2015

A New Action Execution Module for the Learning Intelligent Distribution Agent (LIDA): The Sensory Motor System

D. Dong

S. Franklin

A New Action Execution Module for the Learning Intelligent Distribution Agent (LIDA): The Sensory Motor System

Daqi DONG[1], Stan FRANKLIN[2]

Department of Computer Science and the Institute for Intelligent Systems, University of Memphis

[1]FedEx Institute of Technology #403h, 365 Innovation Dr., Memphis, TN 38152;

E-mail: ddong@memphis.edu

[2]FedEx Institute of Technology #312h, 365 Innovation Dr., Memphis, TN 38152

Abstract. This paper presents a cognitive model for an action execution process—the Sensory Motor System (SMS)—as a new module of the systems-level cognitive model LIDA (for "Learning Intelligent Distribution Agent"). Action execution refers to a situation in which a software agent or robot transforms a selected goal-directed action into low-level executable actions, and executes them in the real world. A sensorimotor system derived from the subsumption architecture has been implemented into the SMS; several cognitive neuroscience hypotheses have been incorporated as well, including the two visual systems. A computational SMS has been created inside a LIDA-based software agent in Webots to model the execution of a grip action; its simulated results have been verified against human performance. This computational verification by the comparison of model and human behaviors supports the SMS as a qualitatively reasonable cognitive model for action execution. Finally, the SMS has been compared to other alternative models of action execution; this supports the assertion that our new action execution module is applicable across a range of cognitive architectures.

Keywords. Sensory Motor System (SMS), action execution, LIDA model, subsumption architecture, grip action, robot Herbert.

1. Introduction

In the field of cognitive modeling, the challenge of creating a real-life computational simulation of the human mind calls for efforts to develop biologically-inspired intelligent agents and robots. The LIDA[1] Model [1] is a conceptual, systems-level model of human mental processes. It has integrated understanding, attention, and action (selection) so as to achieve this mission. Here we create a mechanism called the Sensory Motor System (SMS) to model the process of action execution in LIDA.

Action presents two aspects. On the one hand, it is driven by the agent's intention. This means the agent selects the action motivated from inside 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[2] [2]. 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

[1] For historical reasons LIDA stands for Learning Intelligent Distribution Agent.
[2] In this paper we'll only be concerned with the external environment, and not with LIDA's internal environment.

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.

Milner and Goodale have proposed a hypothesis in their work on the two visual systems[3] [3-7], which supports a model for how a human maintains and integrates these two facets of action: "what to do" and "how to do it"—in other words, the understandable and the executable. 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" [4].

Following the hypothesis of the two visual systems, 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 [1]; here we specify the action execution in the form of the Sensory Motor System (SMS). 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 LIDA Model and its relationship to the SMS. Section 3 contains an overview of the subsumption architecture [8, 9], which is used as the SMS's prototype. Section 4 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. Section 5 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. In Section 6, we compare the SMS of LIDA with the action (execution) process implemented in three other cognitive architectures: Adaptive control of thought-rational (ACT-R), Soar, and Connectionist Learning with Adaptive Rule Induction ON-line (CLARION).

[3] In the LIDA Model, the concept of ventral and dorsal streams for the transmission of visual information has been extended to multimodal transmission.

We conclude with a discussion of the benefits of modeling a natural action execution process with the SMS for LIDA, followed by future plans for SMS development.

In this paper, we emphasize 1) the introduction of a new action execution module developed from the traditional subsumption architecture; and 2) the cooperation between this module and a systems-level cognitive architecture, LIDA[4]. Currently, we are working on the model of action execution learning in LIDA. This modeling work is inspired by several learning enhancements of the subsumption architecture [10, 11], current hypotheses of motor learning[12], and some arguments about limitations of the subsumption architecture [13, 14].

2 The LIDA Model

The LIDA model is a systems-level cognitive model [1, 15]. It implements and fleshes out a number of psychological and neuropsychological theories, but is primarily based on Global Workspace Theory [16-18]. 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 [16], 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 here

Ideas concerning *action execution* have been briefly proposed in the LIDA Model as well, 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 begin to do so here.

The original Sensory Motor Memory and Motor Plan Execution modules have been implemented by the SMS in detail. Two of other LIDA modules, Action Selection and Sensory Memory, provide relevant information—a

[4] Action learning is not yet implemented.

selected behavior and the sensory data through a dorsal stream channel[5]–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[6], 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 to the environment. Note that in Figure 1, the Sensory Motor Learning channel, which is issued by the Global Workspace and sent to the Sensory Motor Memory, has not yet been modeled in the SMS because the current work does not address the learning process. Future work is expected to do so.

3 The subsumption architecture

The subsumption architecture is a parallel and distributed computation formalism for connecting sensors to actuators [8, 9], 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 expect 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 [19]; the environment is used as the communication medium because " [t]he world is its own best model" [20].

Figure 2 here

The capabilities of the subsumption architecture match many required features of action execution as we plan to model it (see Section 4). 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

[5] In LIDA, the dorsal stream channel directly passes sensory data from the sensory memory to the action execution process.

[6] 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 paper, we use "action" in the general sense, while "action of a behavior" refers to a particular component of that behavior.

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 [21], 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…" [22]. We believe the concepts used in SMS are the same as Jeannerod's, although our terminologies differ.[7]

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 decomposes a robot's control architecture into a set of task-achieving behaviors or competences. Competences refer to low-level tasks; they play a role in connecting an understandable action to 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 [9, 23]. 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.

On the other hand, the subsumption architecture doesn't reflect the process of specification for the movement parameters, nor does it interact with high-level goal-directed actions, which are essential requirements of the SMS. We have extended the subsumption architecture in the SMS mainly based on the two visual systems as well as hypotheses borrowed from cognitive neuroscience. The SMS's full definition is described in the next section.

[7] '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.

4. 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 human mental and behavioral processes, and the detailed computational design of SMS.

4.1 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. Since the action execution process is running in a real world with seemingly unlimited environmental data available, much of which heavily affects the order of the motor commands in real time, 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 [24], 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 [8]. Therefore, a reactive structure is introduced to model the source of ordered motor commands. As shown in 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 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 here

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. 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 module and the SMS, which models the hypothesis regarding one of the dorsal stream's roles, online control.

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.

4.2 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. However, since MCs are the output of the SMS, a general MC format has been defined. 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.

4.3 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 4.

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 4 here

Both sensory data through the dorsal stream and the selected behavior determine the specification process. As shown in Figure 4, 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—the *update* process is another option, which will be described later. Each of these cylinders represents a set of associations; every association transforms relevant environmental features into a command value; it is based on the notion that separate visuomotor channels are activated in parallel [25]. 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" [22]. Thus, one corresponding association implemented in the SMS is to transform an object's size to the distance between gripping fingers (see Section 5.4).

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. 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 4. 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" [4]. 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 (see Section 5.4).

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 [26]. 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 4, showing that the MCs command values are updated by the values associated with the sensed data during executing the action.

4.4 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 5, MPT selection chooses one MPT from others based on the selected behavior.

Figure 5 here

5. Implementation and Experiment

Different actions execute variously, due to vastly different actuators, goals, or contexts. In other words, we need a Sensory Motor System (SMS) that allows the modeling of the action's distinctive characteristics in the execution process.We have implemented a computational SMS to model the execution of a grip action inside a LIDA-based software agent. The experimental environment and a software robot youBot are introduced in Section 5.1 and 5.2 separately. Section 5.3 introduces both an online control process and a Motor Plan (MP) for a grip action. Their implementations are inspired by the design principles of the robot Herbert's arm controller [19]. Section 5.4 introduces a specification process and a Motor Plan Template (MPT) for the grip action. Their implementations are biologically inspired, based on the results of current human grip research. The simulated grip aperture results have been compared with human grip performance. In Section 5.5, a simple MPT selection process is implemented, and the entire grip SMS is linked to LIDA.

5.1 Webots and the LIDA Framework

Webots is a mobile robot simulation software package. It offers an environment for rapid prototyping a 3D virtual world, an array of ready-made sensors and actuators, and programmable controllers controlling robots living in its virtual world (www.cyberbotics.com). We use Webots as an experimental environment in which to manipulate a software robot, the youBot (see next section), controlled by the LIDA Framework (see just below), in order to run a computational SMS for gripping.

The LIDA Framework is an underlying computational software framework. We use it to create a simulated human mind as the controller of youBot. "[The LIDA Framework] allows the creation of new intelligent software

agents and experiments based [on] the LIDA model. Its design and implementation aim to simplify this process and to permit the user to concentrate [on] the specifics of the application" [27]. The computational SMS is embedded into the Framework as a submodule for the execution of a grip.

5.2 youBot

The youBot is a software robot. As shown in Figure 6 (a), its actuators are a mobile base, an arm, and two grippers. We chose this robot on the basis of its similarity to Herbert (see Section 5.3), whose arm controller serves as the prototype of our newly created Motor Plan (MP) of grip.

Figure 6 here

5.2.1 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 6 (b) and (c)).

5.2.2 The actuators

As shown in Figure 6 (a), the youBot arm is comprised of five segments, arm0 to arm4, which are linearly connected by five joints, joint0 to joint4. 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 been fixed at a value of zero. The end segment of the arm plays the role of a hand, and the grippers are attached to it.

The arm has four basic movements used to control the hand: *lift*, *descend*, *extend*, and *back*, each of which can occur along one of two lines: up-down or back-forth. Compound movements for the arm are simulated as well, as extensions of the basic ones. One of these compound movements is to move the hand forward and slightly down; its simulation formula is developed from the basic movement *extend*, but instead of the hand being constant in the up-down line, the hand's height slightly decreases. The hand movements are simply opening and closing the grippers. These arm and hand movements are prepared to be the content of output motor commands generated by the MP of grip.

5.3 The simulation of Herbert's arm controller

Herbert's arm controller drives the robot to pick up a soda can, and bring it back to a home location [28]. We simulated Herbert's arm controller in a newly created SMS embedded in a LIDA-based software agent in Webots to execute the grip. This simulation implements the *online control* process and a Motor Plan (MP) for gripping in the SMS. Based on the high-level design introduced in Section 4.1, the simulation's detailed computational design is introduced as follows.

5.3.1 Computational design

First, 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 [8], although they differ in details [9, 19]. The suppress node acts in the same way as the subsumption architecture's suppress process: High-level input replaces the low-level one. Hardware wires are simulated as computational component wires to link between modules and suppress nodes.

The three components shown in Figure 7 illustrate how they look and their constituent parts. Each of them has a core routine and I/O methods. The module (M) core routine acts like an AFSM in the subsumption architecture: it switches among multiple preassigned states, depends on the current state and the input sensory data, and it sends out a motor command. The core routine of 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 wire (W) core routine simply conveys a data copy from input to output.

Figure 7 here

These components' core routines are computationally implemented as *LIDA-tasks* [27] in the simulated Herbert's arm controller. A LIDA-task encapsulates a small process, and has implemented multithreading support, so that the core routines are able to operate in parallel and execute independently.

Second, the design of the simulated Herbert's arm controller is shown in Figure 8, redrawn from original Herbert's subsumption diagrams [19][8]. Sensory data enter from the left; output commands are sent out on the right.

[8] In comparison with the original design 19. Connell JH. A colony architecture for an artificial creature. DTIC Document, 1989., the cradle level, the back module, and the edge module were removed in the simulation because either their function is substituted for by the Webots simulated environment, or they are irrelevant to the hand and arm actuators.

Modules, suppress nodes, and wires are structured into multiple levels (layers), bottom-up ordered by their priorities with the higher priority being above. The module name briefly indicates the associated behavior. 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.

Figure 8 here

A Grip Motor Plan (MP) has been created simulating the Herbert arm controller. This Grip MP is embedded into a newly created SMS. The SMS receives sensory data from LIDA's Sensory Memory into Grip MP's module components, and also passes the MP's output commands issued by modules or suppress nodes to the outside, typically LIDA's Environment module. In this way, an online control process has been modeled. At the present time, the motor commands inside the MP are fixedly bound by default values, rather than being specified at run time, as shown in Figure 3. In accordance with the SMS's biological inspiration, a specification process will be added to the MP in a later implementation, to be described in Section 5.4.

5.3.2 Experiments

Two of Herbert's grip experiments have been duplicated, investigating the controller's reliability and flexibility as shown in Figure 9 and Figure 10, respectively.

Firgure 9 here

First, the results shown in Figure 9 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*[9] in Webots at the run time. During each trial, the hand starts from point a, and then traverses points b, c, and d exploring for the object: first doing a small bounce 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.

Second, the same controller is used in different environments to verify its flexibility. Figure 10 (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

[9] 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.

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 [19]. After the grippers are above the pedestal surface, it executes the remaining portion of the grip action as in the previous case.

Figure 10 (b) shows a case there the object doesn't stand on the surface the hand first touches. The hand touches the top of the barrier 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 10 here

These simulations successfully replicate the execution of a grip action 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.

5.4 Biologically Inspired modification

The simulated Herbert arm controller has been modified based on the SMS concept as described in Section 5.3. 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 (the two cylinders in Figure 4) are created for the specification. In each of them, a concrete association is implemented for the grip, which transforms the object's width into a distance between the grippers, the *grip aperture*.

As represented in Figure 11, the simulated grip aperture is sampled at unit intervals in Webots virtual time during a grip execution, which has been described above as shown in Figure 9. Below, these simulated grip apertures are analyzed and compared with hypotheses and observations of human gripping.

Figure 11 here

First, the grip action is executed without the specification process as an experimental control. As shown in Figure 11 (a), 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 4) has been implemented by connecting the object's width, as sensed through the dorsal stream, to the grip aperture. Its transformation formula is expressed by Eq. (1):

$$\text{Grip aperture} = \text{Width} * \text{Magnification} \quad (1)$$

The variable Width is a numeric value representing a true width directly sensed from the environment through the dorsal stream. Magnification is used to set the grip aperture to be slightly greater than the object width, since experimentally, "the [human] finger grip opens more than required by the size of the object" [22, 29]; Magnification is set to a value of 1.2 in the simulation. As shown in Figure 11 (b), the grip aperture reaches the specified value of 0.03m before the value falls as the grippers close. Compared to the maximum grip aperture (MGA), the value specified here is much closer to the target object width. This simulated calibration 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)" [4], because currently it is the sensory data through the dorsal stream which affects the grip aperture's value during the specification process.

Why is the MGA reached at two different times in Figure (b)? The first MGA peak is in keeping with the observed human behavior that people open their fingers maximum when starting a grip [22, 32], which is modeled by setting a fixed MGA value to the grip aperture for a short while when the execution starts. The second MGA peak occurs because the grippers touch the surface; this behavior both tracks the object's width value, and adapts to an unpredicted collision.

Third, another association (the bottom cylinder in Figure 4) has been implemented by connecting the object width represented in the context component of a selected behavior to the grip aperture value. Instead of the data being sensed through the dorsal stream, the selected behavior's context may affect the relevant command values under several conditions [4]. We simulated two of these conditions: 1) Deleting the association that connects the object's width through the dorsal stream to the grip aperture, in effect rendering the skill unfamiliar to the agent, or 2) Terminating the relevant data received from the dorsal stream, so as to simulate a delay in the dorsal stream.

Under either of these conditions, this association's formula is expressed by Eq. (1) as well, but the variable Width has a different meaning here. Since the object width represented in a behavior's context is a semantic value,

such as “large” or “small”, which are not precise, its value is designed to be distributed in a range, so that the represented object width approximates its true value. As shown in Figure 11 (c), five executions of the same simulated grip produced a range of context-specified values rather than a precise value. We argue that these imprecise movements result from an association connecting the selected behavior’s context to a command value. 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 [4, 33, 34]. This evidence supports the conclusion we reached above that the dorsal stream plays a central role in specification process.

Fourth, an update process is implemented to specify the grip aperture during the execution. Its formula is expressed by Eq. (1), the same as the association which connects the object width sensed through the dorsal stream to the grip aperture. However, instead of a constant value, the *Magnification* here is set to be dynamically decreasing throughout the execution period. As shown in Figure 11 (d), the grip aperture value comes closer to the object width than the specified value mentioned previously in Figure 11 (b). It follows that the sensory 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.

5.5 Linking the SMS to LIDA

The grip Motor Plan Template, whose implementation is described in Section 5.4 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 in Section 4.4.

As discussed in Section 2 and shown in Figure 5, 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 the LIDA Environment module. These I/Os are implemented in the LIDA-based agent including the SMS as shown in Figure 12. Only the SMS related modules are represented. The other LIDA modules are abstractly represented by LIDA’s understanding and attention phases.

Figure 12 here

6. 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 reviews action execution process deployment in three well-known cognitive architectures, and compares them 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.

6.1 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 consists of two types of modules: memory modules and perceptual-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. Procedural memory, encoded in production rules, represents knowledge outside of their awareness that is expressed in their behaviors rules. 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. The action specifies some actions, all of which are to be taken when the production fires (ACT-R 6.0 Tutorial).

ACT-R's perceptual-motor modules provide an elementary cognition layer with which to couple the high-level cognition layer, including declarative memory and production memory, with the world [35].

In ACT-R 6.0, the motor module is designed for modeling a simulated hand to operate a virtual keyboard and mouse. 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)" [35]. The movement is generated through three phases: preparation, initiation (processor), and execution. The preparation is based on movement features, which includes the movement's style and parameters. By default, the first 50ms after the preparation is movement initiation. After that, the movement may be executed if the motor module is not already executing a movement [35]. 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.

In ACT-R, the 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. The executed movement sent out from the ACT-R's motor module is passed to the related device module in order to carry out the execution in the real world.

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.

Third, an interface between the cognitive architecture and the world is prepared in both cases: an environment module in LIDA and the device module in ACT-R. 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[10]. 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.

6.2 Soar

Soar is a cognitive architecture in pursuit of general intelligent agents [36]. "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" [37].

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 [37]. 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 just below [36, 37].

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

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

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* [37]. 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 [37]. 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 [37].

An operator contains preconditions and actions; its action differs to a production rule's action. The operator action is an output for the agent to its internal or external environment. However, actions of a production generally either create preferences for operator selection, or create/remove working memory elements [37].

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* [37]. 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 [37].

The above introduction of Soar reveals 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 operator are performed by production rules that match the current situation and the current operator structure [37].

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.

6.3 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 [38, 39]. 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 [39].

The ACS consists of two levels of representation: the top level for explicit and the bottom level for implicit knowledge [38]. 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 for the major operations that act on it. The bottom level is a direct mapping from perceptual information to actions, implemented with neural networks involving distributed representations. In contrast, the explicit knowledge is more accessible, manipulable, and has

conceptual meaning [38]. At the top level, a number of explicit action rules are stored, which are usually in the following form: *current-state-condition* → *action* [39]. 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 [39].

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" [39].

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 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[11], 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[12], whose representational units in the hidden layer are capable of accomplishing tasks, but are generally not individually meaningful [39].

[11] 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.

[12] 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" [28].

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.

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 [39]. 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.

Additionally, learning is not yet implemented in LIDA's action process, whereas 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).

7. 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, first 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 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.

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

This biologically inspired design, a computational verification by the comparison of model and human behaviors, together with the review and the comparison between our model and other existing models, supports the SMS as a qualitatively reasonable cognitive model for action execution. A learning process will be addressed in future work on the SMS.

References

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

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

3. Goodale MA, Milner AD. Separate visual pathways for perception and action. Trends in Neurosciences. 1992;15(1):20-5.

4. Milner D, Goodale MA. Two visual systems re-viewed. Neuropsychologia. 2008;46(3):774-85.

5. Goodale MA. Separate visual systems for perception and action: a framework for understanding cortical visual impairment. Developmental Medicine & Child Neurology. 2013;55(s4):9-12.

6. Goodale MA. How (and why) the visual control of action differs from visual perception. Proceedings of the Royal Society B: Biological Sciences. 2014;281(1785):20140337.

7. Milner A. Is visual processing in the dorsal stream accessible to consciousness? Proceedings of the Royal Society B: Biological Sciences. 2012;279(1737):2289-98.

8. Brooks RA. A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation. 1986;2(1):14-23.

9. Brooks RA. How to build complete creatures rather than isolated cognitive simulators. Architectures for Intelligence: The Twenty-second Carnegie Mellon Symposium on Cognition. 1991:225-39.

10. Maes P, Brooks RA, editors. Learning to Coordinate Behaviors. AAAI; 1990. pp. 796-802.

11. Mahadevan S, Connell J. Automatic programming of behavior-based robots using reinforcement learning. Artificial intelligence. 1992;55(2):311-65.

12. Wolpert DM, Diedrichsen J, Flanagan JR. Principles of sensorimotor learning. Nature Reviews Neuroscience. 2011;12(12):739-51.

13. Weng JJ, editor The developmental approach to intelligent robots. AAAI Spring Symposium Series, Intergrating Robotic Research: Taking Next Leap; 1998; Stanford, CA.

14. Weng J. Developmental robotics: Theory and experiments. International Journal of Humanoid Robotics. 2004;1(02):199-236.

15. Franklin S, Strain S, Snaider J, McCall R, Faghihi U. Global workspace theory, its LIDA model and the underlying neuroscience. Biologically Inspired Cognitive Architectures. 2012;1:32-43.

16. Baars BJ. A cognitive theory of consciousness. New York: Cambridge University Press; 1988.

17. Baars BJ. The conscious access hypothesis: origins and recent evidence. Trends in Cognitive Sciences 2002;6(1):47-52.

18. Baars BJ, Gage NM. Fundamentals of Cognitive Neuroscience: A Beginner's Guide: Academic Press; 2013.

19. Connell JH. A colony architecture for an artificial creature. DTIC Document, 1989.

20. Brooks RA. Elephants don't play chess. Robotics and autonomous systems. 1990;6(1):3-15.

21. Searle JR. Intentionality: An essay in the philosophy of mind: Cambridge University Press; 1983.

22. Jeannerod M. Motor cognition: What actions tell the self. Oxford, UK: Oxford University Press; 2006.

23. De Vega M, Glenberg AM, Graesser AC. Symbols and embodiment: Debates on meaning and cognition: Oxford University Press, USA; 2008.

24. Simon HA. The sciences of the artificial. Cambridge, MA. 1969.

25. Cutsuridis V, Taylor JG. A cognitive control architecture for the perception–action cycle in robots and agents. Cognitive Computation. 2013;5(3):383-95.

26. Grafton ST. The cognitive neuroscience of prehension: recent developments. Experimental brain research. 2010;204(4):475-91.

27. Snaider J, McCall R, Franklin S. The LIDA framework as a general tool for AGI. Artificial General Intelligence. Berlin Heidelberg: Springer 2011. pp. 133-42.

28. Connell JH. A behavior-based arm controller. Robotics and Automation, IEEE Transactions on. 1989;5(6):784-91.

29. Jeannerod M. Intersegmental coordination during reaching at natural visual objects. Attention and performance IX. 1981;9:153-68.

30. James TW, Culham J, Humphrey GK, Milner AD, Goodale MA. Ventral occipital lesions impair object recognition but not object-directed grasping: an fMRI study. Brain. 2003;126(11):2463-75.

31. Milner D, Perrett D, Johnston R, Benson P, Jordan T, Heeley D, et al. Perception and action in 'visual form agnosia'. Brain. 1991;114(1):405-28.

32. Farnè A, Pavani F, Meneghello F, Làdavas E. Left tactile extinction following visual stimulation of a rubber hand. Brain. 2000;123(11):2350-60.

33. Jakobson L, Archibald Y, Carey D, Goodale MA. A kinematic analysis of reaching and grasping movements in a patient recovering from optic ataxia. Neuropsychologia. 1991;29(8):803-9.

34. Jeannerod M, Decety J, Michel F. Impairment of grasping movements following a bilateral posterior parietal lesion. Neuropsychologia. 1994;32(4):369-80.

35. Byrne MD, Anderson JR. Serial modules in parallel: The psychological refractory period and perfect time-sharing. Psychological Review. 2001;108(4):847-69.

36. Laird JE, editor Extending the Soar cognitive architecture. Artificial General Intelligence; 2008; Memphis TN, USA. pp. 224-35.

37. Laird JE, Congdon CB, Coulter KJ, Derbinsky N, Xu J. The Soar User's Manual 9.4.0. Accessed Dec 31, 2014. Available from: http://web.eecs.umich.edu/~soar/downloads/Documentation/SoarManual.pdf.

38. Sun R. The CLARION cognitive architecture: Extending cognitive modeling to social simulation. In: Sun R, editor. Cognition and multi-agent interaction. New York: Cambridge University Press; 2006. pp. 79-99.

39. Sun R. A tutorial on CLARION 5.0. Accessed Dec 31. 2014. Available from: http://www.cogsci.rpi.edu/~rsun/sun.tutorial.pdf.