controlling a robot. In the subsumption architecture, specific behaviors are merged into a comprehensive

classification, organized in multiple layers (levels), where the components in each layer are Augmented Finite State Machines (AFSMs). Layers (levels) or AFSMs are connected by two types of processes: inhibit or suppress, which are represented by encircled uppercase I or uppercase S respectively. As shown in Figure 2, a signal coming into the high-level input of the inhibit process—from AFSM 2 to the encircled uppercase I—terminates the signal passing through the low-level input—from AFSM 1 to the encircled uppercase I. The suppress process, encircled uppercase S, operates as does the inhibit process except that its high-level input signal replaces (not terminates) the low-level one. Inside the architecture, there are no direct channels between modules, nor is there any central forum for communication (Connell, 1989b); the environment is used as the communication medium because " [t]he world is its own best model" (Brooks, 1990, p. 3).

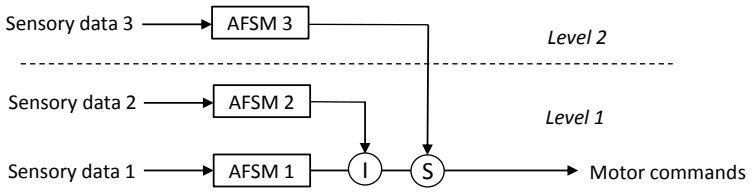


Figure 2. The subsumption architecture example

The capabilities of the subsumption architecture match many required features of action execution as we model it (see next Section). First, the subsumption architecture fulfills the requirements for modeling online control of action execution. In this architecture, the sensor is directly linked to the motor that drives the actuators. This kind of mechanism follows the hypothesis that the executable action is driven by the content of bottom-up sensory information coming through the dorsal stream.

Second, the subsumption architecture also satisfies the requirements of transforming an understandable action, a selected behavior, into executable motor commands. Marc Jeannerod, citing the work of Searle (Searle, 1983), built upon the concept that covert action representation is followed by overt real executed action. In detail, "the conceptual content, when it exists (i.e., when an explicit desire to perform the action is formed), is present first. Then, at the time of execution, a different mechanism comes into play where the representation loses its explicit character and runs automatically to reach the desired goal" (Jeannerod, 2006, pp. 4-5). We believe the concepts used in SMS are the same as Jeannerod's, although our terminologies differ.[1]

In order to run automatically to reach the desired goal without "its explicit character", a general idea is to decompose an understandable action into low-level executable motor commands, and the desired goal into separate sub-goals to be accomplished with low-level tasks. The subsumption architecture supports this kind of mechanism. "It's a method of decomposing a robot's control architecture into a set of task-achieving behaviors or competences" (Dawson, n.d.). In other words, the architecture decomposes both the action and the desired goal into motor commands and competences, respectively. A competence refers to a low-level task that could be considered a link connecting a desired goal to executable motor commands. Although the subsumption architecture is typically considered the classic example of a reactive structure, in the present case, a competence actually works proactively as well because of its task-achieving behavior. It achieves both the "how to do" and the "for what purpose" of an action though the purposes it aims to achieve are very specific and low-level. In the subsumption architecture, a

[1] 'An explicit desire to perform the action' refers to a selected behavior; 'a different mechanism' is our SMS; and 'the representation [that] loses its explicit character' indicates executable motor commands.

competence conceptually plays a role like a watershed, dividing the desired understandable actions and executable motor commands.

Furthermore, the subsumption architecture has no central control, and thus it develops a piece of cognition that minimizes the role of representation (Brooks, 1991; De Vega, Glenberg, & Graesser, 2008, p. 72). This fact is consistent with our design requirement, as Jeannerod proposed above, for the absence of an understandable action's "explicit character" in the action execution process. This explains why action execution remains outside the awareness of the agent, although it could become aware of the execution indirectly. We will discuss this later (See below sections).

On the other hand, the subsumption architecture doesn't interact with high-level goal-directed actions, which is an essential requirement of the SMS. Also, the parameters designed in the architecture are not changeable, so the motor commands it generated cannot be specified at runtime. We have extended the subsumption architecture in the design of SMS to meet these points: interaction with high-level actions and dynamically specifying parameters at runtime. The SMS's full definition is described in the next section.

Conceptual Design of the SMS

This section introduces the concepts relatively abstractly, so that it supports a high-level design of SMS, filling the gap between the hypotheses regarding the human mental and behavioral, and the detailed computational designs of SMS, such as its data structures and algorithms. Working towards a biologically inspired understanding of the action execution process, such as the action mentioned in the two visual systems, this section describes the possible functional representations and processes in detail. In addition, this section describes

reliable implementation requirements, in order to implement the action execution process computationally as we will see later (see next section).

The SMS must transform a selected behavior into a sequence of motor commands, and execute them using the agent's actuators. In this section, the emphasis is on the concept of transformation, while task execution is introduced later in experimental parts (next section).

The motor plan and online control

The output of the SMS, a sequence of motor commands, is sent out in a certain order; hence the agent's movement is not chaotic but is chosen with the intent of reaching a certain goal. However, this "ordering" effect is not a plan working inside the SMS to determine when each motor command will be sent out. The action execution process is running in a real world with unlimited environmental data available, much of which heavily affects the order of the motor commands. It is hard to anticipate such environmental situations fully enough to explicitly prepare a specific sequence of motor commands before the execution begins.

Citing the work of Herbert Simon (1969), Rodney Brooks built upon the concept that complex behavior need not necessarily be a product of an extremely complex control system; rather, it may simply be the reflection of a complex environment (Brooks, 1986). Therefore, in contrast to a fixed plan, a reactive structure is introduced to model the source of ordered motor commands (Figure 3). Inside the SMS, first a set of motor commands are built in; each of them is represented by a ©, which is independent of any timestamp. Next is a set of triggers, represented by Tx; a trigger activates a specific command in order to send it out as a part of the SMS output when the input sensory data matches one or more of the trigger's conditions. The subscript x stands for the number of conditions a trigger contains. Third, before sending out the commands, a choice function chooses a command from possibly multiple candidates as the final output at

each moment. The choice strategy must be implemented when applying this high-level design to a concrete action execution process. The set of motor commands, the triggers, and the choice function are referred to as a Motor Plan (MP), which specifies what to do in a particular situation, independently of time.

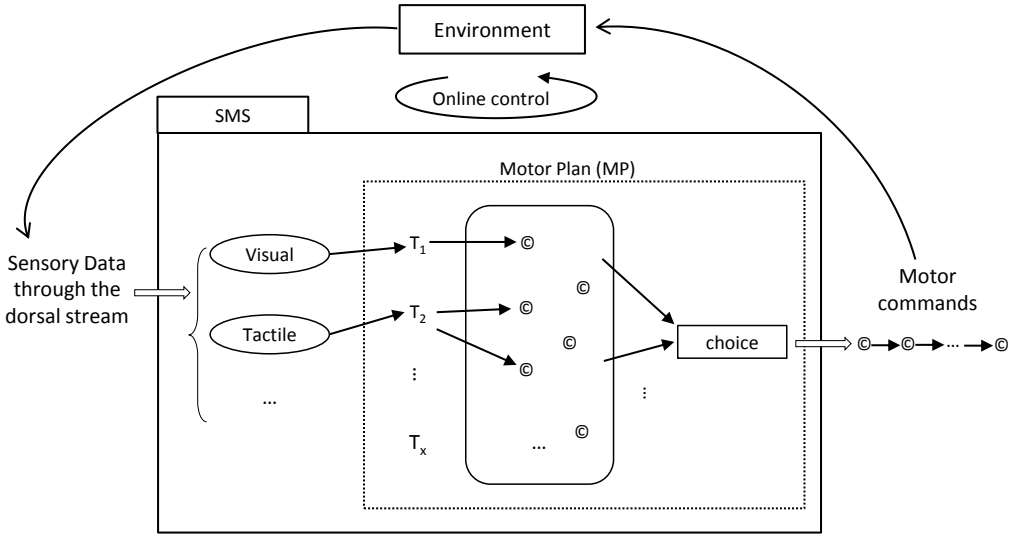


Figure 3. SMS with a MP and online control diagram

An environment located outside the SMS is shown in Figure 3 as well; it provides environmental data to the SMS at the appropriate time through the dorsal stream (see Figure 1). These sensory data are classified based on different modalities, such as visual, tactile, etc., and sent to the triggers. The output of the SMS, a sequence of motor commands, executes using the agent's actuators, and thereby acts on the environment. These processes occur cyclically between the environment and the SMS, which models the hypothesis regarding one of the dorsal stream's roles, online control.

As shown in Figure 3, the SMS resembles a wrapper for the MP, supporting pre-processed sensory data, and passing the MP's output to the agent's actuators acting on the

environment. On the other side, a MP acts like an independent component inside the SMS. It involves static data structures: a set of motor commands, as well as the triggers, implemented as small processes.

This online control mechanism designed in the SMS reflects the ideas of a reactive structure: it allows the MP to generate motor commands based on the sensory data, adapting to the current environmental situation. This is inspired by the principles of the subsumption architecture. The grounding component inside the subsumption architecture is a type of Finite State Machine (FSM); specifically, it could be an augmented FSM (Brooks, 1986) or a module (Connell, 1989b). As shown in Figure 4, a set of states are contained in a FSM, and an internal variable, Current State, maintains which state is the current one. Depending on the input data, the current state's transition conditions may be met so as to change the current state into another one, updating Current State in the process. Besides a transition, some states have another attribute, a command; the command is sent out when the state's transition is activated. Compared to a FSM, a MP's trigger containing conditions corresponds to a FSM state plus its transition conditions, and the trigger is capable of selecting the command based on the input data as well. The only difference between a trigger and a FSM is that the FSM contains commands, while in a MP, a set of motor commands is separately maintained outside of the triggers. Conceptually, a FSM equals a trigger that points to motor commands. Representing a set of motor commands independently from triggers provides a clearer classification between the data structure and the processes that operate on it; also, it helps to clearly emphasize the temporal independency of the motor commands.

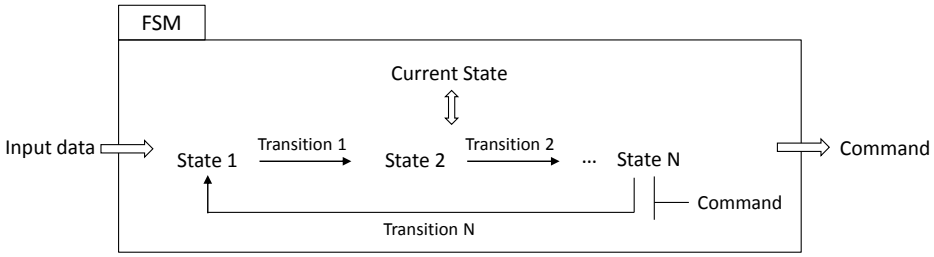


Figure 4. A FSM and its components

The choice function is generalized from the inhibit and suppress operations of the subsumption architecture, which connect many FSMs together based on a carefully designed structure for certain task-achieving behaviors. The choosing criteria used in inhibit and suppress are fixedly created in a hierarchical structure—being in a higher layer gives a motor command higher priority for selection. We generalized the implementation of a criterion, and leave its specification to the computational design, while, as in the case of inhibit and suppress, only one motor command is permitted to be sent out at one time as a choice result.

Motor Commands

A motor command (MC) is applied to an actuator of an agent; therefore its format relies on the configuration of that actuator, which, theoretically, is outside the SMS's purview. On the other hand, since MCs are the output of the SMS, a general MC format has been defined according to the definitions that follow.

Every MC has two components: the motor name and a command value. The motor name tells to which motor of an actuator the MC specifically applies. As an instance, if one joint of a finger is considered a motor of the actuator hand, the joint's name then can be the motor name of a MC.

The command value of a MC encodes the extent of the command applied to the motor. As an example, the command value applied to a finger's joint could be *positive five* within a real number domain. Here the unit of a command value is not specified, which means the type of the command is unknown: is it force, velocity, or distance? Being agnostic to a command's type is reasonable because 1) conceptually, the agent need not be aware of the type of the command in the action execution process (although the agent's designer must be); and 2) computationally, since a MC's command type is implicitly fixed by design—e.g., the type of a command applied to a finger's joint is always the force—the command type need not be explicitly declared in a MC.

The motor plan template and specification

A set of motor commands (MCs) is prepared inside a Motor Plan (MP) and bound with fixed command values. In order to specify a MC's command value before the execution begins—thus modeling one of the dorsal stream's hypothesized roles, specification—a Motor Plan Template (MPT) is proposed and a specification process is created in the SMS as depicted in Figure 5.

A MPT is an abstract motor plan that resides in an agent's long-term memory (Sensory Motor Memory in LIDA). It has a set of motor commands that are not yet bound with the command values; after a specification process, the motor commands inside the MPT are bound with specific command values, instantiating the MPT into a concrete MP. MPTs and MPs have very similar structures, so they will often be designed with nearly the same data structure. Their major differences are 1) an MPT is persistently stored in a long-term memory, while an MP is short-term, and created anew each time it is used; and 2) most of an MP's command values have been specified, while those of an MPT have not.

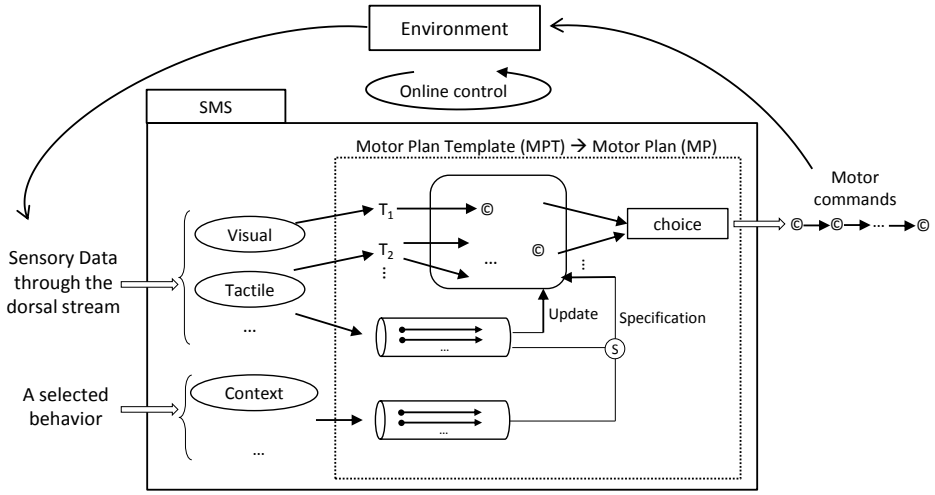


Figure 5. A MPT → MP, online control, and specification diagram

Both sensory data through the dorsal stream and the selected behavior determine the specification process. As shown in Figure 5, two cylinders lie under the set of motor commands (©s); they receive the sensed data and the context of a selected behavior separately, and provide the specific command values to motor commands mainly through a specification process. Each of these cylinders represents a set of associations; every association transforms relevant environmental features into a command value. As an example, in a grasping task, "the hand pre-shapes during reaching…. The pre-shaping of the hand includes the well-known phenomenon of 'maximum grip aperture' (MGA), whereby the finger grip opens more than required by the size of the object, but proportionally to it" (Jeannerod, 2006, p. 5). Thus, one corresponding association implemented in the SMS is to transform an object's size to the distance between gripping fingers.

The data sensed through the dorsal stream provides environmental features' true value, such as a numeric value of positive five as an object's width, while the context of a selected

behavior supports the semantic values "*large*" or "*small*" for the object's size. Usually, the command values specified in the motor commands are only relying on the sensed data, although the context affects the command values in a few conditions (Milner & Goodale, 2008). We have simulated some of these conditions and replicated the effects on the command values from variation of both the sensed data and the context. Accordingly, to implement the relationship of the effects of sensed data and the context, a suppress operation is represented by an encircled uppercase S in Figure 5: the command values associated with the sensed data usually suppress the values associated with the context unless either 1) there is a delay on the sensed data, 2) the association transforming the certain sensed data is not available—unfamiliar action, or 3) relevant objects "need to be analyzed for their semantic or material properties" (Milner & Goodale, 2008, p. 780).

The specification process is supposed to specify a MPT into a MP before the execution begins. The motor commands (MCs) inside a MPT are bound with specific command values during the specification process. However, there are some types of MCs whose command values are conceptually specified in the process of online control but not in the specification process. In the example of gripping an object, the individual finger's force is manipulated, not before the action execution begins, but in the course of the execution process (Grafton, 2010). To model this situation in the SMS, the pertinent command values are set with a default value in the specification process first, and are then updated in the online control. An *update* process is represented in Figure 5, showing that the MCs command values are updated by the values associated with the sensed data in executing the action. We will see a simulation illustrating this case later.

The subsumption architecture sends out a command with a fixed built-in value, telling the agent what to do in every moment. However, the subsumption architecture's reactive behavior is typically classified as a process to answer the "how to do" question of an action. This is due to the fact that the commands the architecture sends out are low-level, the same as the elements of environmental data, which cannot be directly recognized by humans, since they cannot describe what the commands are, though they know a high-level process has been accomplished in some way. Similarly, as shown in Figure 3, a MP outputs the motor command in each moment, answering "what to do" in a low-level way to the agent's actuators, and "how to do it" at a high-level responding for the goal-directed action initiated internally from the agent. Here the specification process operating on a motor command—binding a specific value to it—augments the principle of the subsumption architecture by extending the motor command with a specified value—besides "to do what", answering a question of "how much". Therefore, we argue that the SMS, a mechanism containing both online control and specification processes as shown in Figure 5, answers the "how to do" of an action at both low and high levels.

Furthermore, the SMS is not merely a reactive structure, adapting to its environment, but also a structure that responds to the intent of a goal-directed action. A selected behavior's context also affects the specification process. The high-level understanding necessary for an action's effectiveness in an external environment, such as the target object features of a grip action, can indirectly affect a MPT by specifying the values of some variables. In this way, the SMS may serve as a sub-module of LIDA, a systems level cognitive model that covers the whole cognitive loop from perception to action. The SMS connects LIDA's selected behavior to a MPT, thus achieving the action's execution.

MPT Selection

A MPT awaits initiation by an incoming selected behavior before being specified into a concrete motor plan. From a general engineering viewpoint, a special process called *MPT selection* has been created. As depicted in Figure 6, MPT selection chooses one MPT from others also based on the selected behavior.

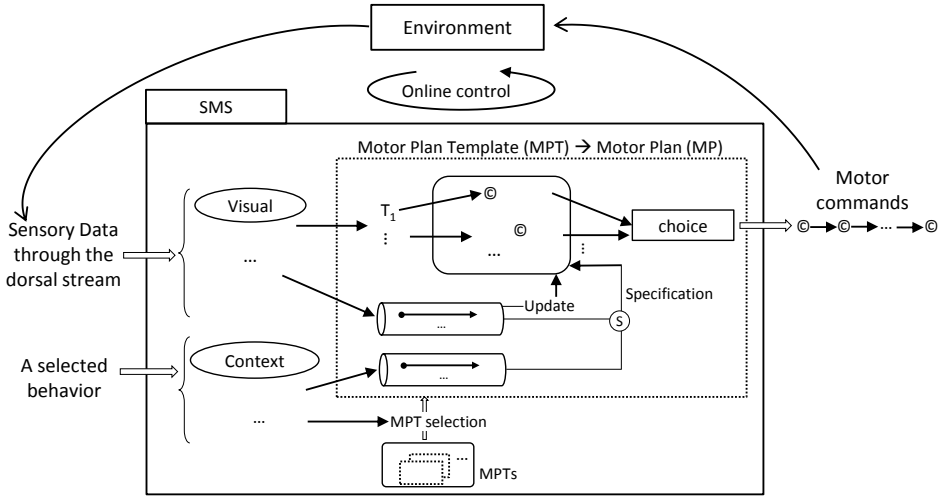


Figure 6. SMS with all of its components. See text for details.

The selected behavior and the dorsal stream

The design of the SMS's concepts has been fully described above, and is summarized in Figure 6. The three processes modeled inside the SMS—MPT selection, specification, and online control—are affected by the selected behavior and/or dorsal streams. Their detailed relationships are shown in Table 1 below.

Table 1. The selected behavior and dorsal stream affect the SMS's processes

	MPT selection	Specification	Online control
The selected behavior	Affect	Affect	Affect
Dorsal stream	N/A	Affect	Affect

Note that the selected behavior affects the online control process because it conceptually initiates the action execution, although it is not directly involved in the online control.

Implementation and Experiment

Different actions execute variously, due to vastly different actuators, goals, or action execution contexts. In other words, each action needs a certain Motor Plan Template (MPT) embedded in a Sensory Motor System (SMS) that allows the modeling of the action's distinctive characteristics in the execution process.

We have implemented a MPT in a newly created SMS to model the execution of a grip action inside a LIDA-based software agent. This MPT's design is both guided by the Herbert arm controller (Brooks, Connell, & Ning, 1988; Connell, 1989a), and biologically inspired by some hypotheses regarding the execution of human's grip action. The simulated results are compared with both robotic and human data.

The involved software agent (robot and its controller) and its experimental environment are introduced in the first two subsections, followed by a description of the simulation of Herbert's arm controller and its biologically inspired modification. Finally, the SMS is linked to LIDA where one experiment—regarding the awareness an agent to its action execution—has been done.

The LIDA Framework and Webots

The LIDA Framework is an underlying computational software framework. "[It] allows the creation of new intelligent software agents and experiments based in [sic] the LIDA Model. Its design and implementation aim to simplify this process and to permit the user to concentrate in [sic] the specifics of the application" (Snaider, McCall, & Franklin, 2011, p. 141).

Webots is a mobile robot simulation software package. It offers a developmental environment for rapid prototyping of 3D virtual worlds, an array of ready-made sensors and actuators, and programmable controllers that control a robot living in that world (www.cyberbotics.com). We use Webots as an experimental environment in which to create an agent developed using the LIDA Framework for running our computational SMS. The technical issue regarding the use of the Framework as a controller for a Webots robot has been addressed with a customized environment module as an interface as shown in Figure 7. Traditionally, the world, the robot and robot's controller are developed inside Webots. We created a simple robot controller inside Webots which responds to start a LIDA Framework as a real controller only. Rather than a typical Webots controller, the Framework serves as a robot controller by way of its customized environment.

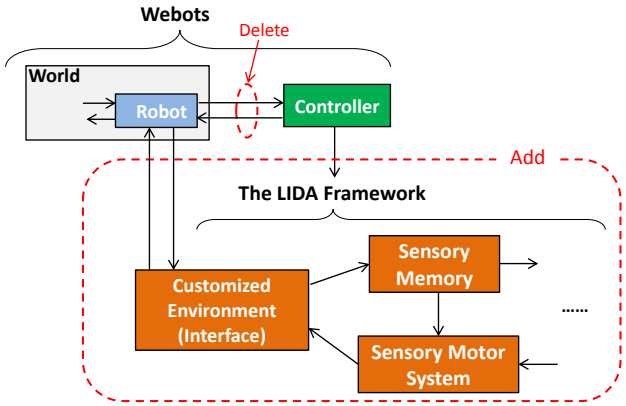
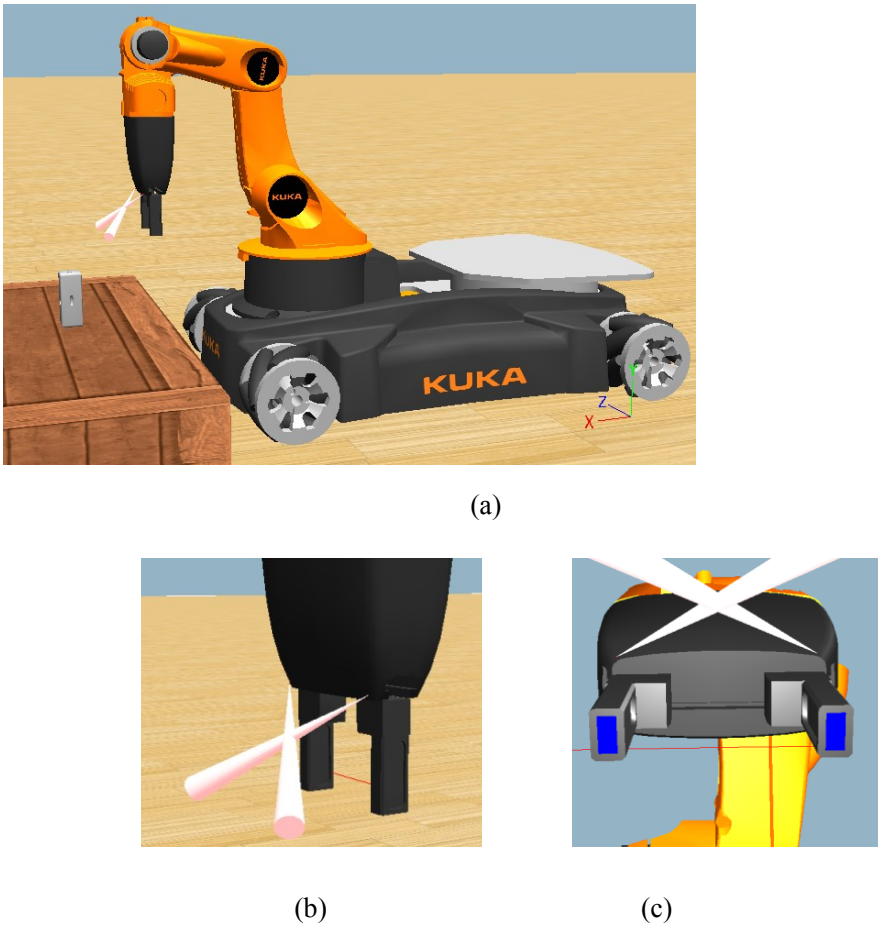


Figure 7. The LIDA Framework controlling a Webots robot

The extended youBot

The youBot is a software robot bundled with the Webots installation. As shown in Figure 8 (a), its actuators are a mobile base, an arm, and two grippers; the end segment of the arm plays the role of a hand and the grippers are attached to it. We chose this robot on the basis of its

similarity to Herbert, whose arm controller serves as the prototype of a Grip MPT inside our newly created SMS.



(a)

(b) (c)

Figure 8. (a) The extended youBot, (b) infra-red beams on the hand and between the grippers, and (c) touch sensors (dark blue, bottom view).

The sensors

Following the configuration of sensors in Herbert, we extended the youBot sensors by additionally simulating two infra-red (IR) beams detecting the area in front of the hand, one IR beam between the grippers as their closing trigger, and a touch sensor on the tip of each gripper. See Figure 8 (b) and (c) for details. The extended youBot sensors are introduced in detail in Table 2.

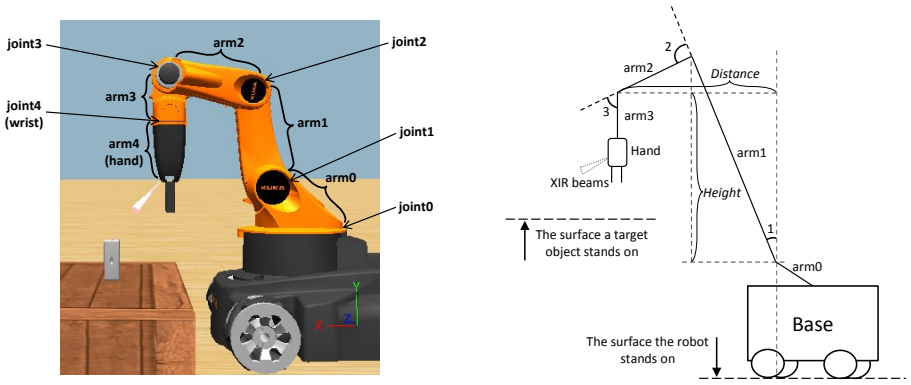
Table 2. The extended youBot sensors

Short Name	Description
Pos	The arm segment positions, which can be accessed by the Webots getPosition() method.
Tact (wrist)	The force exerted on the wrist, the joint between penultimate and last arm segments, which can be accessed by the Webots getMotorForceFeedback() method.
Tact (touch)	The data sensed through the touch sensors, which can be accessed by the touch sensor's getValue() method.
Beam	The distance as measured by the infra-red beam between the grippers. It is used to check whether an object is in between, and is accessed by its getValue() method.
XIR	The distances as measured by the two infra-red beams from the hand to any object which is in the area in front of the hand. It is used to check whether an object is in the area, and can be accessed by the getValue() method.

The actuators

As shown in Figure 9 (a), the youBot arm comprises five segments—from arm0 to arm4, which are linearly connected by five joints—from joint0 to joint4. The joints' angles can be modified by the Webots setPosition() method. Of these five joints, only the middle three—joint1, joint2, and joint3—are changeable in our simulation of grip in a vertically oriented X-Y plane. The first and distal joints don't act in the same plane—they rotate in X-Z plane; thus these two

joints, joint0 and joint4, have not been used, but just being fixedly set to value of zero. Accordingly, arm0 is considered part of the robot base, and arm4—hand is considered part of arm3. Because of the simplification, not all joints are explicitly shown in Figure 9 (b); the reader might have to go back to see Figure 9 (a) for details when we are talking about some formula based on Figure 9 (b) later.



(a) (b)

Figure 9. An extended youBot controlling its arm during a grip action

One important gripping issue is that the hand must be vertical to the surface that a target object stands on. This requirement follows from the fact that the XIR beams that detect the area in front of the hand are at a fixed angle to the hand. This constitutes a considerable simplification of the analogous human behavior. The human chiefly acts upright on land, however, a human's trunk is usually not exactly vertical but slightly forward leaning during motion, such as running; the line of sight adjusts dynamically—XIR does not—so that the human looks ahead constantly while the trunk's angle may vary from moment to moment.

As shown in Figure 9 (b), the hand position is controlled by the angles of joint1, joint2, and joint3—∠1, ∠2, ∠3; their sum must equal π to satisfy the constraint of that the hand being vertical to the surface[2], as described by Eq. (1):

$$\angle 1 + \angle 2 + \angle 3 = \pi \tag{1}$$

The hand has four basic movements: *lift*, *descend*, *extend*, and *back*, each of which can occur along one of two lines: up-down or back-forth. As shown in Figure 9 (b), regarding the up-down line, the parallel distance between joint1 and joint3, hereafter referred to as *Distance* and expressed by Eq. (2), must remain constant, where arm1L and arm2L represent the length of arm1 and arm2 respectively.

$$Distance = \text{arm1L} * \sin\angle 1 + \text{arm2L} * \sin\angle 3 \tag{2}$$

The constraint on *Distance* is expressed by Eq. (3), where ∠1', and ∠3' represent the measures of the updated angles after the execution of an up-down movement.

$$\text{arm1L} * \sin\angle 1' + \text{arm2L} * \sin\angle 3' = \text{arm1L} * \sin\angle 1 + \text{arm2L} * \sin\angle 3 \tag{3}$$

Particularly for the movement of *lift*, it is apparent that ∠3 must increase on the basis of this constraint. We have chosen 0.04 radians as an increasing interval unit so that the movement velocity is moderate, which process is expressed by Eq. (4).

$$\angle 3' = \angle 3 + 0.04 \text{ radians} \tag{4}$$

Now, only the value of ∠1' is unknown in Eq. (3), so it is able to be resolved. Finally, the updated angle of joint2, ∠2', is resolved based on Eq. (1); therefore the *lift* has been computationally simulated. Similarly, based on Eq. (1), (3), and (5), ∠3 decreases, the movement of *descend* has been simulated.

$$\angle 3' = \angle 3 - 0.04 \text{ radians} \tag{5}$$

[2] A precondition has been satisfied that the surface a target object stands on, and the surface the robot stands on are parallel.

Regarding the back-forth line, as shown in Figure 9 (b), the vertical distance between joint1 and joint3, hereafter referred to as *Height* and expressed by Eq. (6), must remain constant. The constraint on *Height* is expressed by Eq. (7). Similar to the up-down line, the back-forth movements *extend* and *back* are simulated based on Eq. (1), (7), and (4) or (5) respectively.

$$Height = arm1L * \cos\angle 1 - arm2L * \cos\angle 3 \quad (6)$$

$$arm1L * \cos\angle 1' - arm2L * \cos\angle 3' = arm1L * \cos\angle 1 - arm2L * \cos\angle 3 \quad (7)$$

Complicated movements are simulated as well, which extend the basic ones. One of these complicated movements is to move forward and slightly down; its simulation formula is developed from the basic movement *extend*, but instead of being constant, *Height* needs to slightly decrease as expressed by Eq. (8), where *Height'* represents the measure of the updated vertical distance between joint1 and joint3. The variable *A* represents the ratio between *Height* and *Height'*, set to be 0.95 in the simulation.

$$Height' = A * Height \quad (8)$$

The constraint on the change in *Height* is expressed by Eq. (9). Equations (1), (4), and (9) computationally simulate moving forward and slightly down. Other complicated movements have been simulated by the same strategy.

$$arm1L * \cos\angle 1' - arm2L * \cos\angle 3' = A * (arm1L * \cos\angle 1 - arm2L * \cos\angle 3) \quad (9)$$

One special case is to carry the hand back to its home position when the target object is held or the arm is stuck. Because the target position is already known, we just adjust the values of $\angle 1$ and $\angle 2$ to approximate their target positions within a reasonable interval, such as 0.04 radians; and the value of $\angle 3$ is passively changed according to Eq. (1).

The simulation of Herbert's arm controller

We have created a Motor Plan Template (MPT) in a new SMS to model a specific execution for a grip action inside a LIDA-based software agent. As described in the above section, this agent involves two types of actuators, the hand and the arm. The hand consists of two grippers simulating the thumb and the index fingers separately, and the arm, multiple segments being linearly connected by joints. The action's goal is to grip an object in the context of the current environment. We borrowed the design principles of the arm controller of a robot, Herbert (Connell, 1989b). Herbert "… is a completely autonomous mobile robot with an onboard parallel processor and special hardware support for the subsumption architecture …"(Brooks et al., 1988, p. 1). Its arm controller drives the robot to pick a soda can up and bring it back to a home location (Connell, 1989a).

Computational design

Three types of arm controller components have been modeled: the module (M), the suppress node (S), and the wire (W). The module is conceptually similar to the Augmented Finite State Machine (AFSM) used in a standard subsumption architecture (Brooks, 1986), although they differ in details (Brooks, 1991; Connell, 1989b). Regarding subsumption architecture's two grounding processes, suppress and inhibit, only the suppress node was needed for Herbert's arm controller. Hardware wires are simulated as computational components to link between modules and suppress nodes. In this way, a module or a suppress node doesn't necessarily have a fixed source or destination; it can be connected later during implementing the execution for a concrete action. Therefore, these three components are not limited to the simulation of Herbert's arm controller; they can be used to implement other types of subsumption architecture as well.

The three components shown in Figure 10 illustrate how they look and their constituent parts. Each of them has a core routine and I/O methods. The core routine executes as an independent task, which behaves according to different procedures (algorithms) among the three different components. The module (M) core routine acts like an AFSM in the subsumption architecture; it switches among multiple prepared states depending on the current state and the input sensory data, sending out a motor command when it stays in a certain state. The core routine in a suppress node (S) exactly simulates the suppress process in the subsumption architecture; it copies the input data coming through the higher layer to the output if the data is not empty, otherwise just copies the lower layer's data. The wires (W) core routine simply conveys a data copy from input to output.

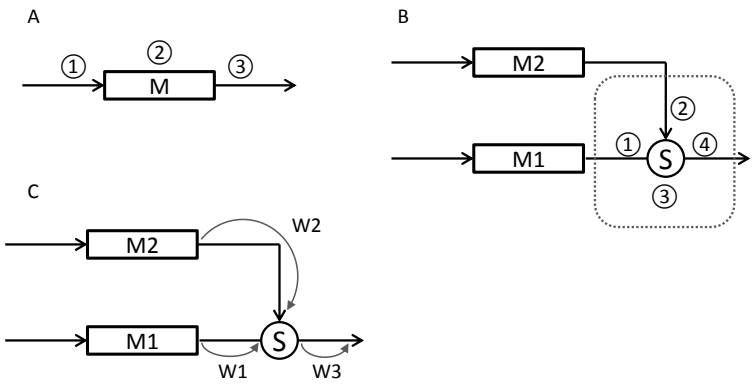


Figure 10. (A) An example of a module (M), where ② represents the core routine, while ① and ③ stand for I/O respectively. (B) A suppress node is boxed up by dotted lines. ③ is the core routine. The remaining parts stand for I/O: the lower and higher inputs are represented by ① and ② respectively, and ④ is the output. (C) W1, W2, and W3 are simulated wires linking modules and suppress nodes. A wire's core routine copies the data from input to output.

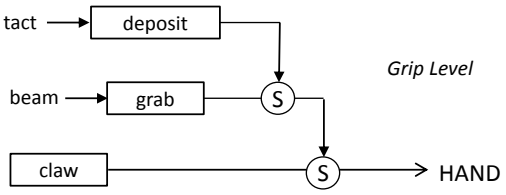
These core routines are computationally implemented as *LIDA-tasks* (Snaider et al., 2011) in the simulated Herbert's arm controller. A LIDA-task encapsulates small processes, and has implemented multithreading support, so that the core routines are able to operate in parallel

and execute independently. On the other hand, the I/O methods of the components are implemented by regular programming language methods rather than LIDA-tasks; therefore these I/O methods need to be invoked by wire components so that the modules and suppress nodes are linked (see Appendix A for their pseudo codes).

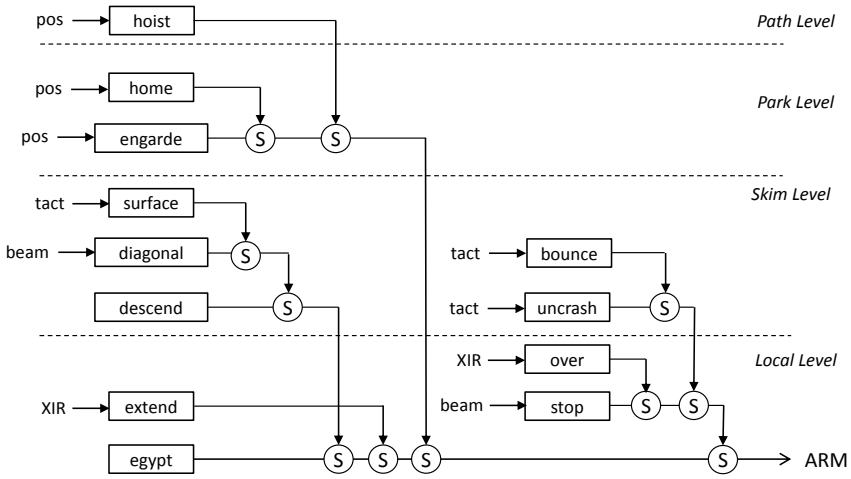
The design of the simulated Herbert's arm controller is shown in Figure 11, redrawn from the original Herbert's subsumption diagrams (Connell, 1989b). Sensory data enter from the left; output commands are sent out on the right. Modules, suppress nodes, and wires are structured into multiple levels (layers), bottom-up ordered by their priorities. The module name briefly indicates the associated behavior while in Figure (a), the *claw* module instructs the fingers to stay wherever they happen to be, and in Figure (b), the *egypt* module freezes the arm in whatever awkward angular configuration it happens to be in at the time (Connell, 1989b). A level's name expresses a behavior-task, also called a competence, which is achieved according to the combination of its modules and suppress nodes' behaviors.

A Grip Motor Plan Template (MPT) was created to maintain these modules, suppress nodes, wires and their organizations, to simulate Herbert's arm controller (see Appendix B for the simulation's software architecture). This Grip MPT is embedded into a newly created SMS and stored in long-term memory. The SMS receives sensory data from LIDA's Sensory Memory into Grip MPT's module components, and also passes the MPT's output commands issued by modules or suppress nodes to the outside, typically the environment in which the LIDA agent finds itself. At the present time, the MPT implementation is simply based on a robotic Herbert arm controller: the motor commands inside the MPT are fixedly bound by default values but not specified at run time, so that the current Grip MPT is conceptually equivalent to a MP as well as

shown in Figure 3. In accordance with SMS's biological inspiration, a specification process will be added to the MPT in a later implementation, to be described in later.



(a) The hand system



(b) The arm system

Figure 11. Simulated Herbert's arm controller consists of (a) hand and (b) arm systems, redrawn from the original Herbert's subsumption diagrams (Connell, 1989b). Compare to the original diagrams, three changes in the simulation are as described below.

1) In (a), a *cradle level* was removed. Because the upper bound on force to the actuators is configured into the simulated environment Webots, the cradle level is unnecessary.

2) In (b), a *back* module was removed. Since Herbert's base controller is not modeled, it is impossible to simulate an arm rotation to centralize the target object, and thus it is not necessary to check whether there is a lateral offset between the hand and the object (the purpose of the *back* module). We assume the target object is already centered with respect to the hand.

3) In (b), an *edge* module was removed because its function was conceptually combined within the *hoist* module.

Experiments

This SMS has been implemented within a LIDA-based software agent. In this section, two grip experiments from the original Herbert (Connell, 1989b) have been replicated, investigating the controller's reliability and flexibility as shown in Figure 12 and Figure 13, respectively. The simulation successfully replicates the online control of a grip execution driven by the simulated Herbert arm controller, lending support to the idea of utilizing the subsumption architecture as a prototype for an SMS model of the action execution process. Furthermore, we have reviewed two additional grip experiments to verify the agent's proper functioning in a range of situations.

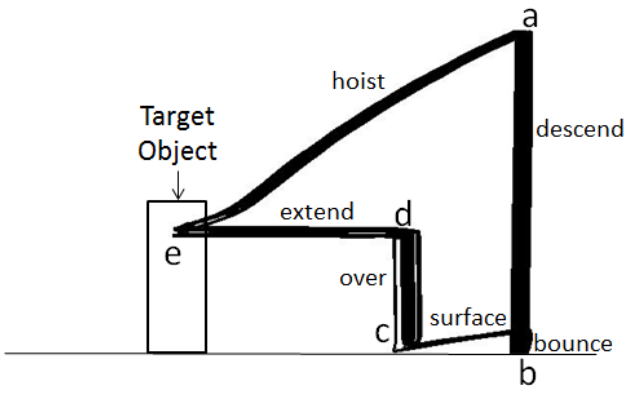


Figure 12. A composite of the grip trajectories produced by the simulated arm controller on 10 consecutive runs

First, the results shown in Figure 12 speak to the reliability of simulated controller's behaviors. The lines show the composite trajectories followed by the tips of grippers during 10 consecutive runs of the simulated arm controller. The sequences of gripper tips positions are recorded by a *Supervisor*[3] in Webots at the run time. During each trial, the hand descends from point a, and then traverses points b, c, and d exploring for the object: first doing a small bounce

[3] The Webots Supervisor "is a privileged type of Robot that can execute operations that can normally only be carried out by a human operator and not by a real robot" (www.cyberbotics.com). It is irrelevant to the machine learning concept of supervised learning.

at point b when it touches the ground surface, and going forward and slightly downward to skim the surface, then lifting above it and extending when it finds that the object is in front of it. The grippers reach the object at point e and finally carry it back to point a by performing a 'hoist' task.

During these 10 runs, the agent's position changes slightly due to reactive forces produced by the grippers' contact with the ground, and the object; thus, the trajectories are not exactly same between the runs since they are sensitive to initial conditions such as the agent's position. These differences between the trajectories were not expected in this experiment's design; whereas a realistic robot experiment involves physical noise, our software agent's body and its environment have been deterministically simulated. This unexpected result supports the existence of an effect originating from within the agent itself, such as its action.

Second, the same controller is used in different environments to verify its flexibility. Figure 13 (a) shows a trial in which the target object lies on a pedestal rather than directly on the ground. The hand starts in the same way as in the previous experiment, finds the surface and begins to skim along it. However, at point c, it detects an object (the pedestal) but fails to grip it. This attempt results in a tactile input to the agent's wrist, activating the task "uncrash". "Uncrash" performs a function similar to "bounce" but for a vertically oriented surface: the grippers move away from the pedestal and lift (Connell, 1989b). After the grippers are above the pedestal surface, it executes the remaining portion of the grip action as in the previous case.

Figure 13 (b) shows a case with the target object behind a barrier. Again, the hand touches the top of the barrier first and then goes forward skimming it. The change of surface is not noticed by the agent so it proceeds with the rest of the grip as before.

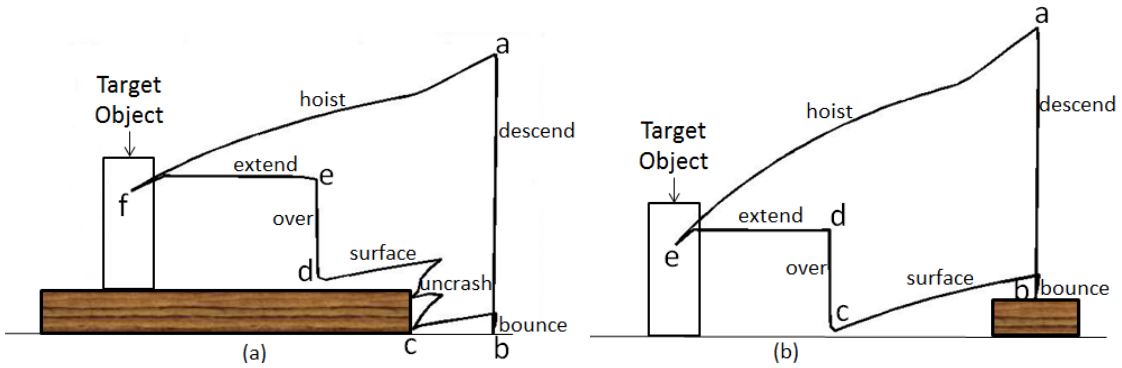


Figure 13. The simulated arm controller grips an object which is (a) on a pedestal or (b) behind a barrier.

Besides the replications of the Herbert arm controller experiments, additional experiments have been performed. Figure 14 (a) shows the same controller gripping a small object; in this case the agent skims the surface more, but lifts and extends less than when gripping the taller object (described previously). The skim is achieved by the combination of multiple "bounce" and "surface" tasks. In Figure 14 (b), no object is available for the hand to grip. The grippers reach the ground first and then begin to skim along it as in the previous Figure 14 (a); however, no object has yet been found, so the whole arm is stuck at point c for a while, and after that the grippers are retracted back to point a. The task "hoist" does this retracting the same as when it carries back a gripped object.

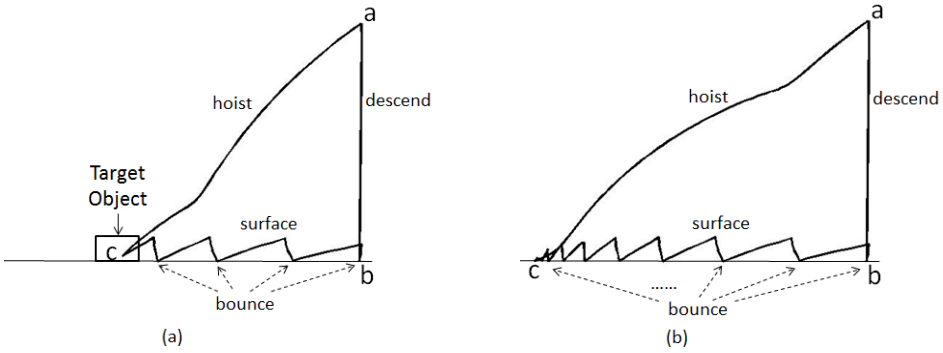


Figure 14. The simulated arm controller (a) grips a small object or (b) fails to grip.

Biologically Inspired modification

The simulated Herbert arm controller has been modified based on the SMS concept as described above. Instead of default values, the motor commands inside a MP are bound with specific command values through a newly created *specification* process before the action execution begins, or a new *update* process at run time; thereby a new grip Motor Plan Template (MPT) conceptually exists before its motor commands are bound. Two sets of associations are created. In each of them, a single type of association is implemented, transforming the object's width into a command value—the distance between the grippers, its aperture. Some human experimental results regarding action execution have been compared with these simulated results.

First, the grip action is executed using the unmodified arm controller as an experimental control. As shown in Figure 15 (a), the agent's grip aperture is sampled at unit intervals in Webots virtual time during the grip execution. Whatever its starting value, the grip aperture almost always reaches 0.0656m (the maximum grip aperture, or MGA) before the grip closes around the target object. The grippers squeeze the target object, and thus the resulting grip aperture is smaller than the original target object width.

Second, an association (the upper cylinder in Figure 5) has been implemented by connecting the sensed object's width through the dorsal stream to the value of the grip aperture. Its transformation formula is expressed by Eq. (10):

$$\text{Grip aperture} = \text{Object's width} * \text{Magnification} - \text{Grippers' gap} \qquad (10)$$

The variable *Object's width* is a numeric value representing a true width directly sensed through the dorsal stream from the environment. *Magnification* is used to set the grip aperture to be slightly greater than the object width, set to a value of 1.2 in the simulation. A small gap

between closed grippers is available, which is substituted from the expected grip aperture (Object's width * *Magnification*) to reach an actual grip aperture necessarily being sent to the grippers. We are well aware that this formula can be improved in numerous ways, for an example, using a more complicated formula to represent the *Magnification* instead of using a variable only, or including additional parameters. As shown in the experimental results introducing this section, this simple formula is effective, and we leave for future work the discovery of various methods for improving it.

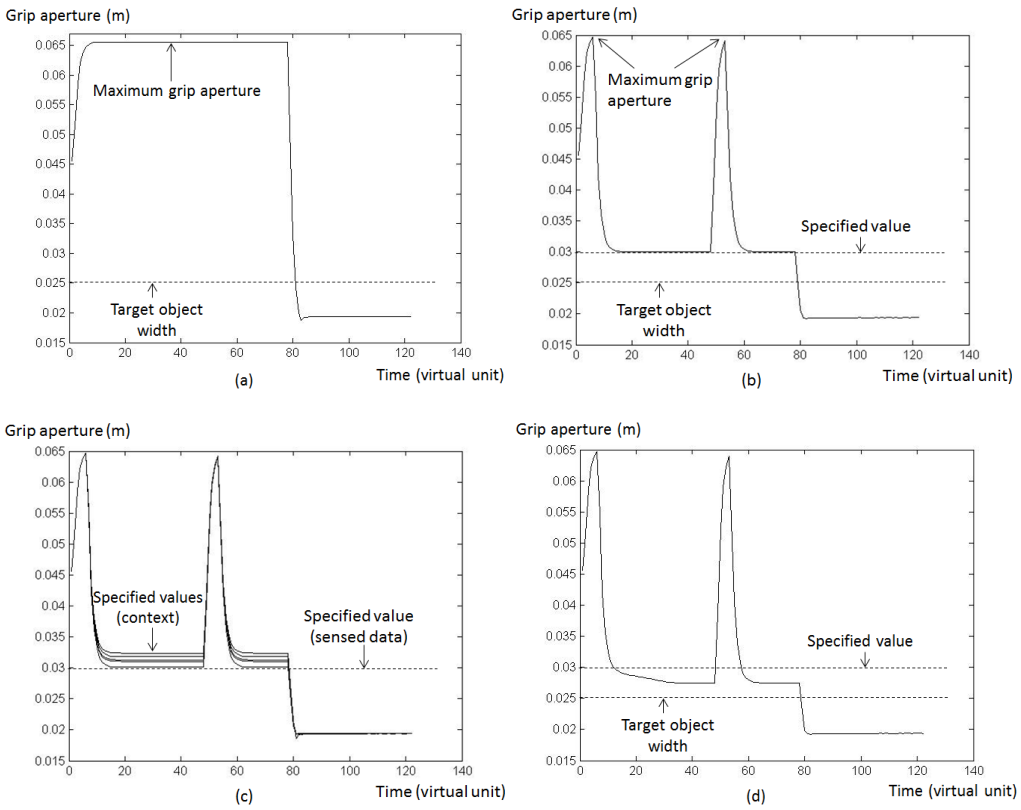


Figure 15. The agent's grip aperture is sampled at unit intervals in Webots virtual time during the grip execution.

As shown in Figure 15 (b), the grip aperture typically reaches the specified value of 0.03m before the value falls as the grippers close (see below for an explanation of the two peaks

in the aperture). Compared to the maximum grip aperture (MGA), which is a fixed aperture value for Herbert's grip, the value specified here is much closer to the target object width of 0.025m. This calibration results from the implementation of the association through the dorsal stream. This simulated result supports—and is qualitatively the same as saying—that "the dorsal stream plays a central role in the programming of actions (i.e. the pre-specification of movement parameters)" (Milner & Goodale, 2008, p. 776), as supported by evidence from observations of the patient D.F. (James et al., 2003; Milner et al., 1991). The specified value in the simulation is larger than the object width: 0.03m > 0.025m, since experimentally, "the finger grip opens more than required by the size of the object" (Jeannerod, 1981, 2006). The first MGA peak is modeled by setting a fixed MGA value to the grip aperture for a short while when the execution starts, in keeping with the observed human behavior (Farnè, Pavani, Meneghello, & Làdavas, 2000; Jeannerod, 2006). The second MGA peak occurs because the grippers touch the surface; the grip aperture is set to become maximal in this situation so that its behavior can track the object's width value as well as adapt to an unpredicted collision.

Third, another association has been implemented by connecting the object width represented in the context component of a selected behavior to the value of the grip aperture (Figure 5, bottom cylinder). Its formula is expressed by Eq. (10) as well, but the variable *Object's width* has a different meaning here. Since the object width represented in the context is a semantic value, such as "large" or "small," which are not precise, its value is designed to be distributed in a range. We simulated this dispersion in the association's transformation according to Eq. (11), where the *approximate rate* is a random value set to be in a range of 1.0 ~ 1.1 in the simulation, so that the object width approximates its true value.

$$\text{Object's width} = \text{Approximate rate} * \text{(True) object's width} \quad (11)$$

Instead of the data being sensed through the dorsal stream, the selected behavior's context affects the relevant command values in several conditions (Milner & Goodale, 2008). We simulated two of these conditions: 1) Deleting the association implemented above which connects the sensed data to the grip aperture; in effect, it makes a skill unfamiliar to the agent, or 2) Terminating the relevant data sensed through the dorsal stream which simulates a delay in the sensed data. Five executions produced a range of context-specified values rather than a precise value as shown in Figure 15 (c). We argue that these imprecise movements result from an association from the selected behavior's context to a command value. This interpretation of the simulation results agrees with the conclusion we reached above that the dorsal stream plays a central role in specification process. Additional evidence is found in patients suffering from bilateral optic ataxia caused by damage to the dorsal stream—these patients show deficits in calibrating their grip aperture (Jakobson, Archibald, Carey, & Goodale, 1991; Jeannerod, Decety, & Michel, 1994; Milner & Goodale, 2008).

Fourth, an update process is implemented to update the grip aperture values during the execution. Its formula is expressed by Eq. (10) the same as the association which connects the sensed object width through the dorsal stream to the grip aperture; however, instead of a constant value, the *Magnification* here is set to be dynamically decreasing through the execution time. In Figure 15 (d), the updated value comes closer to the object width than the specified value; it follows that the sensed data provided through the update process are more precise than the context of the specification process, because the situation becomes clearer to the agent as it executes the action.

Linking the SMS to LIDA

As discussed above and shown in Table 1, both the data sensed through a dorsal stream channel, and a selected behavior corresponding to a goal-directed action are input to the SMS, and the SMS's output is sent out to Environment. The grip MPT is mapped one-to-one onto the action component of a selected grip behavior; this is a simple implementation of MPT selection following the SMS concept introduced above.

These I/Os are implemented in the LIDA-based agent including the SMS as shown in Figure 16. Only the related action selection and action execution modules—the latter being implemented by SMS—are represented. The other LIDA modules are abstractly represented by LIDA's understanding and attention phases.

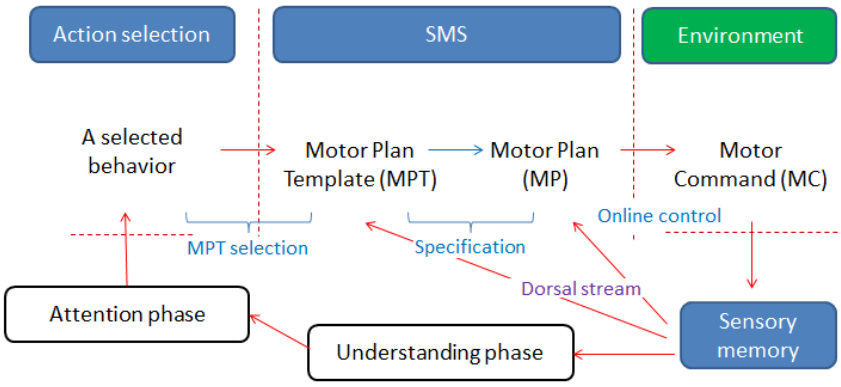


Figure 16. The SMS is embedded into the LIDA Model

Additionally, in order to let the agent monitor the execution status, an expectation codelet (Faghihi et al., 2012) is created when the grip behavior is selected in the action selection module; this particular attention codelet—a small and special purpose computational process—contains the expected result component of the currently selected behavior. It checks whether this result has been reached (sensed and recognized by the agent) at run time. The checking result is sent to

LIDA's Global Workspace module, where it competes for the agent's attention (Baars, 1988). In this way, the agent's awareness of its own action execution is indirectly achieved.

Four check-points have been set up in the expected result: with the grippers 1) in the initial situation, 2) in the final situation, 3) holding the target object, and 4) in a stuck situation. The checked result for these points comes to the attention of the agent if the result wins the competition during executing the grip action. This means that the agent is aware of some significant fragments of the action execution, although it has no idea what exactly it is doing in each moment.

Comparison

We realize that the action process we have developed in LIDA, specifically the action execution process implemented by the SMS, have the same general form as the action processes implemented in some of the other architectures, although they use different structures. This section compares action execution process deployment in three well-known cognitive architectures—their reviews have been given in Chapter 3 above—to that of LIDA. We argue that each cognitive architecture having a representation at an explicit level of knowledge, also needs a process that converts high-level knowledge to motor-level commands, as does the SMS.

ACT-R

Adaptive control of thought-rational (ACT-R) is a cognitive architecture, a theory for simulating and understanding human cognition based on numerous facts derived from psychology experiments (http://act-r.psy.cmu.edu/).

ACT-R doesn't differ from LIDA very much with respect to action. First, both of their action processes are conceptually designed with two steps: 1) the selection of a high-level action, and 2) the execution of low-level actions. In LIDA, a behavior is selected in the action selection

process based on the agent's motivation and its understanding of the current situation. Then in an action execution process—modeled by the Sensory Motor System (SMS), the behavior's action component is transformed into low-level motor commands that are executed in the real world through an environment module. Similarly in ACT-R, a production rule is selected and fired in the production memory, and responds to the patterns of information in the declarative memory buffers, after which the rule's actions request that movements be prepared and executed in the motor memory. The executed movements then control the devices acting in the world by utilizing a device module.

Second, the motor control systems of LIDA and ACT-R, the SMS and the motor module respectively, are structured similarly. In the SMS, a Motor Plan Template (MPT) selection process acts first, connecting a selected behavior from the action selection process to a MPT inside the action execution process, and then the ensuing specification process specifies the values of MPT's variables, so that a Motor Plan is created for generating motor commands. In ACT-R's motor module, the preparation process acts first to connect the preceding production rule's action to a certain movement style, such as 'PUNCH' mentioned above, and then to specify the parameters necessary for the resulting movement execution.

Finally, the low-level actions are modeled with concrete examples of action in both cases: a grip in LIDA, and the manipulation of virtual keyboard and mouse in ACT-R.

In contrast, there are also several differences between LIDA and ACT-R's action processes. First, in LIDA the MPTs are prepared in advance in long-term memory; the operation acting on the MPTs is the selection so as to reuse the selected MPT. However in ACT-R, a movement is specified anew each time in the preparation process; the potential for reuse occurs only if the movement repeats the previous movement.

Second, in LIDA's SMS, an online control process is implemented, directly connecting the sensory data to the action execution process during the execution. However in ACT-R, there is typically no direct communication between the perceptual and motor modules[4]. The data passed to the motor module always comes from the high-level declarative memory. This is almost the only significant conceptual difference between LIDA's SMS and ACT-R's motor module.

Soar

Soar is a cognitive architecture in pursuit of general intelligent agents (J. Laird, 2008). Here we discuss the similarities between the action processes in LIDA and Soar. LIDA has action selection and action execution processes, while Soar has operator selection and operator application. Specifically, first, the multiple schemes of LIDA's Procedural Memory are recruited based on the most salient current situation. This recruitment is similar to Soar's operator proposal, in that both provide candidates for the action selection step that follows.

Second, in LIDA, before the process of action selection, recruited schemes are instantiated into behaviors. Additional information retrieved from the Current Situational Model is bound to the schemes' context and result components, so that actions that are more concrete, known as behaviors in LIDA, are created. Similarly, Soar's elaboration updates the proposed operators with additional detailed current situation information. Thus, in both cases the candidate high-level actions undergo an instantiation or elaboration that "pre-processes" them before the selection process.

Third, the selected behavior's action component is executed in LIDA's Sensory Motor System (SMS) by a particular Motor Plan (MP); while in Soar, the actions of the selected

[4] There is only very limited direct connectivity between perceptual and motor modules. Spatial information in particular is communicated directly.

operator are performed by production rules that match the current situation and the current operator structure (J. E. Laird et al., 2012).

On the other hand, there are some differences between LIDA's action execution and Soar's operator application. We conclude that in the case of Soar's output, motor commands, which cannot be directly performed (executed) on the external world, an external program is always necessary to handle the final "real" execution (performance) for Soar. In LIDA, however, its SMS responds by transforming the selected behavior into a sequence of executable motor commands, presumably in the real world. Note the term "motor commands" expresses completely different concepts in Soar and LIDA, although it is used to represent the final output data in both cases. In LIDA, motor commands are executable, while in Soar they are not.

Soar does not cover the representation of implicit environmental information related to action. This allows it to maintain generality with a clear standard, without the necessity of considering every possible domain that the Soar agent might live in. In contrast, LIDA emphasizes the biological viewpoint that an action execution process, which involves the consideration for domain details, is a reasonable part of an entire cognitive architecture, because the process of generating executable motor commands are not only driven by the low-level environmental implicit information but also initiated and affected by the agent's high-level explicit mental processes.

CLARION

CLARION stands for Connectionist Learning with Adaptive Rule Induction ON-line. The purpose of this architecture is to capture all the essential cognitive processes within an individual cognitive agent (Sun, 2003, 2006).

In CLARION, the action process introduced above has many concepts similar to that of the LIDA Model, though their terminologies and computational representations differ. First, the sensory data retrieved in LIDA influences the action process at two "levels". At one level, sensory data is filtered through the understanding and attention phases, and then helps recruit appropriate actions in the action selection process. At the other level, the data is sensed through a dorsal stream channel, directly connecting from the Sensory Memory to the action execution process implemented by the SMS. Similarly in CLARION, the action-centered subsystem (the ACS) connects the perceptual current state to actions through both the top and bottom levels.

Second, a direct (implicit) mapping from sensory data to action output is modeled in both LIDA and CLARION. In LIDA's SMS, the direct sensory data affects the generation of low-level actions. This process is implemented as a Motor Plan (MP) based on the principles of the subsumption architecture, a reactive structure. One critical feature of the subsumption architecture is that it doesn't maintain any central world state inside[5], and is without any explicit representations. Similarly in CLARION, the ACS's bottom level encodes implicit knowledge as mentioned above, which may be implemented in backpropagation neural networks[6], whose representational units in the hidden layer are capable of accomplishing tasks, but are generally not individually meaningful (Sun, 2003). Furthermore, the MPT in the SMS and the backpropagation neural network in the ACS both have the potential for multiple instances, and a selection process is proposed for both the MPT and the backpropagation neural network.

[5] Although no central world state is one of the essences of the subsumption architecture, implicit understanding and expectation of the environment has been built into the architecture by its layered structure.

[6] There is the learning of implicit knowledge (the backpropagation network) at the bottom level. "In this learning setting, there is no need for external teachers providing desired input/output mappings. This (implicit) learning method may be cognitively justified" (Sun, 2006).

Third, the interaction between the two levels is modeled in both LIDA and CLARION. The output of LIDA's action selection process, known as the selected behavior, is linked to a MPT, mapping from a semantic action to concrete ones. In CLARION, the input state or the output action to the bottom level is structured using a number of input or action dimensions; each of the dimensions has a number of possible values. At CLARION's top level, an action rule's condition or action is represented as a chunk node which is connected to all the specified dimensional values of the inputs or actions at the bottom level (Sun, 2003). CLARION models an interaction between the top and bottom levels, as well as between explicit and implicit knowledge.

On the other hand, action processes modeled in LIDA and CLARION are also different. The two levels of LIDA's action process—action selection and action execution—work interdependently, and operate linearly. A selected behavior, the output of LIDA's action selection, is not executable directly on the environment, but is used to initiate certain processes operating in the concomitant action execution process, ultimately generating executable low-level actions as a sequence of motor commands. However, the two levels implemented in CLARION's ACS operate independently: each of them makes action decisions based on the current state in parallel. The action sent out from both top and bottom levels are all performable. The final output action of the ACS is the combination of the output actions from the top and bottom levels.

Conclusion

Based on the LIDA Model, the subsumption architecture, the two visual systems, as well certain other cognitive neuroscience hypotheses, the Sensory Motor System (SMS) proposes a model of the human action execution process.

In the design of SMS, we have considered the subsumption architecture from a new viewpoint, namely, that its capabilities fulfill the hypothesis regarding the online control role of the dorsal stream. Second, we have modified the original subsumption architecture as inspired by certain hypotheses of cognitive neuroscience so as to combine a reactive structure with a goal-directed action. Finally, we have designed the SMS as a submodule of the systems level cognitive model LIDA, thereby rendering it capable of communicating with other cognitive modules naturally in a closed cognitive loop, from sensors to actions.

A computational SMS has been implemented for the execution of a grip behavior, and its simulated results have been compared to human data. Also, the SMS of LIDA has been compared to the action processes implemented in three of other cognitive architectures.

This biologically inspired design, together with a computational verification by the comparison of model and human behaviors, supports the SMS as a qualitatively reasonable cognitive model for action execution.

5. Estimating human movements

Introduction

The perceived visual world remains stable during ongoing eye and head movements. Yet a relatively brief, small, but unexpected visual change in the world may attract our attention explicitly. Jeannerod considers this stability a paradigm for the distinction between self-produced and externally produced changes in the world (2006). He argues that "a displacement of the visual scene is attributed to an external change, not to a self-produced eye movement." (2006, p. 18)

Jeannerod hypothesized that a functional model, the efference copy (Von Holst & Mittelstaedt, 1950), disentangles the changes in the world produced by self-movement, from externally produced changes (2006).

Von Holst and Mittelstaedt hypothesized that each time the motor centers generate an outflow signal for producing a movement, a copy of this signal (the efference copy) is stored in a short-term memory. Afterward the relevant reafferent inflow signals—resulting from the movement and sensed by the agent (Franklin & Graesser, 1997)—are compared with the efference copy (1950). Note that the comparison is actually between the sensed inflow data and the desired estimate that is based on the relevant efference copy. If the two correspond, Von Holst suggests that they would cancel each other out so that there is no inflow data perceived (1954), a suitable situation for anticipating the sensory effects of a self-produced movement. On the other hand, if the actual movement departs from the anticipated one, it is likely due to an external cause (Jeannerod, 2006).

Wolpert and his colleagues (1995) have investigated a sensorimotor integration mechanism by which people produce an estimate of the result of their movement. They have hypothesized that the central nervous system (CNS) internally predicts the result of a self-

produced movement by simulating the dynamics of the environment[1] using a so-called (forward) internal model, which is driven by a copy of human motor commands, the efference copy. This prediction is then combined with a reafferent sensory correction (1995). To test this hypothesis, they have simulated this prediction and correction using the Kalman filter (Kalman, 1960). In this way, they qualitatively replicated how humans estimate their hand movements in the dark.

The question of whether combining such an internal model with sensory correction is in fact neurally implemented in humans, or is just a metaphor for what the human nervous system does, remains open (Grafton, 2010). However, this model is useful for studying further hypotheses, including Bayesian decision theory for sensorimotor control (Körding & Wolpert, 2006), optimal feedback control (Todorov & Jordan, 2002), and motor recognition (Jeannerod, 2006). Moreover, the Kalman filter itself has been applied in different domains in other fields, together with its extended version: extended Kalman filter (EKF) (Auger et al., 2013).

Following the example set by neuroscience researchers (Körding & Wolpert, 2006; Todorov & Jordan, 2002; Wolpert et al., 1995), we embed estimation into the Sensory Motor System (SMS) (see Chapter 4 for details) by implementing a Kalman Filter (Kalman, 1960) as the "core engine" of the forward model's estimation process.

In the Kalman Filter, there are two factors that balance the importance between predicted results and sensory results: the inaccuracy in the knowledge of the dynamics of the environment, and the noise in the sensory process.

We introduce a third balancing factor, changes in the environmental dynamics. Actually, humans may experience, and then remember, such changes as a kind of error, the difference between intended (predicted) results and actual (sensory) results. We propose that this new factor is driven by memory of errors caused by changes in the dynamics. This idea is inspired by a

[1] The environment includes both the agent's motor system (body) and the world the agent lives in.

recent study in sensorimotor learning (Herzfeld et al., 2014). Herzfeld and his colleagues hypothesize that besides learning from errors, the brain may decide how much to learn from a given error depending on its memory of errors. These historical errors help humans determine whether the environment is steady or rapidly changing. Environmental stability thus controls how much of a given error will be learned so as to affect the estimate of the upcoming movement.

In the following section we describe and compare the studies of Wolpert et al. (1995) and Herzfeld et al. (2014). We then introduce our new model, a modified Kalman filter, which estimates human movements using memory of errors, and go on to describe a computational experiment that simulates hand lifting action.

Previous Work

In this section we first review a study regarding how people estimate their hand movements in the dark (Wolpert et al., 1995), and then introduce a recent study about how memory of errors affects sensorimotor learning (Herzfeld et al., 2014). Finally, we compare the two studies. In this way, we provide adequate background knowledge to prepare for the introduction of our new model in the following section.

Simulating a Sensorimotor Integration Process Using the Kalman Filter

Wolpert and colleagues (1995) have argued for the existence of an internal model in the central nervous system (CNS) that simulates the response of the motor system. They have carried out a human experiment in which participants move one of their hands horizontally on a plane either to the left or to the right along one dimension in the dark. In the absence of vision, their sensory feedback consists only of proprioception during the movement. Participants are instructed to continue moving until they hear a tone. The timing of the tone is controlled so as to

produce a uniform distribution in movement duration between 0 and 3 seconds. At the end of each movement (trial), participants indicate (estimate) the unseen new location of their moved hand. The difference (error) between participants' real and estimated new hand locations is recorded as a function of movement duration. In total, eight participants performed 300 trials each. In this experiment, researchers found that on average, 1) participants overestimate their hand locations—the estimated location is further than the actual—and 2) the error peaks after one second and then decreases gradually.

As argued by Wolpert et al. (1995), "these experimental results can be fully accounted for if we assume that the motor control system integrates the efferent outflow and the reafferent sensory inflow". To support this conclusion, they developed a computational internal model, with the use of a reafferent sensory correction, to replicate human self-estimation of hand movements in the dark using a Kalman filter.

The Kalman filter is a recursive algorithm that estimates the state of a discrete-time linear stochastic system (Kalman, 1960; Maybeck, 1979). It first predicts the system's next state in the timeline, based on its current state, on knowledge of the running system's dynamics, and optionally on its current motor command. Then it corrects the prediction based on sensory data that may have noise. This two-step routine operates iteratively online to estimate the system's state. From a mathematical viewpoint, the Kalman Filter is a set of equations that provides an efficient estimate for the state of a process, expressed by Eqs. (12) ~ (16).

$$\bar{x}_t = A x_{t-1} + B u_t \tag{12}$$

$$\bar{P}_t = P_{t-1} + Q \tag{13}$$

$$K_t = \bar{P}_t / (\bar{P}_t + R) \tag{14}$$

$$x_t = \bar{x}_t + K_t(Cz_t - \bar{x}_t) \quad (15)$$

$$P_t = (1 - K_t)\,\bar{P}_t \quad (16)$$

The Kalman Filter's prediction process is represented by Eqs. (12) and (13), and its correction process is represented by Eqs. (14) ~ (16). Variable x represents the state value. Specifically, x_{t-1}, $\bar{x}_t$, and x_t represent the immediately previous, intermediate predicted, and current estimated state values respectively. Variable u_t represents the value of input motor commands, and z_t the value of input sensory data. *A*, *B*, and *C* are the parameters for the above variables. *K* acts as a gain that weights the new sensory data against the predicted result. Parameters *Q*, *R*, and *P* represent the uncertainty of the prediction, the correction, and the entire estimation respectively. For further details, (Kalman, 1960), (Maybeck, 1979), or (Welch & Bishop, 2006) may be consulted. From one viewpoint, the Kalman filter is a kind of non-Markovian extension (Thrun, Burgard, & Fox, 2005) because its estimation relies on its historical data, while optimality is not of concern, and so is not guaranteed in our new model.

Based on the simulated results, Wolpert et al. (1995) have shown that the Kalman filter is able to qualitatively reproduce the propagation of the error of the estimated hand location as a function of movement duration.

A Memory of Errors

In the study of sensorimotor learning, Herzfeld and colleagues (2014) have hypothesized that the brain not only learns from individual errors that it has experienced before, but also accumulates the errors into a memory; this memory of errors makes it possible for the brain to control how much it will learn from a given current error.

Herzfeld et al. (2014) have done human experiments to explore the effect of memory of errors in human hand-reaching movements. The experimental setup is as follows. (1) A participant sits down in front of a table, and holds the handle of a robotic arm; the arm is attached on the table, and its handle can be moved because of several moveable joints in the arm. The participant is asked to repeatedly make out-and-back reaching movements; the goal for a trial is to reach a target location from an initial location. (2) The participant's hand is occluded by an opaque horizontal screen that is located above the plane of the forearm; thus the participant cannot see his hand. (3) An overhead projector displays information on the screen about the actual hand location, the initial location, and the intended target location of the reach. This information is visually available to the participants. (4) During a reaching movement (only on the outward reach), the participant's hand may be perturbed by the robotic arm through its handle with a force perpendicular to the reaching direction. The perturbation produces an error during the reaching movement, the difference between the intended hand location and the actual hand location upon arriving. (5) The magnitude of the perturbing force is constant, and the direction may be either to the left or to the right. Thus the force may create two types of errors.

Using this experiment, Herzfeld et al. examine the relationship between memory of errors, and the amount that is learned from a given error. They hypothesize as follows: "[Consider] an environment in which the perturbations persist from trial to trial, and another environment in which the perturbations switch … In a slowly switching environment, the brain should learn from error because the perturbations are likely to persist (learning from error in one trial will improve performance on the subsequent trial). However, in a rapidly switching environment, the brain should suppress learning from error because any learning will be detrimental to performance on subsequent trials" (Herzfeld et al., 2014).

In the experiment, participants are randomly divided by environmental stability into three groups (9 per group): they first performed 30 trials of reaching in either a slowly, medium-speed, or rapidly switching environment—the direction of the perturbing force switches. And then all participants experience a pure reaching movement without any perturbation for 10 trials[2]; in this way, the effects of the perturbation are removed. Finally, all participants experience one reaching trial with the same perturbation.

The researchers measured the change in the force applied by participants before and after the final perturbation. By considering the force produced by a participant, a proxy for the participant's estimate of the perturbing force, they can indirectly measure how much the participant's estimate of the perturbing force has been updated after experiencing a perturbation. They found that the responses of participants to the same perturbation are different between groups. A participant gives larger responses—corresponding to a higher estimate of the force—in the slowly switching environment and smaller responses—indicating a lower estimate of the force—in the rapidly switching environment. This phenomenon supports their hypothesis quoted above.

Note that in this experiment, although the memory of perturbation has been removed using 10 trials of pure reaching movements before measuring the effect of the final perturbation, a more abstract attribute of the environment, corresponding to a level of persistence of the environment—environmental stability, is still available in the memory, and thus influences the effect of the final perturbation. A term "saving" names the influence of this available abstract attribute.

[2] This pure reaching movement is known as a "washout" (Herzfeld, Vaswani et al. 2014).

Comparisons between the Two Studies

In this subsection, we compare the two studies reviewed above. To conserve words, we cite the two papers (Herzfeld et al., 2014; Wolpert et al., 1995) for the two studies respectively in the whole subsection here, and at times below we simply refer to the study of Wolpert et al. (1995) as the first study and to the study of Herzfeld et al. (2014) as the second study.

First, in both studies, researchers investigate the process by which humans produce an estimate of their movement. Wolpert and colleagues simulate how people estimate, using the Kalman filter, how their hand moves in the dark, and Herzfeld and colleagues propose a causal relationship from memory of errors to the knowledge of the environmental dynamics, which knowledge affects the estimation of upcoming movements.

However, the estimation processes examined in the two studies are at different levels. Wolpert and colleagues study the estimated hand location within a single movement trial. They calculated the propagation of estimation error on average for one movement, while they did not concern themselves with the relationships between multiple movement trials. On the other hand, Herzfeld and colleagues study the estimated hand location between trials. They proposed the hypothesis regarding the effect of historical movements on the estimation of the current movement. But we still consider these two studies comparable, because in fact, they are qualitatively studying the same thing, how humans estimate their movements. From this viewpoint, it is reasonable to borrow ideas from the second study to modify the simulation implemented in the first study.

Second, in both studies, an update process relying on an error—the difference between predicted (intended) results and sensory (actual) results—is used in the process of producing the estimate of movements. In the first study, the predicted result is corrected using sensory results.

A parameter *K* is used to weight the effect of the error in this correction (see Eq. (15)). The value of *K* depends on both the inaccuracy of the knowledge of the environmental dynamics, and the noise in the sensory process (see Eq. (14)). In the second study, a memory of errors controls (weights) how much the current error will be used for updating the newly estimated result, the magnitude of a type of learning rate. Here we see that in the first study, there are two factors—the inaccuracy and the noise—that weight the error, and in the second study, a third factor—memory of errors—is used.

A Model That Estimates Human Movements Using Memory of Errors

In this section, we first propose an operational definition for a learning rate η that determines how memory of errors functionally controls the extent to which errors will be learned. This definition is conceptually inspired by the work of Herzfeld et al. (2014). Then we introduce a modified Kalman filter, in which we add a new factor—memory of errors—to balance the importance between predicted results and sensory results. The effect of this new factor is represented by the magnitude of the learning rate η. In this way, we achieve a new model that is able to reflect its knowledge of memory of errors—a feature of the environmental dynamics—into the process of producing movement estimates. Finally, we add this estimation model, which is implemented by the modified Kalman filter, to the Sensory Motor System (SMS).

The Learning Rate η

The magnitude of the learning rate (η) is controlled by memory of errors. The specific formula for this control is represented as a sigmoid function expressed by Eq. (17), and which is assisted by Eq. (18). The learning rate ranges from 0.5 to 1.5 with a default value of 1.0.

$$\eta = \frac{1}{1+ e^{-\theta t}} + 0.5 \qquad (17)$$

$$t = 1.0 - \frac{2s}{n} \text{ if } n \neq 0, \text{ and } t = 0 \text{ if } n = 0 \quad (18)$$

Specifically in Eq. (17), the variable *t* represents the status of the memory of errors, which is calculated according to Eq. (18). It ranges from -1.0 to 1.0. The parameter θ tunes the effect of *t*, and is set to 6.0 by default. In Eq. (18), the variable *n* represents how many errors have been experienced by the brain, and thus stored in the memory. Variable *n* is an integer starting from zero.

As mentioned in above, the forces of perturbations used in the study of Herzfeld et al. (2014) have the same magnitude with directions either to the left or to the right. Similarly, we set only two types of errors in our model: the same magnitude with either positive or negative sign. Variable *s* represents how many times the error type has switched within the memory of errors. Variable *s* is an integer starting from zero.

Here we explain the behavior of the above formula with examples. If the brain has experienced many errors and most of them have the same sign, the value of *n* is large and the value of *s* is small; therefore, the value of *t* is large, close to 1, so that the value of θt is close to 6.0 and η is close to 1.5. This means a slowly switching environment results in a high learning rate—learning more from the current error. On the other hand, if there are many errors in memory and they have switched signs very often, the values of both *s* and *n* are large, so *t* is negative with a large absolute value; thus the value of η is close to 0.5. This means a rapidly switching environment leads to a low learning rate—learning less from the current error. These simulated behaviors qualitatively agree with the hypothesis proposed by Herzfeld et al. (2014). Note that when there is no error in the memory yet, the value of *t* is 0 because *n* is 0, so the value of η is 1.0, which is considered the default value of η.

A Modified Kalman Filter

Compared to the original Kalman filter expressed by Eqs. (12) ~ (16), we modified Eq. (15) by adding a new variable η, which is defined in above sections, as expressed by Eq. (19). The newly modified Kalman filter is expressed by Eqs. (12)~(14), (16), and (19).

$$x_t = \bar{x}_t + \eta K_t(Cz_t - \bar{x}_t) \tag{19}$$

The added variable η represents a new factor that balances the importance between predicted results and sensory results, occurring together with the parameter K.

Two questions need answering regarding this modified Kalman filter: does this modification make sense, and what is its benefit? For the first question, as we have discussed above, both of the studies (Herzfeld et al., 2014; Wolpert et al., 1995) introduce a process that updates the movement estimate using a given error, the difference between predicted and sensory results. Although the two updating processes are in different granularity: to update the estimate within one movement trial or between trials, they conceptually produce the same thing. A parameter has been used to weight the error in each of the updating processes: a Kalman gain K in the Kalman filter, and a learning rate η described above. Because η has a nature that K does not have—the representation of the effect of memory of errors—it is reasonable to add η into the Kalman filter to weight the error together with K.

Second, the major benefit of adding the parameter η is to handle more cases—allowing us to simulate more human behaviors using memory of errors; the original Kalman filter uses only the previous estimate to make the current estimate. The modified Kalman filter has both inherited the capabilities of the original Kalman filter (Wolpert et al., 1995) that simulates the estimation process within a single trial of movement, and obtained a new way to weight the error for updating the estimate of movements (Herzfeld et al., 2014), so as to simulate the estimation

between movement trials. In the following section, we examine the capabilities of the modified Kalman filter by implementing it into a simulated lifting movement.

Adding an Estimation Process into the SMS

As shown in Figure 17, the original SMS generates low-level motor commands to actuators within the environment. It is driven by 1) a high-level goal-directed action provided by LIDA's Action Selection module, and 2) the sensory data perceived from LIDA's Sensory Memory. We added an estimation process into the SMS of LIDA to implement our modified Kalman filter. The inputs to this estimation process comprise a copy of motor commands (the efference copy) together with real sensory data. The newly added estimation process in the SMS provides estimated sensory data to the SMS's original "motor command generation" component. Inside this new process, we implement two sub-modules, an internal model and a correction process, which accomplish the prediction and correction steps of the modified Kalman filter respectively. The above Eqs. (12)~(14), (16), and (19) explain detailed computational expressions of the prediction and correction steps.

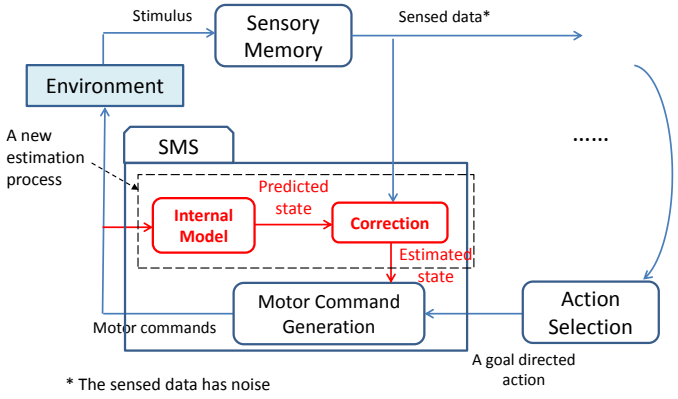


Figure 17. An estimation process in the Sensory Motor System (SMS) of LIDA

We have observed that the motor commands sent out to the actuators need time to be executed, which means that at a given time, the motor commands and the sensory data input into the estimation process may not be consistent. To deal with this, we have created a FIFO (First In First Out) queue for storage of the input motor commands in the internal model and set the queue's length to one. Thus, there is a one-step delay between the motor commands and the current sensory data used for the estimation.

Experiments

In this section, we test the performance of the estimation process of our newly proposed model in a simulated hand lifting action, by comparing its estimation process with human behaviors reported from two previous studies (Herzfeld et al., 2014; Wolpert et al., 1995). The comparison results support our new model's ability to simulate the estimation process not only within one trial of the movement but also between trials using memory of errors.

Experimental Setup

From recent reviews of the study of human hand-lifting movement (Johansson & Flanagan, 2009; Wolpert et al., 2011), we see that some researchers (Berner, Schönfeldt-Lecuona, & Nowak, 2007; Flanagan, Bittner, & Johansson, 2008; Jenmalm, Schmitz, Forssberg, & Ehrsson, 2006) have supported the existence of a (forward) internal model occurring during lifting. They hypothesize that people predict their lifting movements based on a system that simulates the behavior of their body and their environment (Wolpert et al., 2011), and "the CNS signals the sensory discrepancy between the predicted and actual sensory consequences of action" (Jenmalm et al., 2006). These hypotheses have led us to choose lifting as a reasonable target movement to which to apply our model to simulate the human movement estimation

process, because the hypotheses support the primary mechanism of our model, a modified Kalman filter.

We use a software robot simulation (youBot), a robot controller (the LIDA Framework (Snaider et al., 2011)), and a virtual experimental environment (Webots (www.cyberbotics.com)) to simulate a lifting movement. We consider this robotic simulation to be a LIDA-based software agent. The LIDA Framework, youBot, and Webots have been introduced in detail in sections 4.4.1 and 4.4.2 above. Here we present only a screenshot of the LIDA-based agent lifting an object (Figure 18), so as to give an intuitive feel for the agent and its action. Specifically, in our experiment lifting refers to an action in which the agent grips an object, and moves it upwards. The gripper tip locations serve as the hand locations.

Figure 18. A screen shot of a LIDA-based agent lifting an object

We implement our new model into the Sensory Motor System (SMS) of LIDA. In the LIDA Framework's Environment module (see Figure 1), we have added noise to joints 1 ~ 3 (see Figure 8) by randomly setting their angles with a normal distribution: the mean is the actual measure of the angle, and the STD is 0.1 degrees. In this way, uncertain sensory data is sent to the estimation process. We use the added process to estimate the youBot's finger positions during executing an action in the above uncertain situation.

Implementation of the Learning Rate η

As defined above, the value of η depends on both the number of historical errors and the switching time between these errors. Computationally, we created three variables stored in long-term memory: (1) the number of errors n, (2) the number of switches s, and (3) the current error type c. The first two variables n and s have been introduced in above sections. Variable c is used to determine whether the current error and the upcoming error have different types. If the two errors have different types, one instance of error-switching will be accumulated to variable s; otherwise the value of s does not change.

In the experiment, these three variables are retrieved once when the agent initializes a lifting movement; thus, the value of η is calculated before the start of the movement and is constant within one trial. Then, at the end of every lifting trial, the three variables are updated based on the error between the estimated hand location and the actual hand location. In this way, the value of η may change between trials.

Estimation without Memory of Errors

We prepare a computational experiment that is configured similarly to the human experiment reported earlier (Wolpert et al., 1995), which studies the estimation process within a single movement trial without being concerned about memory of errors.

As we have reviewed above, in the study of Wolpert and colleagues (1995), human participants are asked to move their hand in dark, and they stop moving and report an estimated hand location when they hear a tone. In our simulation, the agent does not have visual sensors but senses the angles of its arm's joints; this configuration conforms to the situation in the human experiment, namely, that participants are without vision, and guided solely by proprioception. Also, we created a program that sends a stop command to the agent, instructing it to stop lifting.

This program plays the role of "the experimenters" who control the timing of the tone in the human experiment. In the human case, a pair of real and estimated hand locations was collected at the end of each movement trial. So in total, 2400 data pairs were collected (eight participants with 300 trials each). In our simulation, the "experimenter" program generates one stop command at a different virtual time[3] during each lifting trial. Stop commands are generated so as to give a range of lift durations from 6 to 65 units over 60 lifting trials. We consider the process during the first 5 time units to be the system's initiation process, and did not collect data during this interval. We performed 40 repetitions of the above trial block (60 lifting trials with different durations) for a total of 2400 data pairs of estimated hand location and actual location, in order to achieve parity with the data collected during the human experiment.

On the agent's side, first it senses the stop command from its environment as an input to its Sensory Memory; and then this command is sent to the Current Situational Model (CSM) as part of the agent's current understanding of the environment. The command is represented as a stop node in the CSM. A special Attention Codelet is implemented to attend to this stop node, and form it into a special data structure, a coalition (Baars, 2002; Franklin et al., 2014), sending the coalition then to the Global Workspace (GW). In the GW, the coalition containing the stop node might win a competition among different coalitions, and thus be broadcast to the rest of the system as the conscious content. There are multiple schemes stored in Procedural Memory (PM), which are able to be instantiated to behaviors. We prepared a special scheme that (1) will be recruited by the arrival of the stop node in the conscious content, and (2) contains an action component for executing a stop command. Then when the stop node comes through the conscious content to PM, this scheme is chosen and instantiated into a behavior that has an

[3] The agent executes at unit intervals in Webots virtual time.

action component for stop. Finally when this behavior arrives at the Sensory Motor System (SMS), the currently running lifting movement is stopped.

In our simulation, the differences (errors) between real and estimated hand locations are recorded as a function of the duration of the hand lifting movement. The average error for each moment (virtual time unit) is calculated, and is shown in Figure 19; movement duration is represented as a number of virtual time units. These simulated results are qualitatively similar to the human data (Wolpert et al., 1995): Overall, 1) the hand location is overestimated, and 2) the error peaks in the first part of the movement (at virtual time 23), and then goes down.

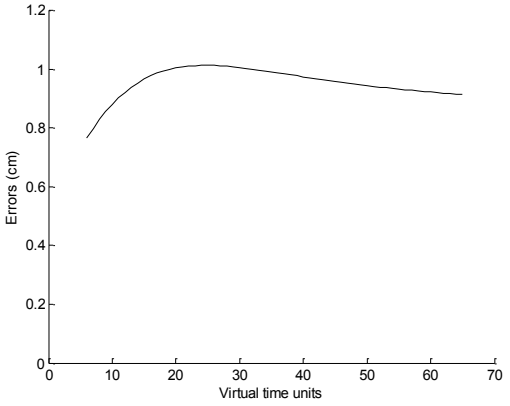


*Figure 19***.** Simulated estimation errors of hand lifting action on average without memory of errors

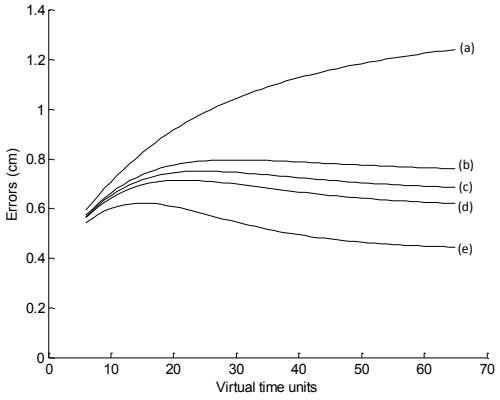


Figure 20. Different propagation of simulated estimation errors of hand lifting action on average. The propagation of errors (a) through (e) are when experiencing different environments that have the error switching rates of 90%, 70%, 50%, 30%, and 10% respectively

Estimation with Different Memory of Errors

In this sub-section, we describe a computational experiment to examine the effect of memory of errors on the estimation process of an agent. This effect has been examined in, and supported by, human experiments (Herzfeld et al., 2014).

In our experiment, the agent may lift three types of objects, which have different weights: 0.1kg, 0.2kg, or 0.3kg. We consider 0.2kg to be the default weight, and 0.1kg to be lighter and 0.3kg to be heavier. To artificially create errors as those that were introduced in the human experiments (Herzfeld et al., 2014), we first configure the agent's knowledge of the object's weight to a default value (0.2kg), and then let the agent lift either a lighter (0.1kg) or a heavier object (0.3kg). In this way, the difference (error) occurs between the estimated hand location and the actual one, and two types of errors, positive or negative, are made by using lighter or heavier objects respectively.

Based on the fact that the sequence of errors stored in memory may switch between positive and negative, we prepare five types of environment that the agent can experience: error switching rates of 10%, 30%, 50%, 70%, or 90% respectively.

To observe the effect of memory of errors, we first let the agent perform 30 lifting trials, using either lighter or heavier objects, to create its memory of errors, and then we let it do one lifting trial using a heavier object. We analyze the propagation of simulated estimation errors during the last lifting trial when the agent has experienced a certain type of environment. In detail, we let the agent perform the above 31 trials 25 times for each type of environment, and calculated the estimation errors on average during the 31st trial, as shown in Figure 20. Within every 31 trials, the value of η changes (see Sections 5.3.1 and 5.4.2), and its value is initialized to zero when the agent starts a new sequence of 31 trials. The approach we are using here to

explore the effect of memory of errors is based on the design of previous human experiments (Herzfeld et al., 2014).

As shown in Figure 20, the simulated estimation errors are different in different environments. In detail, the errors are smaller when the environment the agent has experienced has a lower switching rate of errors—propagation (a) is largest and (e) is smallest; that is, the error propagation peak is lower, and the decline after the peak is more rapid. This difference demonstrates that when the environment is more stable—having lower error switching rate—the estimated hand location is closer to the actual because the agent learns more from a given error created using a heavier object. This effect of the environment (memory of errors) matches the phenomenon found in the human experiment (Herzfeld et al., 2014). In more detail, the value of η is different while generating propagations of simulated estimation errors ((a) ~ (e)). For example, in situation (a), the agent experiences a rapidly changing environment (switching rate of 90%), so in Eq. (18) variable s is close to n, and then together with Eq. (17) the value of η nearly reaches its minimum, 0.5. On the other hand, in situation (e), because the agent experiences a very steady environment (switching rate of 10%), we can infer that the value of η nearly reaches its maximum, 1.5. Similar computational inferences can be done for situations (b) ~ (d) as well. These inferences match the interpretation of Herzfeld and his colleagues for the human results (2014). Therefore, we argue that we have simulated both the phenomenon and causal factors present in certain human experiments (Herzfeld et al., 2014).

Furthermore, for most propagations of simulated estimation errors shown in Figure 20, from (b) through (e), their behaviors are similar to study results of human behavior (Wolpert et al., 1995): 1) the hand location is overestimated, and 2) the error peaks in the first part of the movement, and then goes down. The only exception is the propagation (a) in Figure 20, which

does not exactly follow the human experimental result (Wolpert et al., 1995): although it shows the overestimation of the hand location, its error simply goes up but does not have an ensuing decline. We think this exception may be due to the fact that the 90% switching rate is an extreme situation that is outside the scope of the hypothesis (Wolpert et al., 1995) describing usual human behavior. In this situation, the agent has experienced a very rapidly changing environment, so it almost does not believe the current sensed data—the agent's knowledge dominates the estimation. That is why the decline does not appear after the peak, and the decline is the result of a trade-off between the agent's knowledge of the dynamics of the environment and its sensory data.

In summary, together with the experimental results shown above, we have shown that an agent embedded with our newly proposed model is able to simulate both (1) human estimation of its lifting movement within one trial (Wolpert et al., 1995), and (2) human estimation between lifting trials driven by memory of errors (Herzfeld et al., 2014).

5.5 Conclusion

We have presented a new model that estimates human movements using memory of errors. Furthermore, we have computationally embedded this model into a cognitive model, LIDA (Franklin et al., 2014), to simulate human self-estimation of their movements.

6. Modeling Sensorimotor Learning in LIDA

Introduction

We studied the term "sensorimotor" from the concept of cognitive development proposed by Jean Piaget (Piaget, Brown, & Thampy, 1985; Pulaski, 1980). As he reported, when an infant was in his first two years of life, within the so-called "sensorimotor stage," the infant builds a mental mechanism for its overall interaction with the environment. This building process results from several inherited elements, as well as his experience interacting with the environment over time. In our view, we consider this mental mechanism the infant's initial mind, and we think there is a kind of sensory motor system acting in the mind for the control of such behavioral interactions. As the infant progresses to the higher cognitive developmental stages, more complex mental processes and representations emerge, and the sensory motor system integrates with them while continuing to handle the interaction with the environment. In a mature human's mind, the sensory motor system cooperates with other parts of mind, and directly interacts with the environment.

In a recent review in neuroscience (Wolpert et al., 2011), the authors argued that there are different task components necessary for motor learning, including relevant information gathering, selection of strategies, and both predictive and reactive (motor) control mechanisms. Furthermore, different learning processes are necessary to be applied on these components. These necessities have been conceptually fulfilled in a cognitive architecture, LIDA.

We present a new model of sensorimotor learning in LIDA using the concept of reinforcement learning. This is the second implementation of learning in LIDA, the first being the modeling of attentional learning by Faghihi and colleagues (2012). The new model stores and updates the rewards of pairs of data, motor commands and their contexts, using the concept of reinforcement learning; thus the agent is able to generate (output) effective commands in certain

contexts based on its reward history. Following Global Workspace Theory, the primary basis of LIDA, the process of updating rewards in sensorimotor learning is cued by the agent's conscious content, the most salient portion of the agent's understanding of the current situation, issued by the Global Workspace module of LIDA.

Furthermore, researchers in neuroscience have recently proposed that the brain maintains a memory of errors in sensorimotor learning, and they found that during a motor task, "the brain controls how much it is willing to learn from the current error through a principled mechanism that depends on the history of past errors" (Herzfeld et al., 2014). Here the error is the difference between the brain's predicted result and the sensory result. These researchers proposed a concept, called error sensitivity (Herzfeld et al., 2014) or learning rate (Gonzalez Castro, Hadjiosif, Hemphill, & Smith, 2014), to represent the percentage of error that will be added to the predicted results during its updating.

The researchers found that the brain controls this error sensitivity depending on the error's history (Herzfeld et al., 2014). The brain learns more from the errors—error sensitivity becomes high—when their histories are likely to persist, and it learns less—error sensitivity becomes low—when the histories were likely to change. "Persistent errors" refers to those historical errors that have the same sign, either both positive or both negative. Another relevant work has been reported for the dynamic regulation of reinforcement learning parameters as well (Khamassi, Enel, Dominey, & Procyk, 2013). In their work, the learning rate is dynamically tuned as a function of the environment's volatility. In our view, the environment's volatility (uncertainty) is similar to the error sensitivity introduced above, a kind of environment's stability. Inspired by these hypotheses, we introduce the effect of memory of errors into the newly added learning mechanism, so as to implement a dynamic learning rate.

In the next section, we introduce the work of modeling sensorimotor learning in LIDA. Then in the following section, we describe an addition of a dynamic learning rate into this learning mechanism. Following that, we provide current experimental results. Finally, we conclude the work and propose some directions for further research.

Modeling Sensorimotor Learning in LIDA

Practically it is easy to study this modeling work by using an example that includes concrete motor commands. Therefore, we simulated an autonomous agent (Franklin & Graesser, 1997) to implement our model of sensorimotor learning. The agent consists of a simulated robot body and a controller implemented using the computational LIDA framework (Snaider et al., 2011). This agent is designed to learn how to push a box properly.

Below we introduce the robot, the cooperation between the Sensory Motor System (SMS) and some of LIDA's other modules, the development of the new SMS, and the implementation of other relevant LIDA modules.

A Box Pushing Robot

We reuse a two-wheeled robot provided by Webots, as shown in Figure 21. Its simulated height is 0.08m and its radius is 0.045m. The motor commands of the robot are limited to going forward, and turning left or right either by approximately 13 or 22 degrees.

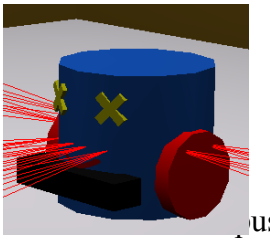


Figure 21: A ushing robot

The robot has eight distance sensors. Each of these sensors has a field of view of about 15 degrees, and can detect objects within 0.3m. We simplified the sensors to detect objects in two distance ranges, one from 0m to 0.15m (NEAR) and the other from 0.15m to 0.25m (FAR). These sensors are arranged in an orthogonal pattern, with four on the front and two on each side. In addition, a touch sensor is placed on the robot's front to detect a bump, and another sensor built inside the robot's body detects when the robot becomes stuck—the sensor senses the agent's location and rotation; it is activated when the agent's location and rotation are the same during two consecutive sensory cycles.

The Cooperation between the SMS and Some Other LIDA Modules

There are two LIDA modules, Action Selection and Sensory Memory (see Figure 1), that provide relevant information as separate inputs to the SMS. The SMS sends out motor commands as its output to the environment. The output of the SMS also modulates other parts of LIDA. The LIDA-based agent is an autonomous agent that senses the effects of its own previous output (motor commands), which influence other modules in LIDA.

We implemented a broadcasting channel from LIDA's Global Workspace (GW) to the SMS (Sensory Motor Memory in Figure 1), sending the agent's conscious content to the SMS. The arrival of this content cues (initializes) the update of the rewards to motor commands in the SMS so as to assign credit to effective commands. Note that the conscious content does not directly provide the rewards, but it leads to the process of making and then updating the rewards. This is in keeping with GWT, in which the conscious content broadcast from the GW modulates learning in the rest of the system.

The Development of the SMS

The SMS is the key module that was augmented when we implemented a model of sensorimotor learning into LIDA. Prior this addition, motor commands were built into a mechanism (subsumption architecture style) implemented in the SMS, and could not be changed at runtime. Now, the sensorimotor learning implemented into LIDA leads to a dynamic selection of motor commands at runtime based on the newly experienced rewards to the commands. Our computational design is inspired by Mahadevan and Connell's previous work (1992). They added a learning aspect into the traditional subsumption architecture using the idea of reinforcement learning (Kaelbling, Littman, & Moore, 1996). We improved theirs in two primary ways: 1) we imbued the original learning with a more biologically inspired interpretation by bringing it into LIDA to implement sensorimotor learning—basically in LIDA, the arrival of new conscious content issued from the Global Workspace module cues the creating and updating of rewards in the SMS, and 2) we implemented a new mechanism to control the rate of learning (see this in the below section later).

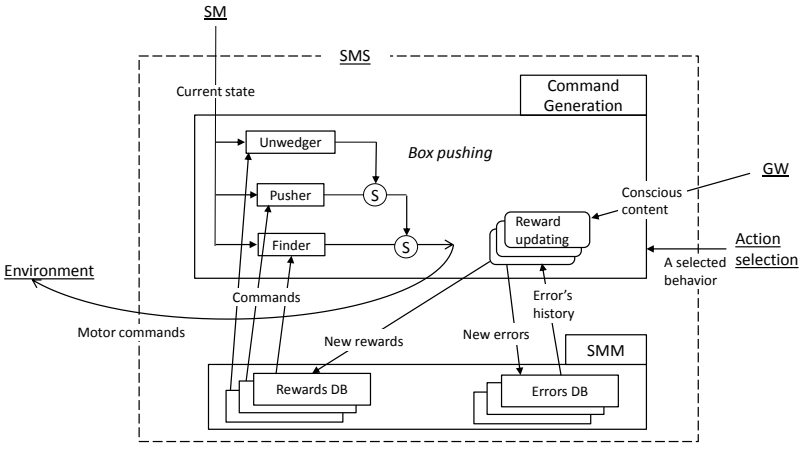


Figure 22: The design of a new SMS

The design of the new SMS is shown in Figure 22. Compared to its previous version, we have added three new components: (1) a set of reward updating processes, (2) a set of rewards databases, and (3) a set of errors databases. We introduce the first two components and the SMS for modeling sensorimotor learning below, and leave the introduction of the errors databases for adding a dynamic learning rate to the following section. In brief, the rewards database maintains the reward values, while the errors database stores the history of reward prediction errors. Note that in our work, the term error does not mean a "punishment", the opposite of a reward; rather an error here refers to the difference between the currently stored rewards and newly generated rewards.

In Figure 22, the SMS contains a (motor) command generation module depicted in the upper part of the diagram and a long-term memory, Sensory Motor Memory (SMM), depicted in the bottom part. The command generation module responds to the execution of a selected behavior. That behavior results from the preceding Action Selection Module on the right, and acts to specify a desired action in LIDA. General details about the behavior data structure can be found in (Franklin et al., 2014). In our case, the selected behavior is pushing a box. On the left side of this module, a reactive motor control mechanism—a kind of subsumption architecture—is built in. The structure of the mechanism implements a priority network that imposes an order on the three sub-tasks of box pushing. The unwedger's behavior suppresses the pusher's, and both of these suppress the finder's. A suppression operation is represented by an encircled uppercase S in the network diagram. Briefly, the agent begins by finding and closing a box, it then continuously pushes the box, and finally the agent can "unwedge" itself if it becomes stuck. These subtasks are implemented by finite state machines (FSMs), which are driven by the

current state of the environment sensed through the Sensory Memory (SM), and which may in certain states send out motor commands to the environment.

Because of the implemented sensorimotor learning, the motor commands sent out from the FSM can now be dynamically chosen at runtime based on their rewards. Each FSM has its own rewards database maintained in SMM. Another part of learning is a set of reward updating processes, which are depicted on the right side of the command generation module. These processes are driven by the conscious content broadcast from LIDA's Global Workspace (GW), and each of them one-to-one updates the reward values stored in a rewards database for a certain FSM.

The algorithm of the reward updating process is inspired by Q-Learning (Watkins, 1989); this updating helps the agent propagate rewards in the time line. The reward update formula (see Eq. (20)) uses a reward function Q(x, m) across states (x) and motor commands (m). This reward function is defined by $Q(x, m) = r + \Upsilon E(y)$, where *r* is the immediate reward, and E(y) is the expected reward of the state y resulting from the command. E(y) is the maximum Q(y, m) over all commands m. Υ is a discount parameter that is set to 0.9, which determines the importance of future rewards. Its current value 0.9 is supported by Mahadevan and Connell's experimental results (1992).

$$Q(x, m) \leftarrow Q(x, m) + \beta(r + \Upsilon E(y) - Q(x, m)) \qquad (20)$$

During updating, since the current stored reward Q(y, m) has not yet converged to the updated value—$r + \Upsilon E(y)$—the difference between them then provides the reward error in the current stored rewards. This error is used to update the stored rewards using a learning rate β. Currently, the value of β is set to 0.5 as supported by Mahadevan and Connell's experimental results (1992); but we will replace it using a dynamic learning rate mechanism described below.

In Eq. (20), immediate rewards (r) are calculated differently depending on the FSMs' behavior (see Figure 22). First, for finding a box, the agent is rewarded by +3 if it detects an object in its front NEAR zone during forward movement, or it is punished by -1 if no object is there. The default reward is 0 if the agent is not moving forward. Second, for pushing a box, the agent is rewarded by +1 if it is touching an object during its forward motion, or it is punished by -3 if not touching. The default reward is 0 as before. Finally, for getting unwedged, the agent is punished by -3 if it is wedged while moving forward, or it is rewarded by +1 if no wedging occurs. The default reward is 0 if the agent is neither wedged nor moving forward.

When a FSM chooses its current command, in 90% of the time, given the same current state, the motor command that has maximum reward values is chosen. In the remaining 10% of cases, a motor command is randomly chosen. Choosing commands only based on their rewards will never allow the exploration of new commands or new states. Sometimes, a random command is chosen to ensure that all states in the state space will eventually be explored. Suggested by Mahadevan and Connell (1992), 10% is a good compromise between exploratory and goal-directed activity, in line with their experimental results.

Implementation of Other Relevant LIDA Modules

We implemented several other relevant LIDA modules appropriate for the specification of learning a box pushing task using sensorimotor learning. We list the implementation of each module below, ordered according to the three phases of the LIDA cognitive cycle: understanding, attention, and action/learning. Figure 1 gives an intuitive feel for the relationship of these modules. Details of these modules can be found in (Franklin et al., 2014).

Sensory Memory (SM)

SM gets sensory data from environment, structured as an array of Boolean bits, representing the data status sensed from each of the sensors, either active or inactive. SM provides the SMS with the current data.

Feature detectors (FDs) and Perceptual Associative Memory (PAM)

PAM stores a set of nodes, each of them representing a specific aspect of an environmental state of concern to the agent. In our work, these nodes are distance nodes including NEAR and FAR, a bumping node, and a stuck node. FDs constantly obtain the current state from the SM, activating relevant nodes in PAM.

The Current Situational Model (CSM) and structure building codelets (SBCs)

The CSM receives currently activated nodes from PAM, and builds the agent's understanding of the current situation. SBCs reorganize data in the CSM, combining sets of nodes and links into node/link structures. They build an agent's higher level understanding of the current situation.

Attention codelets and the Global Workspace (GW)

We added an attention codelet concerned for the entire current situation in the CSM, and bringing it into the GW. In the GW, the current situation may win the competition to produce the agent's conscious content. A channel from the GW to the SMS—newly implemented this time—broadcasts the conscious content to other modules including the SMS.

Procedural Memory (PM)

Following the broadcast of conscious content from the GW, a box pushing scheme is recruited in the PM, and then a relevant behavior is instantiated that is selected by the Action Selection module and sent to SMS, which initiates a motor command generation mechanism for executing the box pushing in the SMS.

A Dynamic Learning Rate in Sensorimotor Learning

In a study of sensorimotor learning, Herzfeld and colleagues (2014) have hypothesized that the brain not only learns from individual errors that it has experienced before, but also accumulates the errors into a memory; this memory of errors makes it possible for the brain to control how much it will learn from a given current error. These historical errors help humans determine whether the environment is steady or rapidly changing. Environmental stability thus controls how much of a given error will be learned so as to affect the prediction of the upcoming movement. This study has been described in detail in the previous sections.

We interpret the above hypothesis as a computational mechanism. In the mechanism, memory of errors controls the value of a "learning rate", which weights the amount of an error—typically the difference between sensory (actual) result and predicted (intended) result—that will be used in an update process of the predicted result.

In the work of modeling sensorimotor learning as described above, one step in updating the reward of motor commands is to learn from the reward error as expressed by Eq. (20). We consider this reward updating process to be similar to the update process mentioned in the above mechanism. Therefore, here we revise Eq. (20) by changing the quality of its parameter β from a constant value to a dynamic value. Now the value of parameter β will be controlled by memory of errors as introduced above (Herzfeld et al., 2014).

Note that the reward of motor commands manipulated in our development of sensorimotor learning is different than the execution result (the movement) of motor commands discussed by those neuroscience researchers in sensorimotor learning (Herzfeld et al., 2014). In our sensorimotor learning, reward of motor commands is maintained inside an agent as the judgment provided by the agent's mind. In contrast, those neuroscience researchers assume that

the movement of motor commands occurs outside of the agent as the feedback "provided" by the environment. We consider the reward and the movement to be the two indicators for the evaluation of a motor command. We note that both of them are used to indicate aspects of motor commands, and it seems they also have some similar principles, such as the way to update their indications of motor commands—they both use a type of learning rate to weight the updating of the old knowledge of the motor command by the new. Thus, we have integrated the idea of memory of errors from Herzfeld and colleagues (2014) into our work for updating the reward of motor commands. We consider this a kind of indirect biological inspiration for our approach.

Next we provide the way in which memory of errors dynamically controls the value of the parameter β. (This parameter β is a similar concept to that of the learning rate η previously introduced, both of them are inspired by the concept of memory of errors (Herzfeld et al., 2014); but their computational implementations are slightly different.)

First, a (reward) error is classified as either positive or negative, depending on the sign of subtracting the old (stored) reward from the new reward. Then when our agent has performed its task (pushing a box) for a while, having experienced a sequence of errors, the type of these errors may switch differently, switching from slowly to rapidly; here we mean switching between positive and negative reward errors, and the switches occurs either frequently (i.e. "rapidly") or sporadically (i.e. "slowly"). Therefore, we have different types of memory of errors from the viewpoint of their stability, based on the rate of switching. We represent the number of errors the agent has experienced by a variable n, and the number of switches between these errors by a variable s. These variables are integers starting from zero. Based on these two variables, we represent the status of the memory of errors by a variable t as expressed by Eq. (21), and then calculate the value of β using a sigmoid function assisted by the variable t as expressed by Eq.

(22). The parameter θ tunes the effect of t, and is set to 1.0 by default. β ranges from 0.0 to 1.0 with a default value of 0.5.

$$t = n - 2 * s \quad (21)$$

$$\beta = \frac{1}{1 + e^{-t*\theta}} \quad (22)$$

Here we illustrate the behavior of β with examples. If the agent has experienced many errors that rarely switch, the value of n is large and the value of s is small; therefore, the value of t and therefore θt are large; so β is close to 1.0. This means that a slowly switching environment results in a high learning rate. On the other hand, if there are errors in memory but they have switched signs very often, the values of both s and n are large, so t is negative with a large absolute value; thus the value of β is close to 0.0. This means a rapidly switching environment leads to a low learning rate. These simulated behaviors qualitatively agree with the hypothesis proposed by Herzfeld et al.(2014). Note that when there is no error in the memory yet, the value of t is 0 because n and s are 0, so the value of β is 0.5, which is the same as the default value of β.

Finally, we add error databases to store the history of variables n and variable s. These databases are maintained in the SMM as shown in Figure 22. They interact with reward processes to (1) update memory of errors based on the arrival of new errors, and (2) provide current error's history to dynamically control the value of β.

Experimental Results

This section describes an experimental study to evaluate this modeled sensorimotor learning and its dynamic learning rate. Figure 23 shows a bird's eye view of the experimental environment and the agent in their initial configuration. The agent stands in a field containing three movable boxes. They are surrounded by walls, a kind of obstacle that cannot be moved.

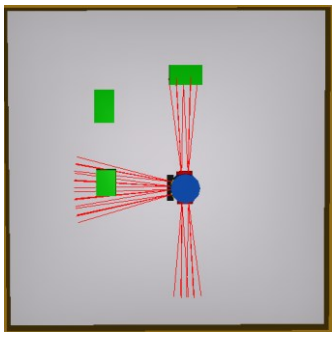


Figure 23: A bird's eye view of the experimental environment and the agent

We are interested in the following questions: 1) How well will the action of pushing a box be executed with sensorimotor learning; and 2) What is the effect of implementing the dynamic learning rate into the learning?

We evaluate the performance of the box pushing using two criteria: 1) the average value of the reward obtained so far by the pusher FSM (see Figure 22) inside the agent—we consider pushing to be the core part of the box pushing task—and 2) the distance that the boxes have been moved in the environment.

We compare the box pushing performance across six different agent conditions: 1) randomly chosen motor commands; 2) handcoded motor commands; 3-5) sensorimotor learning with constant learning rates of 0.1, 0.5, and 0.9 respectively; and 6) learning with a dynamic learning rate. Under the handcoded condition, the expected "best" commands are chosen when the agent is in certain given states: the finder module chooses forward motion if an object has been detected to be near the front; the pusher chooses forward if a bump event is detected; and the unwedger module randomly chooses a command to turn left or right if the agent is stuck. In other states, commands are randomly chosen.

In each condition, we perform 10 consecutive trials. A new trial begins from the initial configuration of the environment and the agent, but the rewards of the motor commands and the

reward errors are remembered throughout the trials. During each trial the agent runs 500 simulation steps, so under each condition, the agent runs 5000 steps. In our case, each step is simulated with 50 virtual time units—the agent executes at unit intervals in Webots' virtual time.

We collected the first criterion of average value of the reward every 50 steps of the agent's run, so 100 average values were collected during the total of 5000 steps. Figure 24 plots the average values under the six conditions. The plotted curves illustrate that 1) with sensorimotor learning added, the pusher module obtains more rewards than the random agent does, 2) the pusher obtains the most rewards under the dynamic learning rate condition (except during the initial steps), and 3) the handcoded agent outperforms the agents with the three constant learning rates.

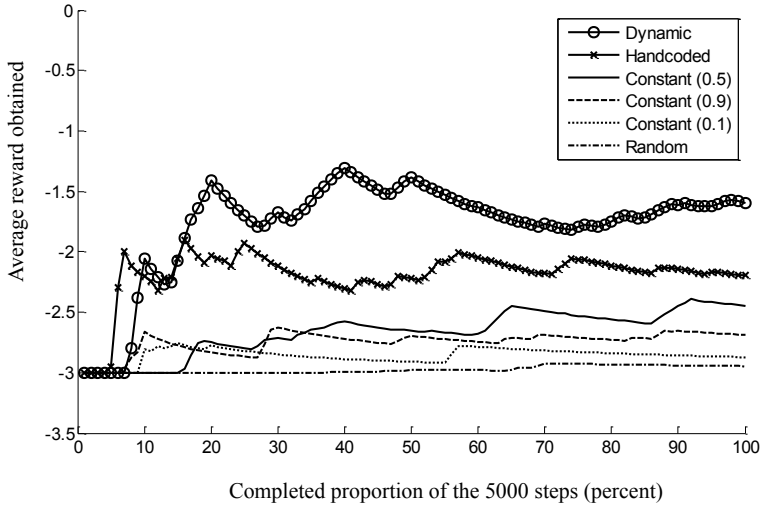


Figure 24: The average values of rewards obtained by the pusher over 5000 steps. The vertical axis represents the sum of all rewards obtained by the pusher divided by the number of steps the agent has run so far, and the horizontal axis represents the completed proportion of the 5000 steps. For learning vs. random, $p < 10^{-5}$; for dynamic vs. hand-coded, $p < 10^{-7}$; for hand-coded vs. constant learning rates, $p < 10^{-11}$.

Regarding the second criterion, the sums of the distances that the boxes have been moved during 10 trials under each of the six conditions are displayed in Table 3. These show that 1) the

agents with sensorimotor learning at the three constant rates have pushed the boxes farther than the random agent, 2) using the dynamic learning rate yields the greatest box pushing distance, and 3) the handcoded agent yields the second greatest distance.

Table 3: The sums of the distances that the boxes have been moved during 10 trials

Learning Rate	Distance (m)
Dynamic	1.6002
Handcoded	1.2023
Constant (0.5)	0.9521
Constant (0.9)	0.6357
Constant (0.1)	0.3223
Random	0.2338

The results reported above support the assertion that sensorimotor learning improves the performance of box pushing to a certain extent, and that adding the dynamic learning rate clearly increases that extent. This increased improvement supports that memory of errors—the agent's knowledge of the environment's stability—helps the agent interact with its environment more effectively.

On the other hand, we think more evidence is needed to support the causality between using a dynamic learning rate and the learning performance. Our motivation for modeling this dynamic learning rate is the replication of some recently proposed hypotheses from neuroscience regarding the effect of memory of errors (Herzfeld et al., 2014). In brief, the hypotheses suggest that a dynamic learning rate helps an agent achieve a better adaptation to its environment based on its memory of errors, but can that adaptation always be translated into improved performance? We leave this as an issue for future work.

We did not compare our results with the results obtained by Mahadevan and Connell (1992) because they have different experimental motivations, and thus a different types of results. They are interested in determining 1) the effect of decomposing the overall task into a set of subsumption modules for learning, and 2) the performances of different learning algorithms, while we are interested in the biologically inspired implementation of sensorimotor learning, and adding a learning rate control mechanism inspired by the idea of memory of errors (Herzfeld et al., 2014).

Conclusions

In this chapter, we implemented sensorimotor learning in LIDA (Franklin et al., 2014). This implementation follows the ideas of Global Workspace Theory, and uses reinforcement learning. Furthermore, we added a dynamic learning rate into the learning, which is controlled by a history of errors. Actually the approach to a variable learning rate has been explored before, such as the principle of "Win or Learn Fast" (WoFL) (Bowling & Veloso, 2001), while in our work the design distinctly relates to recent results on error memory from neuroscience. Our preliminary experimental results suggest that sensorimotor learning using a dynamic learning rate improves the performance of action execution: the generated motor commands are more effective. But as we have mentioned above, we think this conclusion needs more supporting evidence.

One major limitation in the current project is that the motor commands of the robot are very simple and the execution of each of these commands typically can be done within one step of the agent's run. That means the agent can predict the execution result of a motor command very well—its predicted result will very often be approximately the same as the sensory result. Under this condition, it is hard to model, and so then study, the motor command error, the

difference between the predicted and the sensory result. We plan to apply the currently implemented LIDA-based controller to another robot that provides more complicated motor commands, which we expect, will produce more obvious (larger) motor errors.

7. Modeling Motor Priming in LIDA

Introduction

In the field of science, we propose a hypothesis about our study target, one particular aspect of the world, and then constantly refine the hypothesis. Basically, a hypothesis is refined based on the observations of our study target and the relevant inferences from them. We design and perform experiments about the study target, so as to observe it more deeply and broadly.

In a study of human movement (T. Schmidt, 2002), the author reported that the participants' movements were affected by earlier sensory data, suggesting that priming occurs in motor control. However, our current SMS can neither explain nor replicate the priming effect Schmidt reported, so we improved the LIDA Model by extending its SMS to model motor priming in LIDA.

The next section introduces the details of the priming experiment with humans. Following that, I introduce our design of the extended SMS. And then the simulated finger movement is introduced. Finally I give the summary, the limitation, and the future work of the model of the extended SMS.

Previous work

In psychology, priming refers to an effect in which exposure to one stimulus influences the response to a later stimulus. For example, if a person sees a picture of a fish and soon thereafter reads the word "bank", then he is more likely to interpret the word as the bank a river, as opposed to a financial institution. In general, priming can affect task decisions people make, for object identification, motor control, and many others.

This experiment (T. Schmidt, 2002) was designed to measure priming effects for human movement. Participants were required to view a white dot in the center of a dark background screen, and put their right index finger on the dot. Then, they were asked to respond to the

appearance of a target with a pre-specified color (red) by moving the right index finger onto the target. In detail, an experimental trial consisted of four phases: fixation, primes, blank, and masks (See Figure 25). In the fixation phase, a white stimulus, the fixation point, was shown in the center of the screen, and the participant was required to initiate the experiment by placing his right index finger there. The fixation point remained on the screen throughout all four phases.

During the primes phase (10 ms in length), two disk stimuli (one red and one green), the primes, were shown in opposite quadrants of the display, on a rising diagonal (see Figure 25). Then the blank phase began, consisting of a specified delay from 0 to 50 ms at 10 ms intervals, during which the primes disappeared, leaving only the dark background and the fixation point. Finally in the masks phase, two annular stimuli (one red and one green), the masks, were shown at the same positions as the primes; they remained on the screen until the participant's finger reached the target. Mask colors were either switched (inconsistent) with respect to prime colors or not switched (consistent).

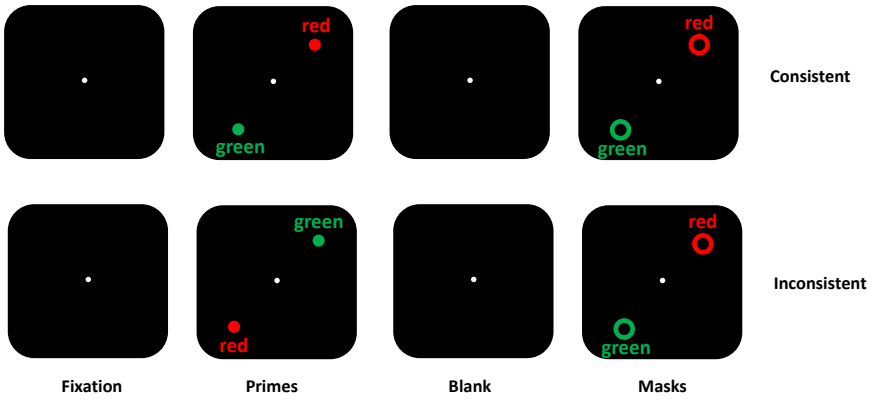


Figure 25: Four phases in the priming experiment

The participants' finger movement trajectories were recorded and analyzed (T. Schmidt, 2002). When primes and masks were consistent, the finger moved directly toward the target (red) mask stimulus. However, when primes and masks were inconsistent, the movement initially

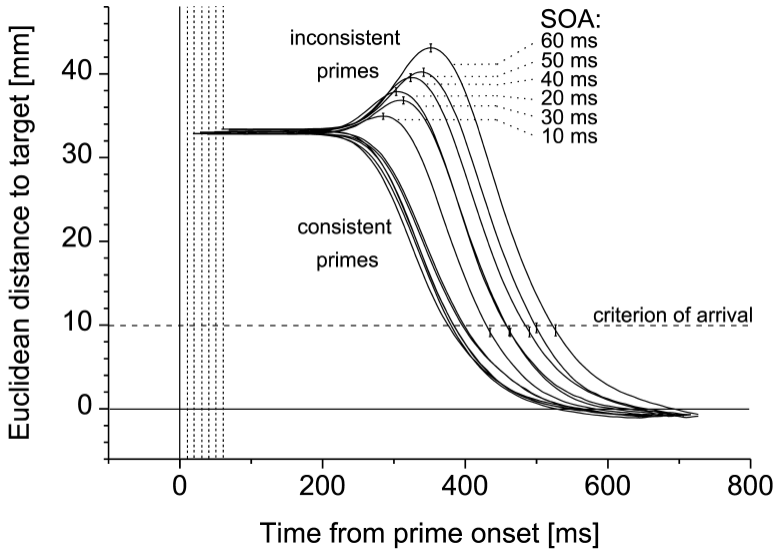
started in the direction of the non-target (green) mask stimulus but then was corrected and moved to the target (red) one.

Schmidt explained why the "wrong" finger movement was observed in the above inconsistent situation, where the participants first moved toward the non-target mask stimulus, though the goal was moving to the target one (2002). He hypothesized that the previously perceived sensory data, the target (red) prime, affects the later movement in the masks phase. Since in the inconsistent situation, the target (red) prime stimulus and the non-target (green) mask stimulus were displayed at the same locations, participants were affected to move toward the non-target (green) mask stimulus in the beginning of the masks phase. This is a typical priming effect occurring in motor control that we would like to replicate.

Furthermore, Schmidt found that in the inconsistent situation, the magnitude of the priming effect, that "wrong" direction moving, increases with the stimulus onset asynchrony (SOA) of prime and mask (2002). SOA refers to the time between the primes onset and the masks onset. Since the blank phase is specified between 0 ~ 50 ms, SOA is between 10 ~ 60 ms. The average finger movement trajectories are shown in Figure 26[1]. Regarding the inconsistent trajectories, their maximum amplitude represents the magnitude of the priming effect, which increases with the SOA. This is another important feature of the priming effect that we wish to replicate.

In Figure 26, each movement trajectory average is calculated based on about 1,000 trials. The movement is considered to have arrived when the distance to target is shorter than 10 mm. More details of the experiment can be consulted in its original report (T. Schmidt, 2002).

[1] Figure 26 is reused from (T. Schmidt, 2002) with permission.



This is the figure taken from *Schmidt, T. (2002). The finger in flight: Real-time motor control by visually masked color stimuli. Psychological Science, 13(2), 112-118.*

Figure 26: Time course of the euclidean distance between finger and target (red) mask during the finger movements. Trajectories are aligned on prime onset to show that the early phases of the movements were similar in all conditions. Vertical lines indicate onsets of primes (solid) and masks (dotted). Standard errors (between trials) at the sample times of maximum amplitude and arrival are shown. SOA = stimulus onset asynchrony of prime and mask.

The experimental results reported above (T. Schmidt, 2002) have been cited in the study of visual priming (F. Schmidt, Weber, & Schmidt, 2014; Tafazoli, Di Filippo, & Zoccolan, 2012), unconscious responses to primes (Deplancke, Madelain, Gorea, & Coello, 2013), and the channel for non-conscious vision (Breitmeyer, 2014). In addition, the further studies have pursued in different directions, including the studies of two sequential primes (Grainger, Scharnowski, Schmidt, & Herzog, 2013) and different stimuli (chromatic vs. achromatic stimuli) used in updating target location (Kane, Wade, & Ma-Wyatt, 2011).

The design of the extended SMS

In the design of the original SMS, sensory data perceived before the start action, the prime, would not have been involved in the process of action execution. Thus the priming effect

occurring in motor control had not been modeled. Here we extended the design of the SMS so as to bring the effect of earlier sensory data into action execution.

Failure to simulate the priming effect using the original Sensory Motor System (SMS)

As described above, the participants' movements were affected by their earlier sensed data. That means that priming occurs in motor control. However, the original SMS can neither explain nor replicate that priming effect, Schmidt observed. In the experiment (T. Schmidt, 2002), the earlier sensory data affects the motor control; specifically it differently affects motor control in the consistent vs. the inconsistent conditions.

Using a LIDA-based agent equipped with the original SMS, we simulated the earlier sensory data influencing a finger pointing movement as described in the human experiment (T. Schmidt, 2002). We simulated the finger movement in three situations: under (1) consistent and (2) inconsistent conditions, and (3) a "control" situation where the earlier sensory data had not been applied. We found that among the three conditions, the trajectories of the finger movements were all the same, meaning that the agent using the original SMS did not successfully simulate the priming effect.

Conceptually speaking, in the original SMS, if we consider the time that a motor plan starts to run the present, then we would have (1) the current sensory data for specification—the specification process makes a motor plan ready and starts it running—and (2) the future sensory data for an update, which occurs after the motor plan starts to run. However, the earlier sensory data, the data perceived before the motor plan starts to run, has not yet been considered, so it does not contribute to action execution in the original SMS. Therefore, applying different earlier sensory data, simulating both the inconsistent and consistent situations, or even not applying earlier sensory data, will not produce any difference.

The extended Sensory Motor System (SMS)

We need to extend LIDA's SMS to give it the capability of correctly handling the motor priming process. In the relevant situation here, the priming process begins with the instructions given by the experimenter to the participants in the experiment (T. Schmidt, 2002). These instructions would result in an expectation on the part of a participant of impending finger movement, either up and to the right or down and to the left. According to the LIDA Model with the extended SMS, when the participant's finger was placed on the centered white dot, this expectation would lead to the selection of two motor plan templates (MPTs), one for the up and to the right finger movement and the other for the down and to the left. These MPTs will be awaiting an expected target in one location each. This is the context in which we extended LIDA's SMS.

In the extended SMS, compared to the original as shown in Figure 16, we did not change the functions of the original processes, motor plan template (MPT) selection, specification, nor online control.

We added the use of earlier sensory data in action execution. In the original SMS, the motor plan is only created after the initiation of action execution triggered by the arrival of a selected behavior. In the extended version, the expectation of a target, together with earlier sensory data, can help the agent to prepare motor plans before the movement starts, and then those motor plans can be running during the movement. This use of the earlier sensory data resembles a priming process that can occur in motor control. See details about this newly added priming process.

Furthermore, in the original SMS, the time at which a motor plan begins to run is not explicitly defined. It was implicitly assumed that when a motor plan is ready, it immediately

starts to run. But is it true that there is no gap between a motor plan's ready time and its start time? If being ready is not sufficient, what other triggers or reasons are there for a motor plan to run? We discuss this next.

Finally, we discuss the cooperation of multiple motor plans, and give descriptions of how this extended SMS is used in LIDA to replicate such priming effects as are reported in the human experiment (T. Schmidt, 2002).

The priming process and a tension variable

We define the priming process extending SMS as a process in which the earlier sensory data assists in the preparation of a motor plan that affects an action execution process that starts later.

To model a priming process, we extended LIDA's SMS to include a system that allows certain sensory data to associate to a MPT that is chosen due to the presence of an expectation of an impending movement. When the MPT has been chosen, in other words, when its activation becomes relatively high but not high enough to be selected yet, then the arrival of the required sensory data may help to select this MPT, and then specify it to a ready motor plan.

One important aspect of priming is the way in which the prepared motor plan affects the upcoming action execution. To simulate this effect in a quantitative way, we added a new feature to the motor plan, represented by a floating point variable, tension. From a conceptual viewpoint, the value of the tension in a motor plan represents the motor plan's potential power, how strong the motor commands generated by this motor plan will be. For example, with respect to a motor plan generating motor commands that control the index finger moving to a target location, higher tension in the motor plan will yield motor commands to be applied to the finger with stronger force.

A motor plan's tension will have different values depending on the situations. First, when a motor plan is initially created (prepared), the tension value is set to 0 by default. Second, if a motor plan does not run immediately after it has been created, then its tension will increase. The motor plan accumulates its "desire to run" by adding to the value of tension. On the other hand, when a motor plan starts to run, its tension decreases. The motor plan releases its tension when its "desire to run" is satisfied.

We designed the tension to change quickly, whether increasing or decreasing. That means its value is time sensitive. Even at an interval of 1 ms, change in tension is still observable. Based on this design, a motor plan's tension increases quickly before the motor plan starts to run, decreasing as quickly afterward.

In the extended SMS, the motor plan may be created (prepared) via the priming process before the arrival of the selected behavior. In this case, that motor plan has to wait to run until the arrival of a selected behavior. The value of a motor plan's tension is a function of this waiting time. Different motor plans might have different tensions at a moment if they have waited over different times.

The starting of a motor plan

In the extended Sensory Motor System (SMS), because of the expectation and the earlier sensory data, a motor plan may be created before action execution starts; in other words, the motor plan does not always start to run immediately after its creation. The arrival of a selected behavior acts as a trigger that starts the motor plans to run.

Furthermore, the selected behavior may provide a specified power to its associated motor plan so that the following generated motor commands will have a specified value of the force applied to the actuators. Note that both the selected behavior and the motor plan's tension may

give the relevant motor commands some amount of force. On the other hand, when a selected behavior arrives, the force contributed by the behavior will last throughout the movement, while the force given by the tension will die away very quickly since the tension's value decreases quickly when a motor plan starts to run.

Cooperation among multiple motor plans

If there are multiple motor plans ready when the selected behavior arrives, all of the motor plans will begin to run, generating their motor commands. However, depending on the tension value of a motor plan and whether the selected behavior associates to it, the motor commands generated may have different values of their forces.

If the motor commands generated by different motor plans are applied to the same actuator, for example a finger, then the total force applied from the different commands will be combined together. The command with larger force will have greater control over the actuator.

By using this extended SMS in LIDA, we can explain and replicate the priming effects reported in the human experiment (T. Schmidt, 2002).

We can replicate the "wrong" finger movement in the inconsistent situation as described above. In the primes phase, the perceived sensory data, the target (red) prime, triggers the preparation of a motor plan generating motor commands for moving to the target prime location. When this motor plan is prepared (created), its tension begins to increase quickly. Later, in the masks phase, another target (red) mask appears. It triggers the preparation of another motor plan as well, that generates motor commands for moving to the target mask location. This second motor plan's tension also quickly begins to increase. Since the motor plan moving to the target prime was created earlier than the motor plan moving to the target mask, the former motor plan will have larger tension than the later one.

When the two motor plans start to run, they generate motor commands moving to their own specified target locations. In the inconsistent situation, the targets (red) set in the primes and the masks phases are in the opposite location, and so the relevant motor commands are trying to move the finger in opposite directions. Because the earlier (prime) motor plan has accumulated its tension for longer than the later (mask) motor plan, the motor commands moving to the target prime will have stronger force, resulting in the finger initially moving towards to the target prime (wrong) location.

Then after a short while, both motor plans' tension disappear, so that the motor command force given by the tensions drops to zero. But the motor plan associated with the selected behavior is given a specified power, and so it generates motor commands with specified force lasting during the movement. In the context of the LIDA Model, the selected behavior is recruited by the sensory data of the target mask filtered through the understanding and attention phases, so the motor commands moving to the mask target have a constant specified force. The end result is that motor commands moving to the target prime lose all of their force, while the commands to the target mask still have some amount of force, so the finger is corrected to move to the mask target (correct) location during the masks phase.

We can also replicate the phenomenon of the magnitude of the priming effect increasing with the length of the blank phase. As shown in Figure 26, in the inconsistent situation, longer SOA gives more wrong direction movement. In our extended SMS within LIDA, a longer blank phase gives the prepared (prime) motor plan more time to accumulate its tension, and more tension causes more force assigned to the motor commands moving to the target prime (wrong) location. This results in more wrong direction movement.

Simulation experiments

We prepared a simulation of the environment reported in the human experiment (T. Schmidt, 2002). Following that, we created a LIDA-based agent using the LIDA Framework with some customized configurations, including an implementation of our extended sensory motor system (SMS). We ran an experiment to replicate the priming effect reported in the human experiment.

The setup of the simulation

Using graphic libraries provided by Java[2], we simulated the dark background pictures to make a four-phase environment (see Figure 27). We prepared different pictures for the primes phase so simulating both consistent and inconsistent situations. Besides the middle white dot, red and green disk, and annular stimuli, we added a yellow rectangle to represent the simulated finger's current location during the movement. We programmed the four phases presented in the same order and durations as in the human experiment.

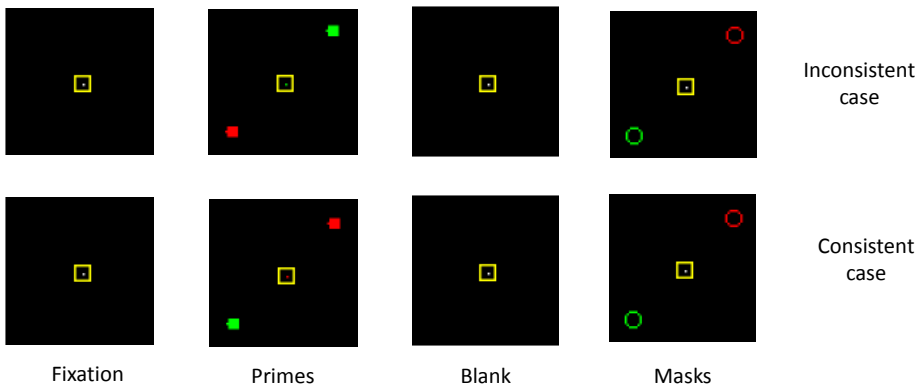


Figure 27. The simulated environment

[2] Check the package java.awt for more details at https://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html

Simulation time was measured in *ticks*, an artificial time unit created in the LIDA Framework (Snaider et al., 2011). One tick represents one millisecond of experimental time.

We configured things such that (1) the duration of the blank phase could be specified from 0 to 50 ticks with 10 ticks as interval, and (2) the environment could be specified as either inconsistent or consistent. So there were 12 types of environmental configurations in total.

Also, we have developed a program (the driver) to control the position of the yellow rectangle, representing the movement of the finger, to which the agent could send the motor commands. Two types of the forces are supposed to be applied to this finger. Each of these forces controls the finger moving towards one of the two possible target locations, either top right or lower left.

Replicating the priming effect using the extended SMS in LIDA

Method

Using a LIDA-based agent including the extended SMS, we ran the finger movement under each of the 12 environmental configurations. For each type of environment, we ran the finger movement for 1,000 trials. Participants have done a similar amount of trials in the human experiment (T. Schmidt, 2002) as well.

We calculated the distance between the simulated finger location and the target location over time during the movement for every trial. So we had collected 12,000 movement trajectories over time for the 12 environmental configurations.

Because the modules implemented in the LIDA Model are asynchronous, our LIDA-based agent's moving behaviors are not 100% deterministic in the view of time and space (Franklin et al., 2016). Thus the collected trajectories may differ among every 1,000 trials, and especially the running time of the 1,000 trials are not always the same. We did a pre-process to

clean up the collected raw data. For every 1,000 trials we discarded a few of them if their running times are too far away from the mean (> 1 std) of the 1,000 trials. Table 4 shows exactly how many trials we used from each 1,000 for the analysis. Totally we used 11,627 trials in our analysis and it is about 96.9% of the entire raw data. Schmidt performed a similar pre-process in the human experiment (1.86% of the total trials were discarded).

Table 4. The cleaned trials for the replication

	Blank 0 ticks	Blank 10 ticks	Blank 20 ticks	Blank 30 ticks	Blank 40 ticks	Blank 50 ticks
Consistent	988	991	970	967	992	989
Inconsistent	945	933	955	964	975	958

Results

In the simulation's masks phase, when using the consistent environmental configurations, the simulated finger directly moves to the target (red) location, while when using the inconsistent configurations, the finger initially moved towards the wrong location and then changed to the target location. This is qualitatively the same as the results reported from the human experiment (2002).

As shown in Figure 28 (x-dim: time and y-dim: distance), we plotted the average trajectories based on the above cleaned simulated data. This result qualitatively resembles movement trajectories of the human participants, as shown in Figure 26. The simulated trajectories are in two groups, consistent and inconsistent parts. For the inconsistent part, the trajectories have maximum amplitude indicating the movement correction moments; also, the longer blank duration makes the movement initially further towards the wrong direction (the magnitude of the priming effect increasing with the blank durations).

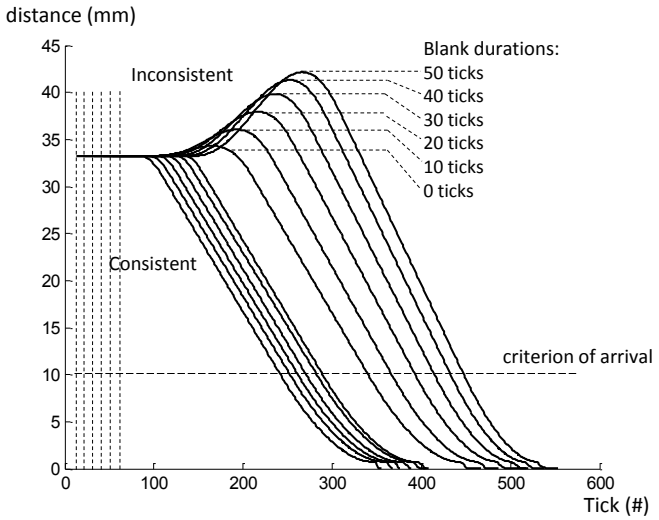


Figure 28. The distance between the simulated finger and the target location over time during the movement. Trajectories are aligned on prime onset and drawn on mask onset. Dotted vertical lines indicate the various onsets of masks.

We calculated standard deviations at the times of maximum amplitude and arrival for the trajectories of the inconsistent part as shown in Table 5. These deviations have also been reported by Schmidt (T. Schmidt, 2002). Our results are comparable to his.

Table 5. The standard deviations at the times of maximum amplitude and arrival for the trajectories of the inconsistent part

	Blank 0 ticks	Blank 10 ticks	Blank 20 ticks	Blank 30 ticks	Blank 40 ticks	Blank 50 ticks
Max Amp	0.3183	0.50359	1.5397	1.7098	2.0049	2.3702
Arrival	1.0077	1.1855	1.6722	0.8926	0.91699	1.0466

In the experimental results with humans, the maximal finger moving speeds are very similar: the maximal slopes of the all trajectories shown in the Figure 26 are very close. For each of our simulated movements, we limited the same maximal total force that could be applied on the finger. This setting indirectly sets a maximal speed the finger could move during the

movement. So in our simulation, a finger will reach a maximal speed after the initial part and before the final arrival part (see Figure 28).

Furthermore, the trajectories' slopes shown in Figure 26 are slightly different; especially the inconsistent trajectories have higher slope than the consistent trajectories. We think that is because the participants were instructed to move their fingers as soon as possible, so a participant would correct his or her movement with higher maximal speed when the finger was moving further towards to the wrong direction. We simulated this feature by dynamically tuning the maximal speed set to a movement: in the inconsistent case, longer blank duration will have a little bit higher maximal speed set. The result shown in Figure 28 included this small difference.

Borrowed from the human experiment, we also simulated a so called arrival period (the distance <= 10 mm). From Figure 26, we found that in this period, the decreasing slopes of the trajectories were not fixed and that they dynamically slowed down. To simulate this feature, we simply used a linear function to decrease the changing rate over time. This implementation is not biologically inspired but just an engineering simulation. Since this feature of the changing rate is not one of our major research interests, it is fine to mimic the trajectories occurring in the arrival period using an engineering simulation.

In Figure 26, among the all trajectories, their starting parts are slightly different in the y dimension. Schmidt (2002) did not discuss this difference. We think it could be because that in the beginning of the movement, the participants did not put their fingers on the exactly the same point on the screen but within a very small area around the initial white dot. Since this difference was not discussed in the original paper (2002) and was not among the major phenomena we planned to study, we did not simulate that.

Conclusion

Summary

The LIDA systems level cognitive model attempt to model minds. This should include modeling priming, including motor priming, so as to explain and predict motor priming phenomena. We have updated the LIDA Model so as to be able to computationally replicate an unconscious priming effect on motor control as reported from a human experiment (T. Schmidt, 2002). In order to successfully carry out this replication, we had to extend the sensory motor system (SMS) of the LIDA cognitive model including introducing the notion of tension.

We have created a LIDA-based agent including the extended SMS. In order to simulate the priming effect, we needed to introduce a variable, tension, in the extended SMS. The value of this tension variable influences the extent of the priming effect among the different configurations of the experimental environments. With these extensions to its SMS, the LIDA Model was able to successfully explain and predict the results of Schmidt's experiment, something it could not do without them.

Limitation and future work

In the experimental results with humans as reported in Figure 26, the all finger movements start at close to the same time. But in our simulated result as shown in Figure 28, the movements start at distinctly different moments. In our simulation, if the mask stimulus, the consciously reportable one, arrives later, then the finger movement starts later. This is because in our simulation, the timing of the conscious process has not been affected by the priming, so that it has a fixed duration. But in the human experimental results (Figure 26), the target mask onset varies among different experimental conditions, but the movements start at close to the same time. Based on this, we assume that the priming, in the context of the experimental instructions

given to the participants, affects the following conscious process so as to somehow make the conscious process have a dynamic duration dependent on the time period between the prime and the target mask onset.

This discussion about the effect of the priming on the following conscious process is outside of the scope of our current study, where we study the priming effect on motor control (unconsciously). We plan to continue studying this issue in the future.

8. Contributions

Action execution plays a critical role in the modeling of human level agents. It allows the agent to actually act in the environment for particular goals (agendas), over time to affect what it senses in future (Franklin & Graesser, 1997). As introduced in Chapter 1, low-level environmental information is needed for action execution, although that information may not be understandable. Based on the hypothesis of two visual systems (Goodale & Milner, 1992; Milner & Goodale, 2008), the low-level environmental information may pass directly from the agent's sensory memory to its motor control mechanism, e.g., the SMS in LIDA. Furthermore, action execution is not a simple reflex in response to a stimulus; it is driven by the agent's goals, and transformed by the agent's goal-directed action selection. A neuroscience hypothesis states that the agent's covert desires are supposed to be taken over by an overt action execution mechanism (Jeannerod, 2006). In this way, action execution implements one aspect of action, "how to do it", which naturally connects to another aspect, "what to do".

We have reviewed a set of (systems level) cognitive architectures in Chapter 3, especially with regards to action execution. The generic characteristics of action execution for cognitive architectures have been discussed here. Also, we found that prior to the present work, none of these architectures implement the aspect of "how to do it" fully. For example, ACT-R typically does not have a channel that directly links sensory memory to the motor control. And in Soar, the output motor commands are not executable in the environment; another domain-dependent program is always necessary to execute the commands.

Therefore, we have created the Sensory Motor System (SMS) for LIDA, to implement how an action is executed in LIDA. This work has been introduced in Chapter 4. As the core of the SMS, we have adopted the subsumption architecture, which generates executable motor commands driven by the arrival of sensory data. Also, we have enhanced the original

subsumption architecture by (1) adding variables to manipulate motor command values dynamically and (2) connecting a goal-directed action, the selected behavior in LIDA, to this architecture, so as to combine its reactive structure with a systems level cognitive architecture.

A computational SMS has been implemented for the execution of a grip behavior, and its simulated results have been compared to human data. This biologically inspired design, together with a computational verification by the comparison of model and human behaviors, supports the SMS as a qualitatively reasonable cognitive model for action execution.

We have modeled estimation in action execution, as introduced in Chapter 5,adding a computational model of the sensorimotor integration process (Wolpert et al., 1995) into the SMS. Wolpert and colleagues have provided the experimental results of human hand movement studies to support this model (1995). We replicated these results using a LIDA-based agent embedded with the SMS having the estimation process implemented. Furthermore, we have extended this sensorimotor integration process by using memory of errors (Herzfeld et al., 2014), enabling our model to replicate more features of human movement estimation.

Learning in LIDA, inspired by Global Workspace Theory (GWT) (Baars, 1988, 2002), is instigated by the broadcasting of the contents of consciousness to the rest of system during each cognitive cycle; this conscious content cues the update of relevant data stored in the various long-term memories. We have implemented an additional kind of learning in LIDA, sensorimotor learning, which follows the ideas of GWT, and uses reinforcement learning. It allows a dynamical selection of the motor commands at runtime based on the newly experienced rewards of the commands. Also, we have added a dynamic learning rate for this learning, which is controlled by a history of errors. This sensorimotor learning is introduced in Chapter 6.

Finally as shown in Chapter 7, I have further extended the SMS by adding a model of how earlier sensory data affects the current movement, as inspired by human experiments (T. Schmidt, 2002) studying the priming process occurring in motor control.

I list a contribution table below (Table 6), of Contributions and citations to their published descriptions.

Table 6. Contributions and citations to their published references

No	Contributions	References
1	A review of action execution (Ch. 3)	Dong, D., & Franklin, S. (2014). The Action Execution Process Implemented in Different Cognitive Architectures: A Review. Journal of Artificial General Intelligence, 5(1), 47-66.
2	The development of the Sensory Motor System (SMS) for LIDA (Ch. 4)	Dong, D., & Franklin, S. (2014b). Sensory Motor System: Modeling the process of action execution. Paper appeared in the Proceedings of the 36th Annual Conference of the Cognitive Science Society, Quebec, Canada, 2145-2150.
3	The extension of the subsumption architecture (Ch. 4)	
4	A LIDA-based agent to execute a grip action (Ch. 4)	Dong, D., & Franklin, S. (2015). A New Action Execution Module for the Learning Intelligent Distribution Agent (LIDA): The Sensory Motor System. Cognitive Computation, 7(5), 552-568.
5	Adding an internal model into the SMS (Ch. 5)	Dong, D., Franklin, S., & Agrawal, P. (2015). Estimating Human Movements Using Memory of Errors. Procedia Computer Science, 71, 1-10.
6	The extension of the internal model with the concept of memory of errors (Ch. 5)	
7	Adding sensorimotor learning into the SMS (Ch. 6)	Dong, D., & Franklin, S. (2015). Modeling Sensorimotor Learning in LIDA Using a Dynamic Learning Rate. Biologically Inspired Cognitive Architectures, 14, 1-9.
8	A dynamic learning rate for the sensorimotor learning (Ch. 6)	
9	An extension of the SMS for modeling motor priming in LIDA (Ch. 7)	Agrawal, P., Dong, D., & Franklin, S. (submitted in 2016). Modeling Motor Priming in LIDA. Cognitive Science.

Limitations and Future Work

As introduced in Chapter 5, we have modeled an estimation process in action execution, implemented by adding a sensorimotor integration process into the Sensory Motor System (SMS). In this model, the efference copy hypothesis (Jeannerod, 2006; Von Holst & Mittelstaedt, 1950) suggests that people use the difference between real and estimated sensory data to distinguish self-produced changes from externally produced ones. But this purpose, a single SMS is definitely not adequate. We need other module(s) responding to the difference sent out from the SMS to build up new understandings, such as the knowledge of "self" vs. "non-self". One direction for future work is to explore further towards manipulation of other cognitive modules to model the human capability of distinguishing self-produced changes from externally produced changes.

As introduced in Chapter 6, we have implemented sensorimotor learning in LIDA, and we have added a dynamic learning rate into the learning. Also, we have built this learning into a LIDA-based agent to examine how it affects the agent's capability of pushing, a particular type of action execution. But one major limitation for this experiment is that the motor commands of the agent are very simple, and the execution of each of these commands typically can be done within one step of the agent's run. This entails that the agent can predict the execution result of a motor command very well—its predicted result will very often be approximately the same as the sensory result. Under this condition, it is hard to model for study the motor command error, the difference between the predicted and the sensory result. Thus another direction for future work is application of the currently implemented LIDA-based controller to another robot that provides more complicated motor commands, which we expect will produce larger motor errors that allow better examination of our model.

References

ACT-R 6.0 Tutorial. (2012). Unpublished manuscript. Retrieved from http://act-r.psy.cmu.edu/wordpress/wp-content/themes/ACT-R/actr6/actr6.zip

Albus, J. S., & Barbera, A. J. (2005). RCS: A cognitive architecture for intelligent multi-agent systems. *Annual Reviews in Control, 29*(1), 87-99.

Anderson, J. (2007). *How can the human mind occur in the physical universe?* : Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review, 111*(4), 1036-1060.

Arrabales, R., Ledezma, A., & Sanchis, A. (2009). *CERA-CRANIUM: A test bed for machine consciousness research*. International Workshop on Machine Consciousness.

Auger, F., Hilairet, M., Guerrero, J. M., Monmasson, E., Orlowska-Kowalska, T., & Katsura, S. (2013). Industrial applications of the kalman filter: A review. *Industrial Electronics, IEEE Transactions on, 60*(12), 5458-5471.

Baars, B. J. (1988). *A cognitive theory of consciousness*. New York: Cambridge University Press.

Baars, B. J. (2002). The conscious access hypothesis: origins and recent evidence. *Trends in cognitive sciences, 6*(1), 47-52.

Berner, J., Schönfeldt-Lecuona, C., & Nowak, D. A. (2007). Sensorimotor memory for fingertip forces during object lifting: the role of the primary motor cortex. *Neuropsychologia, 45*(8), 1931-1938.

Bothell, D. (n.d.). *ACT-R 6.0 Reference Manual (Working Draft)*. Retrieved from http://act-r.psy.cmu.edu/wordpress/wp-content/themes/ACT-R/actr6/reference-manual.pdf

Bowling, M., & Veloso, M. (2001). *Rational and convergent learning in stochastic games.* Paper presented at the International joint conference on artificial intelligence (1021-1026).

Breitmeyer, B. G. (2014). Contributions of magno-and parvocellular channels to conscious and non-conscious vision. *Philosophical Transactions of the Royal Society of London B: Biological Sciences, 369*(1641), 20130213.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation, 2*(1), 14-23.

Brooks, R. A. (1990). Elephants don't play chess. *Robotics and autonomous systems, 6*(1), 3-15.

Brooks, R. A. (1991). How to build complete creatures rather than isolated cognitive simulators. *Architectures for Intelligence: The Twenty-second Carnegie Mellon Symposium on Cognition*, 225-239.

Brooks, R. A., Connell, J., & Ning, P. (1988). Herbert: A second generation mobile robot.

Budiu, R. (2013). ACT-R Website. from http://act-r.psy.cmu.edu/

Byrne, M. D., & Anderson, J. R. (2001). Serial modules in parallel: The psychological refractory period and perfect time-sharing. *Psychological Review, 108*(4), 847-869.

Castiello, U. (2005). The neuroscience of grasping. *Nature Reviews Neuroscience, 6*(9), 726-736.

Connell, J. H. (1989a). A behavior-based arm controller. *Robotics and Automation, IEEE Transactions on, 5*(6), 784-791.

Connell, J. H. (1989b). A colony architecture for an artificial creature: DTIC Document.

Dawson, M. R. W. (n.d.). Dawson's Margin Notes On "Understanding Intelligence" (Rolf Pfeifer, Christian Scheier, 2001). from http://www.bcp.psych.ualberta.ca/~mike/Pearl_Street/Margin/Pfeifer/chap7.html

De Vega, M., Glenberg, A. M., & Graesser, A. C. (2008). *Symbols and embodiment: Debates on meaning and cognition*: Oxford University Press, USA.

Deplancke, A., Madelain, L., Gorea, A., & Coello, Y. (2013). Perception-action dissociations depend on the luminance contrast of the stimuli. *Journal of neurophysiology, 110*(8), 1974-1983.

Dickmanns, E. D. (1992). A general dynamic vision architecture for UGV and UAV. *Applied Intelligence, 2*(3), 251-270.

Dickmanns, E. D. (2000). *An expectation-based, multi-focal, saccadic (EMS) vision system for vehicle guidance.* Paper presented at the International Symposium of Robotics Research (ISRR'99) (421-430). Snowbird Utah, USA.

Dong, D., & Franklin, S. (2014a). The Action Execution Process Implemented in Different Cognitive Architectures: A Review. *Journal of Artificial General Intelligence, 5*(1), 47-66. doi: 10.2478/jagi-2014-0002

Dong, D., & Franklin, S. (2014b). *Sensory Motor System: Modeling the process of action execution.* Paper presented at the Proceedings of the 36th Annual Conference of the Cognitive Science Society (2145-2150). Quebec, Canada.

Dong, D., & Franklin, S. (2015a). Modeling Sensorimotor Learning in LIDA Using a Dynamic Learning Rate. *Biologically Inspired Cognitive Architectures, 14*, 1-9. doi: 10.1016

Dong, D., & Franklin, S. (2015b). A New Action Execution Module for the Learning Intelligent Distribution Agent (LIDA): The Sensory Motor System. *Cognitive Computation, 7*(5), 552-568. doi: 10.1007/s12559-015-9322-3

Dong, D., Franklin, S., & Agrawal, P. (2015). Estimating Human Movements Using Memory of Errors. *Procedia Computer Science, 71*, 1-10. doi: 10.1016

Drescher, G. L. (1991). *Made-up minds: a constructivist approach to artificial intelligence*: MIT press.

Duch, W., Oentaryo, R. J., & Pasquier, M. (2008). *Cognitive Architectures: Where do we go from here?* Paper presented at the AGI (122-136). Memphis TN, USA.

Faghihi, U., McCall, R., & Franklin, S. (2012). A computational model of attentional learning in a cognitive agent. *Biologically Inspired Cognitive Architectures, 2*, 25-36.

Farnè, A., Pavani, F., Meneghello, F., & Làdavas, E. (2000). Left tactile extinction following visual stimulation of a rubber hand. *Brain, 123*(11), 2350-2360.

Flanagan, J. R., Bittner, J. P., & Johansson, R. S. (2008). Experience can change distinct size-weight priors engaged in lifting objects and judging their weights. *Current Biology, 18*(22), 1742-1747.

Franklin, S., & Graesser, A. (1997). Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents *Intelligent agents III agent theories, architectures, and languages* (pp. 21-35). London, UK: Springer-Verlag.

Franklin, S., Madl, T., D'Mello, S., & Snaider, J. (2014). LIDA: A Systems-level Architecture for Cognition, Emotion, and Learning. *IEEE Transactions on Autonomous Mental Development, 6*(1), 19-41. doi: 10.1109/TAMD.2013.2277589

Franklin, S., Madl, T., Strain, S., Faghihi, U., Dong, D., Kugele, S., Snaider, J., Agrawal, P., & Chen, S. (2016). A LIDA cognitive model tutorial. *Biologically Inspired Cognitive Architectures*, 105-130. doi: 10.1016/j.bica.2016.04.003

Goertzel, B., Lian, R., Arel, I., De Garis, H., & Chen, S. (2010). A world survey of artificial brain projects, Part II: Biologically inspired cognitive architectures. *Neurocomputing, 74*(1), 30-49.

Goertzel, B., & Pennachin, C. (2007). *Artificial general intelligence* (Vol. 2): Springer.

Gonzalez Castro, L. N., Hadjiosif, A. M., Hemphill, M. A., & Smith, M. A. (2014). Environmental consistency determines the rate of motor adaptation. *Current Biology, 24*(10), 1050-1061.

Goodale, M. A., & Milner, A. D. (1992). Separate visual pathways for perception and action. *Trends in neurosciences, 15*(1), 20-25.

Grafton, S. T. (2010). The cognitive neuroscience of prehension: recent developments. *Experimental brain research, 204*(4), 475-491.

Grainger, J. E., Scharnowski, F., Schmidt, T., & Herzog, M. H. (2013). Two primes priming: Does feature integration occur before response activation? *Journal of vision, 13*(8), 19-19.

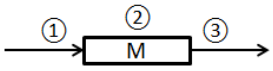
Herzfeld, D. J., Vaswani, P. A., Marko, M. K., & Shadmehr, R. (2014). A memory of errors in sensorimotor learning. *Science, 345*(6202), 1349-1353.
Jakobson, L., Archibald, Y., Carey, D., & Goodale, M. A. (1991). A kinematic analysis of reaching and grasping movements in a patient recovering from optic ataxia. *Neuropsychologia, 29*(8), 803-809.
James, T. W., Culham, J., Humphrey, G. K., Milner, A. D., & Goodale, M. A. (2003). Ventral occipital lesions impair object recognition but not object‐directed grasping: an fMRI study. *Brain, 126*(11), 2463-2475.
Jeannerod, M. (1981). Intersegmental coordination during reaching at natural visual objects. *Attention and performance IX, 9*, 153-168.
Jeannerod, M. (2006). *Motor cognition: What actions tell the self*. Oxford, UK: Oxford University Press.
Jeannerod, M., Decety, J., & Michel, F. (1994). Impairment of grasping movements following a bilateral posterior parietal lesion. *Neuropsychologia, 32*(4), 369-380.
Jenmalm, P., Schmitz, C., Forssberg, H., & Ehrsson, H. H. (2006). Lighter or heavier than predicted: neural correlates of corrective mechanisms during erroneously programmed lifts. *The Journal of neuroscience, 26*(35), 9015-9021.
Johansson, R. S., & Flanagan, J. R. (2009). Coding and use of tactile signals from the fingertips in object manipulation tasks. *Nature Reviews Neuroscience, 10*(5), 345-359.
Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *arXiv preprint cs/9605103*.
Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of Fluids Engineering, 82*(1), 35-45.
Kane, A., Wade, A., & Ma-Wyatt, A. (2011). Delays in using chromatic and luminance information to correct rapid reaches. *Journal of vision, 11*(10), 3-3.
Khamassi, M., Enel, P., Dominey, P. F., & Procyk, E. (2013). Medial prefrontal cortex and the adaptive regulation of reinforcement learning parameters. *Prog Brain Res, 202*, 441-464.
Kieras, D. E., & Meyer, D. E. (1996). The EPIC architecture: Principles of operation. *Unpublished manuscript from ftp://ftp.eecs.umich.edu/people/kieras/EPICarch.ps*.
Kieras, D. E., & Meyer, D. E. (1997). An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. *Human-computer interaction, 12*(4), 391-438.
Körding, K. P., & Wolpert, D. M. (2006). Bayesian decision theory in sensorimotor control. *Trends in cognitive sciences, 10*(7), 319-326.
Laird, J. (2008). *Extending the Soar cognitive architecture.* Paper presented at the In proceedings of the artificial general inteligence conference (224-235). Memphis TN, USA.
Laird, J. (2012). *The Soar cognitive architecture*: MIT Press.
Laird, J. E., Congdon, C. B., Coulter, K. J., Derbinsky, N., & Xu, J. (2012). *The Soar User's Manual Version 9.3.2*. Computer Science and Engineering Department. University of Michigan. Unpublished manuscript.
Langley, P., & Choi, D. (2006). *A unified cognitive architecture for physical agents.* Paper presented at the Proceedings of the National Conference on Artificial Intelligence (1469-1474). Boston MA, USA.
Langley, P., Laird, J. E., & Rogers, S. (2009). Cognitive Architectures: Research Issues and Challenges. *Cognitive Systems Research, 10*(2), 141-160. doi: doi: 10.1016/j.cogsys.2006.07.004
Maes, P. (1989). How to do the right thing. *Connection Science, 1*(3), 291-323.
Mahadevan, S., & Connell, J. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artificial intelligence, 55*(2), 311-365.
Maybeck, P. S. (1979). *Stochastic models, estimation, and control* (Vol. 1). New York: Academic Press.
Milner, D., & Goodale, M. A. (2008). Two visual systems re-viewed. *Neuropsychologia, 46*(3), 774-785.
Milner, D., Perrett, D., Johnston, R., Benson, P., Jordan, T., Heeley, D., Bettucci, D., Mortara, F., Mutani, R., & Terazzi, E. (1991). Perception and action in 'visual form agnosia'. *Brain, 114*(1), 405-428.

Newell, A. (1973). You can't play 20 questions with nature and win: Pro- jective comments on the papers of this symposium. In W. G. Chase (Ed.), *Visual information processing*. New York: Academic Press.
Piaget, J., Brown, T., & Thampy, K. J. (1985). *The equilibration of cognitive structures: The central problem of intellectual development* (Vol. 985): University of Chicago Press Chicago.
Pulaski, M. A. S. (1980). *Understanding Piaget: An Introduction to Children's Cognitive Development*. New York: HARPER & ROW.
Rohrer, B. (2012). *BECCA: Reintegrating AI for natural world interaction.* Paper presented at the AAAI Spring Symposium on Designing Intelligent Robots: Reintegrating AI. Stanford California, USA.
Russell, S. J., & Norvig, P. (2009). *Artificial intelligence: a modern approach* (Third ed.): Prentice hall.
Samsonovich, A. V. (2010). Toward a Unified Catalog of Implemented Cognitive Architectures. *BICA, 221*, 195-244.
Schmidt, F., Weber, A., & Schmidt, T. (2014). Activation of response force by self-splitting objects: Where are the limits of feedforward Gestalt processing? *Journal of vision, 14*(9), 20-20.
Schmidt, T. (2002). The finger in flight: Real-time motor control by visually masked color stimuli. *Psychological Science, 13*(2), 112-118.
Searle, J. R. (1983). *Intentionality: An essay in the philosophy of mind*: Cambridge University Press.
Shapiro, S. C., & Bona, J. P. (2010). The GLAIR cognitive architecture. *International Journal of Machine Consciousness, 2*(2), 307-332.
Simon, H. A. (1969). *The sciences of the artificial*. Cambridge, MA: The MIT Press.
Snaider, J., McCall, R., & Franklin, S. (2011). The LIDA framework as a general tool for AGI *Artificial General Intelligence* (pp. 133-142). Berlin Heidelberg: Springer
Sun, R. (2003). A tutorial on CLARION 5.0. http://www.cogsci.rpi.edu/~rsun/sun.tutorial.pdf
Sun, R. (2006). The CLARION cognitive architecture: Extending cognitive modeling to social simulation. In R. Sun (Ed.), *Cognition and multi-agent interaction* (pp. 79-99). New York: Cambridge University Press.
Tafazoli, S., Di Filippo, A., & Zoccolan, D. (2012). Transformation-tolerant object recognition in rats revealed by visual priming. *The Journal of neuroscience, 32*(1), 21-34.
Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic robotics*: MIT press.
Todorov, E., & Jordan, M. I. (2002). Optimal feedback control as a theory of motor coordination. *nature neuroscience, 5*(11), 1226-1235.
Von Holst, E. (1954). Relations between the central nervous system and the peripheral organs. *The British Journal of Animal Behaviour, 2*(3), 89-94.
Von Holst, E., & Mittelstaedt, H. (1950). Das Reafferenzprinzip: Wechselwirkungen zwischen Zentralnervensystem und Peripherie. *Naturwissenschaften, 37*, 464-476.
Wang, P., Goertzel, B., & Franklin, S. (2008). *Artificial General Intelligence 2008: Proceedings of the First AGI Conference* (Vol. 171): IOS Press.
Watkins, C. J. C. H. (1989). *Learning from delayed rewards.* University of Cambridge.
Welch, G., & Bishop, G. (2006). An introduction to the Kalman filter: Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC 27599-3175.
Wolpert, D. M., Diedrichsen, J., & Flanagan, J. R. (2011). Principles of sensorimotor learning. *Nature Reviews Neuroscience, 12*(12), 739-751.
Wolpert, D. M., & Ghahramani, Z. (2000). Computational principles of movement neuroscience. *nature neuroscience, 3*, 1212-1217.
Wolpert, D. M., Ghahramani, Z., & Jordan, M. I. (1995). An internal model for sensorimotor integration. *Science, 269*(5232), 1880-1882.
www.cyberbotics.com. Webots, a commercial mobile robot simulation software developed by Cyberbotics Ltd.

Appendix A

Table 7. Three types of Herbert's arm controller components and their simulated pseudo codes

Name	Design diagram	Pseudo code (Java)

1. Module		FSMImpl extends FrameworkTask{
		① receiveData (Input){ … };
		② runThisFrameworkTask(){
		execute();
		}
		execute(){ … switch (state) {…}};
		③ Cmd output(){ … };
		}

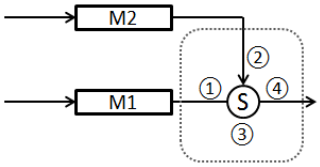
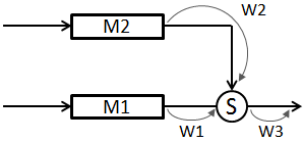
2. Suppress node		suppress extends FrameworkTask{
		① inputLowerLayer(LowerInput)
		{ … };
		② inputHigherLayer(HigherInput)
		{ … };
		③ runThisFrameworkTask(){
		if (HigherInput != Null)
		Cmd = HigherInput;
		else
		Cmd = LowerInput;
		}
		④ Cmd output() { … };
		}

Table 7. Three types of Herbert's arm controller components and their simulated pseudo codes

Name	Design diagram	Pseudo code (Java)

3. Wire		W1→ wire1 extends FrameworkTask{
		runThisFrameworkTask(){
		Ⓢ.inputLowerLayer(M1.output));
		}
		}
		
		W2→ wire2 extends FrameworkTask{
		runThisFrameworkTask(){
		Ⓢ.inputHigherLayer(M2.output));
		}
		}
		
		W3→ wire3 extends FrameworkTask{
		runThisFrameworkTask(){
		commands = Ⓢ.output;
		}
		}

Appendix B

The software architecture for the simulated Herbert arm controller is shown in Figure 29. The module component, depicted on the right, originates from a LIDA Framework interface (FrameworkTask), and starts from an interface FSM that indicates common features of the module. An abstract class FSMImpl implements FSM and extends a LIDA Framework abstract class (FrameworkTaskImpl) to achieve the methods common to the modules, such as I/O and the task run. The interface ArmsFSM extends FSM to claim specific methods for arm modules, such as receiving the arm's position and moving the hand, and which are implemented in the abstract

class ArmsFSMImpl. Three hand modules (GrabFSM, OpenFSM, and DepositFSM) extend FSMImpl to implement their own agendas. Another twelve arm modules extend ArmsFSMPlmpl for different arm tasks, ExtendFSM, SurfaceFSM, and others as indicated in Figure 11.

On the top left of Figure 29 is a MPT interface representing a general Motor Plan Template (MPT). An abstract class SubsumptionMPT implements the interface to claim a type of MPT that is inspired by the principles of the subsumption architecture. SubsumptionMPT contains two types of inner classes that extend FrameworkTaskImpl, suppress and wires. Suppress simulates the suppress node used as a component of the simulated controller, and five wire classes simulate five types of wire component: wiring from a module or a suppress node to a suppress node's higher or lower input (four types), or from a suppress node to the final output (the fifth type).

A GripMPT class extends SubsumptionMPT to implement a specific MPT for a grip. It structures all three types of components shown in Figure 10—module (FSM and ArmsFSM), suppress, and wire—and provides methods for I/O and running components.

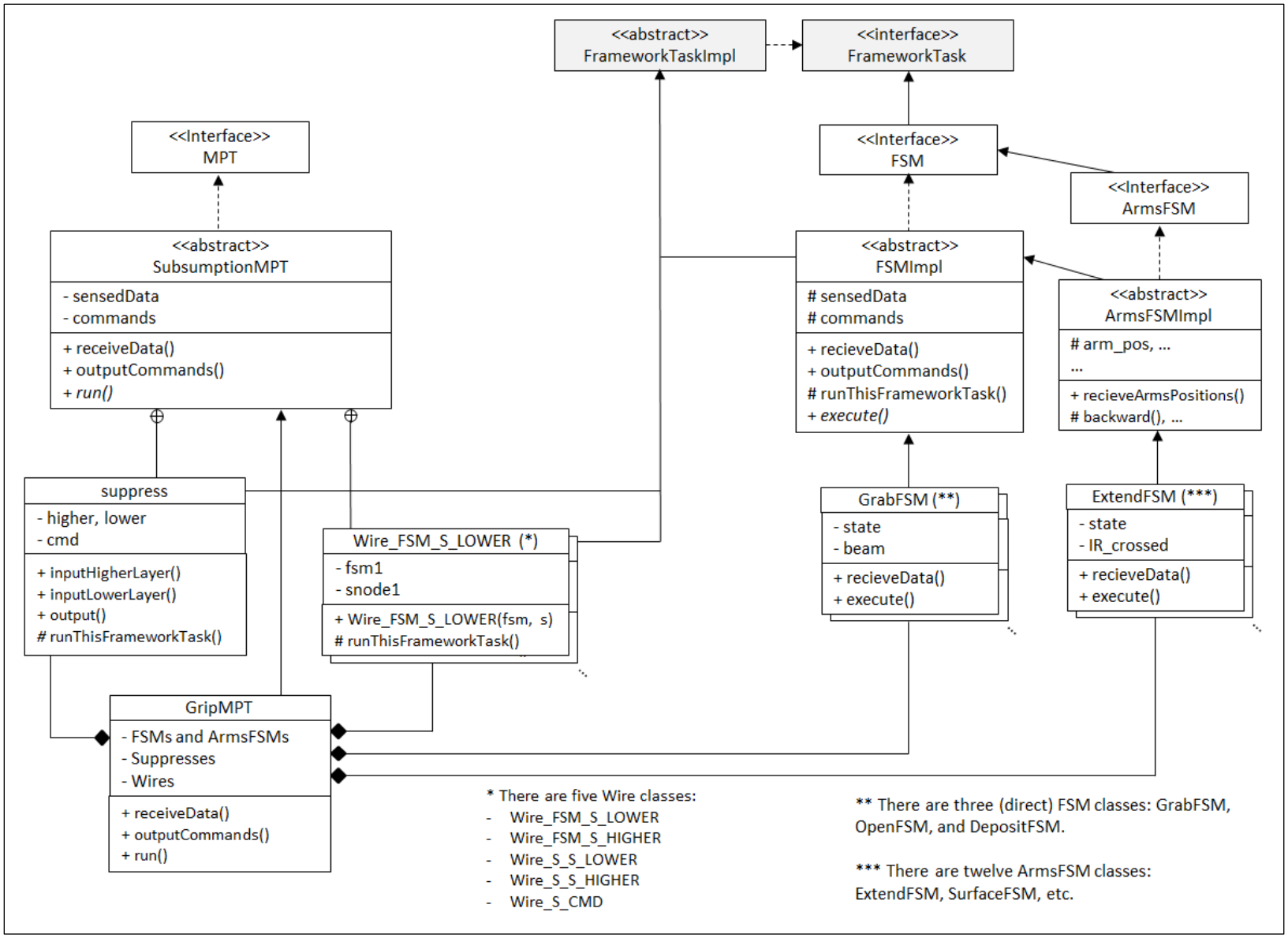


Figure 29. The software architecture for the simulated Herbert arm controller

Permission letter

SAGE PUBLICATIONS LICENSE
TERMS AND CONDITIONS

Jun 01, 2016

This Agreement between Daqi Dong ("You") and SAGE Publications ("SAGE Publications") consists of your license details and the terms and conditions provided by SAGE Publications and Copyright Clearance Center.

All payments must be made in full to CCC. For payment instructions, please see information listed at the bottom of this form.

License Number	3880560263972
License date	Jun 01, 2016
Licensed Content Publisher	SAGE Publications
Licensed Content Publication	Psychological Science
Licensed Content Title	The Finger in Flight: Real-Time Motor Control by Visually Masked Color Stimuli:
Licensed Content Author	Thomas Schmidt
Licensed Content Date	03/01/2002
Licensed Content Volume Number	13
Licensed Content Issue Number	2
Pages	7
Type of Use	Journal
Requestor type	Non-commercial
Format	Print and electronic
Portion	Figure/table
Number of figures/tables	1
Will you be translating?	No, only English
circulation	3
Title of your book	A model of how earlier sensory data affects the current movement
Publisher of your book	Biologically Inspired Cognitive Architectures
Author of your book	Agrawal, P., Dong, D., & Franklin, S.
Expected publication date of new book	Dec 2016
Estimated size of your book	10
Requestor Location	Daqi Dong 162 Legacy barn trail #201 COLLIERVILLE, TN 38017

	United States Attn: Daqi Dong
Billing Type	Credit Card
Credit card info	Visa ending in 6407
Credit card expiration	01/2019
Total	56.00 USD

Terms and Conditions

SAGE Terms and Conditions for Permissions Administered Through Rightslink

You, the Requestor, are requesting to use the material specified in the permission request (the "Work"). Your agreement to the terms and conditions herein and completion of a permission request constitutes a Permission Request. You are in no way required to use the Work; however, should you decide not to use the Work after you complete this request, you agree to cancel your order through this website. Under the above conditions, and the following terms and conditions, permission to use Work ("Permission") is granted solely to you:

1. SAGE reserves the right to revoke any Permission, at SAGE's sole discretion, within two (2) business days of the request.
2. The number of copies ("Copies") for print use is defined as the total number of copies made for distribution or for repurposing, and the number of copies ("Copies") for electronic use is defined as the total number of viewers of the Work, recipients of copies of the Work, and individuals or entities who may have access to the Work. The Copies must not exceed the Copies as stated in the Permission Request.
3. If your Permission Request is for use on a website, internet, intranet, or any publicly accessible site, you agree to remove the material from such site after six (6) months or else renew your permission request. Requests to post a full article on a website, internet, intranet, or any publicly accessible site must be submitted to the Publisher at permissions@sagepub.com.
4. Permission is granted only for the type of use specified in the Permission Request. If any information pertaining to your Permission Request changes, you must cancel this request and resubmit a new request with the correct and current permission request information. SAGE may exercise its right to revoke Permission, if SAGE finds, in SAGE's sole opinion, that the context in which you have used or repurposed the Work is considered libelous, in violation of any right of privacy, or otherwise unlawful; infringement or in violation of any copyright or other proprietary right of others; or can be construed to possibly cause harm or injury. You agree that use of Work will be professional, in the context of fact-based and generally acceptable professional practices.
5. **Permission does not include the use within Custom Publishing Programs, and all use within such programs is explicitly prohibited.**
6. **Permission does not include use of the material within Massive Open Online Courses (MOOC's). For permission to include material in a MOOC, please contact SAGE directly at permissions@sagepub.com.**
7. If your Permission Request includes the right to translate the Work, you agree that the translation of the material shall be made faithfully and accurately, and that abbreviations and/or alterations in the text and/or title of the Work shall be made only with SAGE's prior the written consent. Requestor shall not own or acquire any copyright or other proprietary rights in the Work or any other material furnished by SAGE, including without limitation translations or transcriptions thereof, all of which rights shall be owned by and/or are hereby assigned to SAGE. Requestor shall indemnify SAGE against any and all claims, including without limitation attorneys' fees and legal costs, that concern or relate to (a) inaccurate translation or transcription of the Work, (b) infringement claims arising out of the inclusion of material not furnished by SAGE or (c) infringement or other claims asserted by anyone retained by Requestor to translate the Work. Requestor agrees that the name of the Author (s), Copyright Holder, and Publisher shall appear in due prominence on the title page of every copy of the translation and in all advertisements for the translation, and that the translation

shall include: (1) the Work's original copyright notice and original American title, both in English, and (2) notice in granted translated language in identifying the SAGE Publications as the original publisher and stating the translation is published by arrangement with SAGE. The rights licensed to Requestor are not transferrable. **The translated article reprint must include the following disclaimer in English and in the language of the reprint: "While every effort has been made to ensure that the contents of this publication are factually correct, neither the authors nor the publisher accepts, and they hereby expressly exclude to the fullest extent permissible under applicable law, any and all liability arising from the contents published in this Article, including, without limitation, from any errors, omissions, inaccuracies in original or following translation, or for any consequences arising therefrom. Nothing in this notice shall exclude liability which may not be excluded by law. Approved product information should be reviewed before prescribing any subject medications."**

8. Permission is granted for prospective use only, and does not apply to any use that has occurred prior to the date of the Permission Request.
9. Permission does not apply to any material (reprints, illustrations, figures, tables, etc.) not controlled by SAGE. Requests for permission to re-use third-party material should be submitted to the original copyright holder, as indicated in the credit line for the materials.
10. Full acknowledgment of your source must appear in every copy of your work as follows:
    - Author(s), Journal Title (Journal Volume Number and Issue Number)
    - pp. xx-xx, copyright © YEAR by (Copyright Holder)
    - Reprinted by Permission of SAGE Publications, Inc.
11. Unless specified in the request or by prior arrangement with SAGE, payment is due from you within sixty (60) days after the completion of Permission.
12. It is assumed that the Requester is using the selection in question, and is subject to billing and collections procedures, unless otherwise noted.
13. Permission Requests for reuse of text excerpts shall not exceed 50% of the article's content.
14. No more than 20% of any one SAGE journal issue may be reused at one time.

Other Terms and Conditions:

v4.1

Questions? customercare@copyright.com or +1-855-239-3415 (toll free in the US) or +1-978-646-2777.