FP6-004381-MACS

MACS

Multi-sensory Autonomous Cognitive Systems Interacting with Dynamic Environments for Perceiving and Using Affordances

Instrument: Specifically Targeted Research Project (STReP)

Thematic Priority: 2.3.2.4 Cognitive Systems

D2.2.1 Evaluation of Existing Control Architectures for using Affordances

Due date of deliverable: March 31, 2005
Actual submission date v1: May 12, 2005
Actual submission date v2: Nov 13, 2006

Start date of project: September 1, 2004 Duration: 36 months

Middle-East Technical University (METU-KOVAN)

Revision: Version 2

Project co-funded by the European Commission within the Sixth Framework Programme (2002–2006)		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

EU Project MACS

Deliverable D2.2.1

Evaluation of Existing Control Architectures for using Affordances

Erol Şahin, Emre Uğur, Maya Çakmak

Number: **MACS/2/2.1**
WP: *2.2*
Status: *Version 2*
Created at: *April 15, 2005*
Revised at: *July 19, 2006*
Internal rev: *v2 – November 13, 2006*

FhG/AIS	Fraunhofer Institut für Intelligente Analyse- und Informationssysteme, Sankt Augustin, D
JR_DIB	Joanneum Research Graz, A
LiU-IDA	Linköpings Universitet, Linköping, S
METU-KOVAN	Middle East Technical University, Ankara, T
OFAI	Österreichische Studiengesellschaft für Kybernetik, Vienna, A

This research was partly funded by the European Commission's 6th Framework Programme IST Project MACS under contract/grant number FP6-004381. The Commission's support is gratefully acknowledged.

Corresponding author's address:

Dr. Erol Şahin
Middle East Technical University
Dept. of Computer Engineering
Inonu Bulvari
TR-06531 Ankara, Turkey

Fraunhofer Institut für Intelligente Analyse- und Informationssysteme
Schloss Birlinghoven
D-53754 Sankt Augustin
Germany

Tel.: +49 (0) 2241 14-2683
(Co-ordinator)

Contact:
Dr.-Ing. Erich Rome

Joanneum Research
Institute of Digital Image Processing
Computational Perception (CAPE)
Steyrergasse 9
A-8010 Graz
Austria

Tel.: +43 (0) 316 876-1769

Contact:
Dr. Lucas Paletta

Linköpings Universitet
Dept. of Computer and Info. Science
Linköping 581 83
Sweden

Tel.: +46 13 24 26 28

Contact:
Prof. Dr. Patrick Doherty

Middle East Technical University
Dept. of Computer Engineering
Inonu Bulvari
TR-06531 Ankara
Turkey

Tel.: +90 312 210 5539

Contact:
Asst. Prof. Dr. Erol Şahin

Österreichische Studiengesellschaft
für Kybernetik (ÖSGK)
Freyung 6
A-1010 Vienna
Austria

Tel.: +43 1 5336112 0

Contact:
Prof. Dr. Georg Dorffner

Contents

1 Introduction

This deliverable reports the results of the review work done within Task 2.2. It uses the previous deliverable in this task, namely D2.1.1 (Identification of architectural requirements of an affordance-based control), as a "wish list" and evaluates the major contemporary robot control architectures with respect to this list.

This version of the deliverable, extends and details the review that was submitted during the first year of the project. Specifically, we first provide a detailed review of existing robot control architectures in the literature. In this review, we describe the fundamental concepts and approaches used in these architectures while providing cross-cutting comparisons among them. These reviews, which may read rather generic at a first glance, set the stage for the proceeding discussion on their appropriateness to the development of an affordance-based robot control architecture.

2 Overview of Existing Robot Control Architectures

Robot control architectures emerged from the need to organize the software that makes up what one might call the *intelligence* of a robot. As with most problems, robot control architectures evolved within two contradicting paradigms, ending up with better solutions somewhere in between. Modern control architectures combine the aspects of *hierarchical* and *reactive* architectures, for which they are called *hybrid* or *deliberative/reactive* architectures.

In this section we chronologically overview the robot control paradigms, going into more details on hybrid architectures and we review the major contemporary robot control architectures reported in the literature.

2.1 Hierarchical Architectures

The hierarchical architectures, often dubbed as sense-plan-act (SPA) architectures, emerged during the 70's. They are considered to be vertically separated into three functional units. The sense unit is required to fuse all sensor data into a single, rather complete world model. Using this model the plan unit makes the high-level planning for a desired goal. The act unit then takes these plans and generates the necessary motor commands.

In hierarchical control, execution occurs without any feedback from the sensors. There is always a risk that the world model on which the plan is based, is not up-to-date during execution. Therefore the SPA sequence is to be repeated in cycles, in order to update the world model, and the period of the cycle should be as small as possible. The problem about this suggestion is that world modeling and planning are time-consuming and complex task and even with today's processors it is practically impossible to reduce the period of this cycle to a level at which it would be possible to cope with the dynamism of the real world.

The hierarchical architectures suffered many problems during operation in real world conditions. The long computational cycle of the architecture prevented the robot to be responsive to changes in the environment and it lacked robustness in execution. It was shown that the dynamic world that the robot needs to survive can never be fully modeled. The execution would often result in an unexpected way, characterized by the "running researcher syndrome". These were the reasons for constructing artificial, structured environments for the operation of robots with such architectures.

Researchers believed that problems faced in the hierarchical paradigm could be overcome with better sensors, faster processors and accurate actuators. Planning was believed to be *the* essential part of the architecture, and most of the research was focused on planning and world modeling while no significant attention was given to execution. That's why they mostly contributed different planning techniques rather than different architectures and "classical planning" became a field of great interest. STRIPS [Fikes and Nilsson, 1971], the first major planning system, was actually designed as the planning component of the architecture of the first AI robot Shakey.

In classical planning, problems are described in terms of states, actions and goals and environments are assumed to be fully observable, deterministic, finite, static (changes are caused only by the agent) and discrete (in time, actions, objects and effects)[Russell and Norvig, 2003]. In a classical planning domain, the representation of a state is a conjunction of positive literals, which might be a proposition or a first-order predicate, either grounded or free variable. The state representation has the *closed world assumption* which means that any condition not mentioned in a state are assumed false. A goal is represented as a partially specified state. The actions are specified in terms of preconditions that must hold before it can be executed (precondition list) and the effect of executing it (add-delete lists). This is called the state-action model as the application of an action transforms one state into a subsequent one.

The earliest approach to planning was to use state-space search. One of the first planners, STRIPS, used state-space search with means-end analysis. Planners that followed, used different approaches, such as progression or regression (forward-backward search), different search algorithms, such as A* or hill-climbing, and they proposed different heuristics. In order to improve their heuristics, researchers used methods like the sub-goal independence assumption and the relaxed-problem approach.

The early planners generally worked on totally ordered plans. They achieved problem decomposition by solving each subgoal separately and ordering the computed sub-plans. This approach called linear planning was soon discovered to be incomplete [Sacerdoti, 1975]. For completeness, a planner must allow interleaving of the sub-plans. This led to the idea of partial-order planners, which are based on detection of conflicts between sub-plans and protection of achieved conditions from inference [Sussman, 1975]. This approach is also referred to as plan-space search planning. Many partial-order planners using the developed algorithms were implemented (NOAH,NONLIN). Planning graphs were also introduced to the field as a tool for useful heuristics for state-space and plan-space planners. They were as well used directly for planning in the GRAPHPLAN algorithm. Also, in order to benefit from the methods for solving the satisfiability problem, the idea of converting the planning problem to a satisfiability problem was proposed [Kautz and Selman, 1992]. SATPLAN was the first algorithm to apply this idea.

We chose not to review the hierarchical architectures in detail since they are outdated, and all the relevant architectural issues for our discussion already exist in hybrid architectures. But two of the commonly referred examples are Nested Hierarchical Controller (NHC) and Real-time Control System (RCS).

2.2 Reactive Architectures

In the beginning of 80's researchers started to realize the problems about the hierarchical paradigm in dealing with the problems caused by the *unstructured, dynamic* nature of the real world. Reactive architectures were proposed as a reaction to the slow and poor performing hierarchical architectures.

Arbib, was among the first ones starting to observe animals to gain an insight into how robots could be improved to perform better in the real world. He proposed to make use of schema theory in robotics and he represented both animal and robot behavior with schemas [Arbib, 1981]. Braitenberg suggested that understanding a behavior by observation and inducing the underlying controller, is much more difficult than trying to create a controller that will produce that behavior. This suggestion, about the bottom-up and top-down approaches, is also known as "law of uphill analysis and downhill invention" [Braitenberg, 1984]. In his book "Vehicles: Experiments in Synthetic Psychology", he showed how robots (vehicles) with very simple controllers displayed behavior which we would conceive as intelligent behavior.

Reactive architectures, inspired from simple animals, utilized the concept of behaviors which coupled the sensors directly with actuators. The term "behavior" was initially borrowed from biology and psychology, but it ended up being the root of a brand new field, *behavior-based robotics*. Yet, there seems to be no coherence on the meaning of "behavior". A broad definition of a behavior is "the way in which something reacts to the environment". Roboticists seem to exclude the

"environment" part of this definition. For instance, Murphy defines a behavior as a mapping from sensory inputs to a pattern of motor actions which are then used to achieve a task [Murphy, 2000]. In order to avoid the confusion Gat uses the following convention. A "Behavior" (capitalized) is considered as the code that implements a transfer function mapping the input to an output and a "behavior" is the set of physical activities obtained by that transfer function when running in an environment. Consequently, one may say that "a Behavior is a piece of code that produces a behavior when it is running" [Gat, 1998]. Similarly Bonasso et al. prefer to use the term *situated skills* [Slack, 1992] to refer to a configuration of a robot's control system that if placed in the environment will achieve or maintain a certain state.

Reactive architectures remove the planning and global world models of hierarchical architectures. Any symbolic representation is avoided and input from relevant sensors are directly used to modulate a behavior. A number of such behaviors interacting with the environment results in task-directed emergent behaviors, which are sometimes unexpected. Very responsive real-time robotic control systems were constructed with this approach. The basic idea is the same in all reactive architectures, but researchers came up with different interesting methods to apply the principles of the reactive paradigm. Reactive architectures can be distinguished according to three properties [Arkin, 1998]: coordination strategies (competitive vs. cooperative), granularity of behaviors and encoding of the response (discrete vs. continuous). Next we will review three representative architectures.

Subsumption Architecture [Brooks, 1986]

The subsumption architecture, proposed by Brooks is considered one of the most seminal works in robot control architecture research. It is designed to work in real world conditions where noisy and inconsistent sensor readings are usual and time-critical response is required. It has been demonstrated on a number of different robots.

The basic idea in subsumption is to decompose control into layers, each composed of "task achieving behaviors". Each behavior defines a mapping from sensory inputs to the actuators. Layering, can be considered as horizontal decomposition of the complete task where each layer is supposed to achieve a certain *level of competence*. A level of competence is "an informal specification of a desired class of behaviors for a robot over all environments it will encounter". Higher behavioral layers correspond to increasing level of competence. These layers are incrementally implemented and debugged in the real world.

Subsumption architectures can be described with the help of blocks corresponding to modules, each defined by a finite state machine augmented with some instance variables, and a number of inputs, outputs, inhibitors and suppressors. Each module has a number of input and output lines connected with wires to other modules for communication. A wire can *inhibit* the output or *suppress*, i.e. *replace*, the input of a module. Inhibition and suppression from more complex behaviors enable incremental construction of the system.

Behaviors in the subsumption architecture are sensor-driven, that is, an action produced by a behavior is a consequence of a stimulus received from the sensors. No symbolic representation and high-level processing is done. Brooks proposed that complex behaviors "need not necessarily be a product of an extremely complex control system", but they arise from the interactions of behaviors that execute in parallel.

Robot Schema Architecture [Lyons and Arbib, 1989]

The Robot Schema (RS) architecture, derived from the characteristics of robot programming domain, is proposed as a computational model. It has two major properties: concurrency at the behavioral level, and a tight sensor-motor coupling. In this architecture, schemas which are generic specifications of computational modules form the basic building blocks of the architecture. Complex behaviors are obtained by connecting the input and output "ports" of the schemas. Tasks

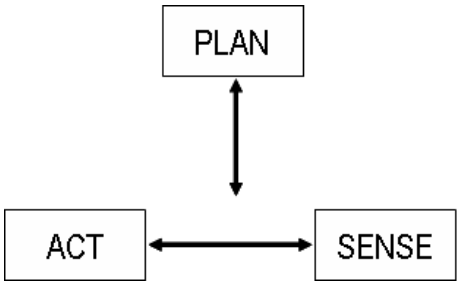

Figure 1: Organization of units in the hybrid paradigm

are accomplished through the interaction of a number of concurrent computing agents, which are responsible for perception, motor actuation or information processing.

Dual Dynamics [Jaeger and Christaller, 1997]

The Dual Dynamics (DD) architecture implements the control system as a continuous dynamical system. In DD, each behavior is implemented by an ordinary differential equation. As a consequence, each behavior has a target dynamics which is determined by its equation. This dynamics, however, can be modulated by behaviors from higher levels, called activation dynamics. This modulation is done through variables whose values are set by the higher-level behaviors. Note that, unlike the subsumption architecture, in DD higher level behaviors can not set directly any variables, but only change the activation of lower levels. The inspiration of this architecture came from the observation of animals whose behaviors are modulated by their moods.

2.3 Hybrid Architectures

The reactive paradigm was very successful in constructing robots that could survive in real world conditions. However, the absence of a planning module made it impossible to deal with complex tasks, and limited the application of the reactive architecture to simple domains. The limitations of the reactive architectures led the way to the emergence of hybrid architectures in the beginning of 90's. These architectures can be seen to re-introduce planning component back into the control architecture. Although reactive components were utilized at the lower layer of these architectures not to degrade the responsiveness of the robot, a deliberative layer was added to make them feasible in complex domains.

The organization of sense, plan and act units in hybrid architectures is often described with the rather ambiguous relationship given in figure 1 (from [Murphy, 2000]). The tight coupling between sense and act units is directly inherited from reactive architectures, however the role of planning is not clear. First of all, it should be noted that the plan unit is not limited to planning, it also performs all deliberation and world modeling. Secondly, sensing provides information for world modeling as well as the reactive layer. Symbols used at the upper level should be mapped to the sensory input, a process known as "perceptual anchoring" or "symbol grounding". This requires some of the sensors to be shared by both layers and some to be dedicated to either. From the figure it appears that hybrid architectures have two main layers, usually labeled as reactive and deliberative layers. With a careful examination of hybrid architectures in the literature one would see that some architectures advocate the necessity of a third middle layer while others are not necessarily layered.

The hybrid paradigm can be viewed as a step back from the biologically inspired reactive paradigm to the unnatural hierarchical paradigm, however there are psychological basis that support hybridization. Norman and Shallice [Norman and Shallice, 1986] distinguish *willed* and *automatic* control of behavior. Neisser proposes a two-layered perceptual system [Neisser, 1994] in

which the lower layer performs direct perception of affordances while the higher level, working in parallel, is responsible for perception related to cognitive abilities such as problem solving. Similarly, J. Norman [Norman, 2001] suggests a visual perception system with two interacting levels, which are tied to dorsal and ventral portions of the brain. Dorsal system is involved in direct perception to modulate actions, while ventral system is concerned with high level perceptual tasks, like recognition and identification.

There are a number of hybrid architectures that have been proposed and successfully implemented since the early 90's. Yet there's no agreed categorization of hybrid architectures. Different approaches are usually referred to by the representative architecture itself, rather than a class of architectures. There are some attempts to abstract hybrid architectures into categories however they often lack to cover all architectures in the literature and do not go further than "loose" or "broad" categorization. For example, Murphy categorizes the hybrid architectures according to their styles into three broad categories: managerial, state hierarchies and model-oriented [Murphy, 2000]. *Managerial* styles divide the deliberative layer into sub-parts responsible for different control or managerial tasks. *State hierarchies* style makes the task distribution among deliberative and reactive layers according to the state of the robot. Present state is associated with reactive behaviors which are viewed as instantaneous and having no memory while past and future states are associated with the two kinds of deliberative functions, execution of sequence of actions and planning for future actions. *Model-oriented* styles focus on the world model, possibly used by behaviors as well as deliberation.

Murphy also presents the bottom-up and top-down approaches in hybridization [Murphy, 2000], with a sense that a hybrid architecture roots from either a hierarchical or reactive architecture. Hybrid architectures which integrate deliberation on top of a working reactive system are considered to have a bottom-up approach, whereas those which stick with global world models and classical planning techniques and which usually stem from the traditional AI community are considered to have a top-down approach. Similarly, in [Bonasso et al., 1997] architectures are divided into two broad categories as the ones with the outset to control physically embedded agents and those initially designed to study general intelligence and later adapted to control physical agents. Another similar opinion comes from Gat who states that hybrid architectures in the literature either integrate the hierarchical and reactive approaches or "start with one approach and try to push its capabilities towards that of the other" [Gat, 1998].

Arkin [Arkin, 1998], on the other hand, divides hybrid architectures into four principle categories depending on how the deliberative and reactive components are interfaced. He focuses on how plans are used by the reactive layer. The four types of interfaces are *selection*, *advising*, *adaption* and *postponing*. In selection, planning is viewed as *configuration*, that is the composition of behaviors and parametrization. In advising, planning is viewed as *advice giving*, in the sense that planning suggests the necessary changes but the reactive does not necessarily apply them depending on the world conditions. In adaptation, the reactive component is *adapted* by the planning component as to cope with world conditions and achieve a given task. Lastly, in postponing, planning is considered as a *least commitment process*, it is postponed to as late as possible after determination of the goal.

Some researchers attempt to compare their architectures with others and express the distinctive attributes. These attributes also provide a base for categorizing hybrid architectures, such as homogeneous vs. heterogeneous, layered vs. non-layered or asynchronous vs. synchronous. Still, there is no common ground for evaluating hybrid architectures. We will try review best known architectures and place them in different categorizations stated above. We will consider attributes associated with different architectures as much as possible in the evaluation and discuss their relevance to an affordance based control architecture.

AuRA [Arkin and MacKenzie, 1994; Arkin and Balch, 1997]

The Autonomous Robot Architecture (AuRA) was developed by Arkin in the mid 80's. It is considered as the earliest hybrid architecture, having both a controller based on schema theory

and a deliberative planner based on classical planning. The deliberative system is divided into three sub-units, a mission planner, spatial reasoner (navigator) and a plan sequencer (pilot), making it a managerial style architecture according to Murphy's categorization.

The *mission planner* acts as an interface with the human, who will determine the goal. The *spatial reasoner*, uses the map situated in the long term memory, to compute the path of the robot, as a sequence of subtasks. The map can either be constructed by the robot or supplied externally. The *plan sequencer* then converts the subtasks determined by the spatial reasoner into a set of behaviors. The list of necessary behaviors is sent to the *schema manager* who activates them. Based on the output of *perceptual* schemas of each active behavior, the corresponding *motor* schema generates an output vector, using the potential field method. The vector sum of all vectors is executed by the low-level control system.

The *homeostatic control* system can effect the performance of the motor response by changing the strength of behaviors or other internal parameters according to internal sensors (e.g. fuel level, temperature). This system is inspired from the mechanism of internal hormones in animals, which subconsciously modulate their behaviors.

Most of the important aspects of AuRA, are motivated by studies ranging from biology to psychology. The reactive component based on schema theory and the homeostatic control system can be examples for biological basis. It is also influenced from Gibson's theory of affordances for using action-oriented perception in the reactive component, in the sense that each motor schema is associated with a perceptual schema which extracts the sensory input relevant for the particular behavior.

ATLANTIS [Gat, 1992]

ATLANTIS is three-layer architecture consisting of a controller, a sequencer and a deliberator. Unlike other architectures that use the state-action model as in classical planning, ATLANTIS is based on a *continuous action model*. This model employs operators which initiate or terminate a process that will cause a change in the state, instead of the operators that transform the state to a new one. The processes are called *activities* and the executed operators are called *decisions*. The name ATLANTIS is an acronym for "A Three-Layer Architecture for Navigation Through Intricate Situations".

The *controller* corresponds to the reactive layer and it is responsible for controlling activities which are mostly reactive sensorimotor processes. The architecture provides a framework for the design of this layer by imposing a programming language, ALFA (A Language For Action), specially designed for structuring the behaviors with modules and communication channels. Module definitions include both state-machine computational model and the interconnections.

The *sequencer* controls both sequences of primitive activities and deliberative computations. It strongly relies on the notion of *cognizant failures* [Firby, 1989], failures that can be detected by the system. As a principle, instead of trying to design algorithms that never fail they try to design "algorithms that never fail to detect failure". The sequencer is responsible for making the decisions for the required task in the controller and monitoring the progress of the activity between the decisions. It can also send parameters to the controller. The list of tasks that the system must perform are organized in a *task queue*, based on Firby's reaction-action packages (RAPs) [Firby, 1989]. A task is a list of methods and a description of the circumstances in which each method is applicable. A method can either be a primitive to be transferred to the controller layer or a list of subtasks to be installed onto the task queue. The execution of one RAP proceeds by successively expanding tasks on the queue until they finish or a failure occurs. In the case of a failure alternative methods are tried.

The RAP system was actually a complete control architecture developed by Firby for his PhD thesis. We did not review this system separately, but we may mention Gat's comparison of the two systems. The main difference of ATLANTIS is that it uses a continuous action model. The original RAP system therefore had to be modified to replace actions with activities, bringing some issues to be dealt with. For example, the resources used by each activity must be specified in order

to avoid conflicts.

The *deliberator* is responsible for time-consuming task like planning and world modeling. Although it appears to be the highest level of the architecture, it is controlled by the sequencer, which initiates the computations of the deliberator. The results of the computation are passed to the sequencer for execution.

The continuous action model is a concern for the planner as well. ATLANTIS uses the theory of plan-as-communication by Agre and Chapman [Agre and Chapman, 1988]. In this approach the plan constructed by the planner is not directly executed but rather used as an advice. Hence the representation of the actions inside the planner is not important as long as the output can be used by the sequencer.

ATLANTIS has the properties of a state hierarchy style architecture. Gat is indeed the one who used "the role of internal state" [Gat, 1998] to justify how hybrid architectures developed independently turned out to have three layers. He claims that three-layer architectures organize their algorithms according to whether they contain *no state*, contain state reflecting *past memories* or contain state reflecting *future predictions*. Stateless algorithms run a the controller layer, while algorithms with memory about the past run at the sequencer and the ones that make future prediction run at the deliberator.

SSS [Connell, 1992]

SSS (servo, subsumption, symbolic) architecture, is layered into three components according to the quantization of time and space of the tasks they perform. The architecture tries to combine the capabilities of three systems; a linear servo-like system, a reactive rule-like system and a discrete-time symbolic system, into a single system. The three have already been exclusively employed in special-purpose systems, but the challenge stands in their interaction. Servo systems operate in continuous time and space, in the sense that they constantly monitor the state of the world and representation of states is in terms of scalars. Subsumption systems, similarly work in continuous time however state representation is discretized into a number of categories depending on the domain. In symbolic systems, continuous time is replaced by discrete events. The three layers of the SSS architecture respectively apply the principles of these technologies.

The subsumption layer guides the control loops in the servo layer by adjusting its setpoints (e.g. setting the desired speed for a PID controller). The interface between sensors and subsumption layer is based on the idea of *matched filters* [Wehner, 1987], which suggests that certain sensor states are equivalent if they call for the same motor response, just as the "entity equivalence" in [Şahin et al., 2006]. There are typically some "key features" that discriminate the "relevant situations" for a certain motor schema. These matched filter recognizers strongly resemble the perceptual schemas that perform action-oriented perception. The symbolic layer controls the subsumbtion layer by activating and deactivating behaviors and parameterizing certain modules. The upward communication between subsumption and symbolic layers occurs through event recognizers. In order to decouple the symbolic system from real-time operations they added a structure called the "contingency table" that contains what actions to take when certain events occur.

Although, SSS is three-layered as ATLANTIS, there's not an exact correspondence of the layers. The lowest layers are very similar and they are directed by the middle layers in both architectures, with different methods though. ATLANTIS uses RAPs whereas SSS uses subsumption. ATLANTIS places more emphasis on sequencing by dedicating the middle layer (sequencer). In SSS, sequencing is handled at the symbolic layer who has direct control on behaviors in the subsumption layer. ATLANTIS is a good example of state hierarchies style hybrid architecture described earlier, while SSS is not. The distribution of responsibilities among the layers of SSS, is based on discretization of time and space rather than state information.

More importantly, the symbolic layer of SSS is actually in the control loop, whereas the highest layer in ATLANTIS (deliberator) merely provides advice to the sequencer [Gat, 1992]. This is why the contingency table is essential in SSS.

SFX [Murphy and Arkin, 1992]

Sensor Fusion Effects (SFX) architecture is a descendent of AuRA, extending it with special emphasis on sensor fusion and sensor failures. Inspired from the neuro-psychological model of sensing in animals, it applies branching in perception, allowing same sensory stream to be used by the reactive and deliberative layers simultaneously. The architecture sets the key elements of a sensor fusion architecture as the sensing plan, the uncertainty management, the feedback from the sensing process to individual sensors and the detection and handling of exceptions. The SFX architecture incorporates the previous developments in sensor fusion into a robot control architecture.

Sensor fusion in SFX strongly relies to action-oriented perception, asserting the term "action-oriented sensor fusion". The authors put this by saying that "robots should perceive only what they need to". Perception in SFX consists of independent perceptual processes each trying to achieve a *sensing objective* determined by the current motor behavior, as action-oriented perception suggests. The perceptual processes may compete or cooperate with each other in achieving their sensing objectives.

Sensing objectives are given in terms of *descriptions* of the percept that are necessary for the motor behavior. A description is a collection of *features*, which are $\langle attribute, value \rangle$ pairs. A percept is modeled in terms of features whose relationships are described by a perceptual context-free grammar. A prediction of the required percept can be built using the grammar specified by the motor behavior. According to the required description the perceptual process constructs a *sensing plan*, which specifies the sensors and algorithms to be used and other constraints like ordering or focus of attention. The sensing plan also determines the feedback rule and error bounds, and it guides the execution sequence of sensor fusion activities.

The execution sequence is mainly controlled by the *uncertainty management* module which takes the sensing plan and outputs the percept and its measure of certainty to the corresponding motor behavior. The exception handling module is responsible for adapting the sensing plan in case of exceptions, for example when the error bounds are exceeded. The execution sequence ends either by normal termination of the motor behavior or by an exception in the sensing plan.

The deliberative layer is divided into specialized modules, each being an independent software agent. The *mission planner* interacts with the human and other agents in the deliberative layer, to specify the constraints of the mission. Resource management is distributed to three managers (task, sensing and effector), which use AI techniques to allocate sensing and effector resources for the perceptual and motor schemas of a behavior. The sensing manager is also responsible for performance monitoring. It detects failures and inconsistencies, and it can propose alternative perceptual schemas or behaviors. The deliberative layer also contains a cartographer responsible for map making and path planning. The reactive layer is divided into two layers including strategic and tactical behaviors and the emergent behaviors are obtained by the *filtering method*.

3T [Bonasso et al., 1997]

The 3T (3-Tiered) architecture is another 3-layer architecture sharing various common aspects with ATLANTIS. It divides control into three tiers; a reactive tier, a sequencing tier and a deliberative tier. The *reactive tier* consists of a set of situated skills coordinated by a skill manager. Representation of skills and their interface to the sequencer are standardized so as to keep the architecture robot-independent. A skill is characterized by a transform function, which continually maps its inputs to an output. The input of a skill may come from the output of other skills as well as from sensors directly. The skill manager allows the skills to communicate with each other while providing a uniform interface between the sequencer and the skills. The *sequencing tier* is simply a RAP interpreter where a RAP can be considered as a description of how to accomplish a certain task. Each RAP is keyed to specific situations in which different set of skills are activated. The skill layer contains some special skills called *events*, which take inputs from other skills. An event skill notifies the sequencer when a certain state is detected. Execution monitoring also performed through these skills.

The interaction between the tiers occurs as follows. Given a set of goals the deliberative tier constructs partially-ordered plans, as lists of tasks to be performed. Each task is a set of sequenced actions (RAPs). Planning is performed by an adversarial planner [Elsaesser and MacMillan, 1991] which takes into account resources and time constraints. The planner selects the appropriate RAPs for a given task, by matching the task's *propositional effect clauses* with the *success clauses* of the RAPs in the library. A RAP is either a set of other RAPs or a set of skills. The sequencing tier interprets the selected RAP by decomposing it or by activating the corresponding skills in the reactive tier. Event skills are activated at the same time.

Making the three tiers of the architecture operate *concurrently* and *asynchronously* is considered to be the key to make planning useful without sacrificing responsiveness. 3T architecture tries keep its planner at the highest level of abstraction possible. The sequencing tier has an important role in raising this level. One operator in the planner may correspond to "a family of similar execution time actions". While the planner deals with a single operator, the RAPs system is responsible for selecting the appropriate execution time actions and trying alternatives in case of failures. Murphy's statement of the same idea is to "interleave planning and reaction".

Bonasso et al. point out an important property of the 3T architecture by making a comparison with another hybrid architecture developed by Noreils [Noreils and Chatila, 1995]. This architecture has three layers and uses a formalism similar to RAPs in the middle layer. The main difference is that in their formalism the outcome of executions are grouped as *success* and *failure* whereas RAPs considers *multiple outcomes*.

The architecture also presents a set of rules on how to distribute the tasks across the layers so that it can be applied to different domains that introduce different tasks. This suggests separating tasks according to four dimensions: time, bandwidth, task requirements and modifiability. *Time* dimension divides the tasks according to their speeds, and suggests implementing the fast ones as skills and abstracting slower ones. *Bandwidth* suggests that data flow between the layers should be as small as possible. *Task requirements* may be determinant depending on the mechanisms used in each layer. Finally, *modifiability* suggests putting tasks that require frequent modification in higher levels and the ones that are not modified once implemented into the skill layer.

DD&P [Hertzberg and Schnherr, 2001]

DD&P (Dual Dynamics and Planning) is a two-layer architecture developed to achieve concurrency both among the two levels of the architecture and among the behaviors within the lower layer. It extends the DD architecture into a hybrid architecture through the addition of a planner and a knowledge base. Behaviors in the DD component are independent and are designed to run concurrently. The formulation of behaviors is the same as in the reactive DD architecture. However, the activation and target dynamics of the behaviors are additionally influenced from the planner component, which is a classical propositional planner. This requires a definition of the interface between the two layers. In other words, it is necessary to specify the way that a plan being executed will influence the DD component and the way that the world model will be constructed from the DD and sensor information.

In DD&P planning is not directly incorporated as an independent term in differential equations. Instead, after ordered set of actions are created, related behaviors are activated or deactivated by *stimulation* or *inhibition*.

DD&P architecture is in a way similar to AuRA, which has concurrent reactive and deliberative layers and concurrent schema-based behaviors at the reactive level.

In DD&P, the DD layer is implemented according to the principles of the flip-tick architecture (FTA). FTA organizes software as independent program instances called *actors* and synchronized *ensembles* of actors. Instead of direct communication between actors, a *tagboard* system is preferred. This provides modularity in design and gives the flexibility to add and remove actors without having to think about the communication network. Each behavior in DD is implemented as an actor in FTA.

DD&P was conceptually developed in an earlier publication by Hertzberg et al. where they

address the problem of filling the gap between the information on the DD side and the information required at the symbolic side. Differing from other hybrid architectures, they suggest that this should not be achieved directly through sensors [Hertzberg et al., 1998]. They state the epistemological reason behind this assumption to be the inability of agents to perceive the world per se. In their framework, only actions of the robot are used to obtain world descriptions. Sensors contribute indirectly, due to the fact that actions are based on sensor readings. They propose to model the world from interaction experiences, more precisely from the activation history of behaviors. In their world models, concepts of objects are allowed only in terms of agent's interactions with them.

Saphira [Konolige et al., 1997]

Saphira is an architecture that integrates many capabilities using methods from different studies. The heart of the architecture is a world model called the *local perceptual space (LPS)*. Information about the environment in LPS has different levels of abstraction that are updated at different rates depending on the complexity of sensor data processing and interpretation required. Consequently activities at different levels of the reactive-deliberative distinction, refer to different parts in the representation. LPS includes a grid-based representation obtained with sensor fusion, representations obtained by interpretation of sensors according to prior world models (e.g. linear surface features) and representation of artifacts, which belong to one of pre-specified classes, in the environment. All activities in the architecture use LPS. This is the reason making Saphira a model-oriented style hybrid architecture.

The Saphira architecture gives emphasis to three concepts: coordination of behaviors, coherence of the world model and communication. Coordination is supported with layered abstraction of both perceptual and action routines. Coherence is attained by the use of LPS. The focus on communication is rather peculiar to Saphira, seeing it is often not a concern in other architectures, but Saphira takes a good first step by integrating natural language processing and other perceptual information about the other agents.

The lowest level control units of Saphira are the *basic behaviors* which use fuzzy logic techniques described by Saffiotti [Saffiotti et al., 1993]. Each behavior is characterized by a *desirability function* which gives a measure of how desirable a possible motor control is at a given state. The control command that is eventually executed, is obtained by defuzzification, a weighted sum in the case of Saphira. Certainly, this method works only with reasonable desirability functions, which do not suggest opposing controls at the same time.

More complex behaviors can be obtained through the combination of basic behaviors, by fuzzy operations on desirability functions. This is again problematic in the case of conflicting goals which would attenuate each other's effects in the defuzzification process. To deal with this problem, Saphira assigns rules about the context of applicability to each behavior, which are given as simple if-then rules. The resulting system resembles both the subsumption architecture in which behaviors subsume each other depending on the context and robot schema architecture in which a weighted sum of the outputs of each schema is executed by actuators. The method is called *context dependent blending* of behaviors.

Control of behaviors is coordinated by the procedural reasoning system (PRS), which is based on *activity schemas* that enclose the goals to be achieved. The intentions of the robot at a given instance is determined by the currently active schemas. Goal-directed behavior is obtained by instantiating the correct schemas for a given goal. There are different classes of operations supported in PRS: testing some conditions, waiting for conditions, executing commands or intending/unintending schemas. These operations are organized in *intention structures* which impose the goal. Reactivity of the architecture is supported with the waiting behavior, which suspends the current task in order to deal with dynamic changes detected in the environment. PRS supports both atomic actions and continuous processes. Outcome of actions is simultaneously monitored as to detect execution failures.

Goals are determined through communication. Speech and gestures are both used to give commands, which are either of the four: direct motion commands, sensor-based movements, tasks or

information. *Direct commands* refer to a set of behaviors to be executed for a certain duration or until an ending condition occurs. *Sensor-based movements* include attending and following. *Tasking* involves setting a goal and constructing the corresponding intention structure. This may include the construction of plans or conditional plans, mainly related to navigation. Finally *information* plays a role in world model construction by supplying facts in addition to the ones obtained from sensors. This is especially challenging in that a common representation of knowledge is required.

Artifacts in LPS have a central role in maintaining coherence between the internal representation of the robot and the real environment. Artifacts are obtained by the combination of a priory information and perceptual features and partly from other artifacts and goal information. Features are mostly related to surfaces. The a priory information consists of *object hypothesis* which map a set of features to certain objects or artifacts. *Anchoring* is the name given to the process of matching the internal artifacts with perceptual features and updating the information as the perceptual field or the environment changes.

TCA - Task Control Architecture [Simmons, 1994]

TCA has been developed as a framework for combining deliberative and reactive behaviors, rather than a fully specified architecture. The term "task control" refers to the coordination of perception, planning, and execution, given a set of goals. Task control problems include noticing that goals need attention, deciding which goals to attend to, constructing plans at different detail levels, monitoring progress, and dealing with exceptions.

Unlike architectures that try to adapt deliberation on top of a set of reactive behaviors, TCA starts with a deliberative component and incrementally layers some reactive behaviors around it. While the deliberative component deals with nominal situations, behaviors are supposed to handle exceptions. TCA has a central controller, common in all robots, and some robot specific modules. Central controller is the center for deliberation. Perception and actuation are handled by the robot-specific modules.

Modules in TCA are not directly connected to each other, instead they are all connected to the central controller which is responsible for transmitting messages from one module to another. This central routing method may seem to degrade the efficiency in communication, but it does't have a significant effect as the central controller focuses on routing and does not perform any interpretation. Besides, this system facilitates modular design of the architecture.

Messages passed among the modules may be of one of the six special types. *Inform* messages pass information from one module to another. *Query* messages are two way messages consisting of a request and a reply. *Goal* messages decompose tasks into subtask enabling planning and execution to be interleaved. *Command* messages consist of executable actions of the robot system. *Monitor* messages activate execution monitors. *Exception* messages handle exceptional situations.

The central controller uses *task trees* to hierarchically represent plans. This representation holds task/subtask relationships as well as temporal constraints. It includes three types of nodes and three types of arrows. *Goal* nodes are non-terminal and they can be decomposed into subtasks with *task decomposition* arrows. Subtasks are either other goals, which are nodes to be further decomposed, or leaves which may be a *command* or a *monitor*. Temporal constraints among the nodes are expressed with *sequential achievement* arrows. All goal nodes in a task tree are not to be expanded at once, their decomposition can be delayed with the help of *delay planning* arrows. Nodes followed by these arrows are decomposed only after the subtask at the node where the arrow originates is achieved. The central control applies constraints imposed by the task tree while transmitting messages from one module to another. The high level information on how to construct a task tree is maintained in some modules for later usage.

Reactivity in TCA is obtained by the use of *polling* and *interrupt-driven* monitors which detect unexpected changes in the environment. They are distinguished from the monitors in the task tree, which are referred to as *point monitors*, and they are concurrent with the execution of the task tree. Monitoring modules are responsible for informing the central control whenever the

a previously specified event is detected. Exception handling is hierarchical, in the sense that exceptions handlers are searched upwards in the task three and the plan can resume as soon as the exceptions are handled. If the problem cannot be fixed the task is ended.

3 Evaluation of existing architectures

In this section, we use the architectural requirements (grouped into Perception, Representation and Learning in D2.1.1) as our guide for the discussion and we review how and to what extend these can be fulfilled by the existing architectures.

3.1 Perception

The main perceptual requirement of the affordance-based control architecture was stated in D2.1.1 as: Perception of the world should be altered based on the goal of the robot, both to prevent the robot "drowning in affordances" and to activate only the perceptual processing schemas that are relevant for the goal for real-time response. This requirement, which can be summarized as the "modulation of perception by the goal", is an essential aspect of the affordance concept and therefore needs to be explicitly addressed by the architecture.

Some of the existing architectures address this requirement with the action-oriented perception approach in which perception is modulated by the active action or behavior. This is applicable both in reactive architectures and at the reactive layer of hybrid architectures. Some examples from the architectures reviewed in this document are:

- Robot schema architecture where the perceptual schemas are activated or deactivated by motors schemas
- AuRA which uses the robot schemas at the reactive layer
- SFX in which necessary sensors are activated by the sensing plan
- SSS which uses the principle of matched filters to interface sensors to the middle layer

Purely reactive architectures lack the concept of intentional and purposeful actions. They do not have a goal although one might say that they implicitly aim to survive. For instance, the robot schemas provide a simple generic structuring of functional modules (schemas), which allows one to implement the goal as an internal input. However none of the reactive architectures is taskable, that is they have to be reprogrammed in order to achieve a different task.

Hybrid architectures do possess the concept of a goal. In AuRA, execution of a plan constructed according to the given goal, consists of activating motor schemas determined by the plan and the relevant perceptual schemas. This is in a way deliberative modulation of perception. A more obvious example is the SFX architecture in which the symbolic world model depends on the current behavior, as a consequence of action-oriented sensor fusion. In SFX, the percept that is to be sensed is determined by the current motor behavior and it is acquired by the help of a sensing plan specific to that behavior.

However, some of the hybrid architectures use perception as a stand-alone process. The deliberative layers rely on a perception of the world generated by the perception module. The perception module usually takes the sensory inputs and generates a world description at a pre-specified level regardless of the tasks to be executed which can then be used as inputs for planning and execution.

3.2 Representation

As the reactive paradigm suggests, the agent does not need to internally represent the environment, or its affordances, in order to correctly act in it. This is similar to saying that reactive architecture using action-oriented perceptual filtering implicitly use the concept of affordances. On the other

hand, one would agree that deliberation is not possible without any kind of representation. In an affordance-based control architecture, which aims at performing deliberation based on affordances, an explicit representation of affordances is essential. Since affordances are neither the properties of the robot nor the object, they require a different representation at the deliberative layers.

In the MACS project, there is a common agreement about the formal representation of an affordance as a triplet of cues, effects and the behavior associated with that affordance. This representation will allow deliberative usage of affordances stored in a knowledge base. Knowledge representation used for planning in existing architectures is indeed very similar to our representation of affordances. Consider the very basic STRIPS representation of actions, they are stored together with their preconditions and effects. A more sophisticated example is the RAPs, which handle sequences of actions as well.

Representation of the cues and the effects is the main challenge in the affordance based architecture. Our review has shown that most of the architectures use propositional or predicate logic. This requires a mechanism to map the sensory input to symbolic literals such as the event skills in the 3T architecture. Predefined perceptual schemas are very helpful in acquiring the specific information from the environment however learning new schemas is only possible with a good representation of schemas. Similarly learning new affordances requires a formal representation of the cues and the effects. Sometimes representing cues for an affordance may be more difficult than specifying a value or a range for attributes detected by perceptual filters. There may be nonlinear or discrete characteristics.

The SFX architecture has a clever solution for representing complex relationships between features of a percept. A perceptual context-free-grammar determines what features are necessary and how each feature contributes to the representation of the percept. Still, learning such grammars is not straight forward. Providing a good base for learning should be the central concern of the representation.

3.3 Learning

Learning changes the internal structure of a control architecture adapting the robot to new conditions. Yet, learning has usually been side-stepped in existing control architectures. Most of the existing robot control architectures are designed for the robust execution of desired behaviors by the robot. The reactive architectures often aim for fast response times whereas the hybrid architectures try to achieve fault-tolerant execution strategies through detection of errors in execution. Despite the emphasis on execution and lack of structural facilities for learning, standard learning methods are learning is studied at all levels. Learning in reactive systems usually appear as simple parameter adaptation methods to improve the performance of reflexive behaviors. Among reactive architectures, neural networks provide well-studied learning mechanisms to do primitive low-level pattern learning. Such learning mechanisms could be of use in detecting and learning of coherent spatio-temporal features that would support a desired affordance. Learning in hybrid architectures mostly occur at the deliberative layers at the symbolic level. Generic machine learning methods, ranging from case-based reasoning to reinforcement learning are usually utilized at the deliberative layers. For instance in AuRA rule based methodology is used for on-line adaptation of motor behaviors, case-based reasoning methods to provide discontinuous switching of behaviors based upon the recognition of new situations. In general reinforcement learning methods are used for behavior selection. Our review on existing affordance learning approaches (D5.1.1) shows that learning is often implemented in ad-hoc control architectures that are specifically designed to test certain aspects of affordance learning. In this sense, our attempt to develop an affordance based control architecture which embeds learning mechanisms will be a novel contribution.

4 Conclusions

This report surveyed the existing control architectures and evaluated their fitness for the development of an affordance-based control architecture. Our evaluation clearly shows that the requirements put forward for an affordance-based control architecture are not satisfied by the existing architectures. Although we set out to develop a control architecture for using affordances in autonomous robotic systems, within this project we will concentrate only on the parts of the architecture that are specific for affordances, and will re-use concepts or components from existing architectures to the maximum extent.

References

[Agre and Chapman, 1988] Agre, P. E. and Chapman, D. (1988). What are plans for? Technical report, Massachusetts Institute of Technology.

[Arbib, 1981] Arbib, M. A. (1981). *Perceptual Structures and Distributed Motor Control. Handbook of Physiology - The Nervous System II.* The American Physiological Society and Oxford University Press.

[Arkin, 1998] Arkin, R. (1998). *Behavior Based Robotics*. MIT Press.

[Arkin and Balch, 1997] Arkin, R. C. and Balch, T. (1997). Aura: Principles and practice in review. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2):175–189.

[Arkin and MacKenzie, 1994] Arkin, R. C. and MacKenzie, D. (1994). Planning to behave: A hybrid deliberative/reactive robot control architecture for mobile manipulation. In *International Symposium on Robotics and Manufacturing*.

[Bonasso et al., 1997] Bonasso, R., Firby, R. J., Gat, E., Kortenkamp, D., Miller, D., and Slack, M. (1997). Experiences with an architecture for intelligent, reactive agents. *Journal of the Robotics Society of Japan*, 9(1):237–256.

[Braitenberg, 1984] Braitenberg, V. (1984). *Vehicles: Experiments in Synthetic Psychology*. MIT Press.

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

[Connell, 1992] Connell, J. (1992). Sss: A hybrid architecture applied to robot navigation. In *Proceedings of theIEEE International Conference on Robotics and Automation*, pages 2719–2724, Los Alamitos, California.

[Şahin et al., 2006] Şahin, E., Çakmak, M., Doğar, M. R., Uğur, E., Üçoluk, G., Breithaupt, R., and Rome, E. (2006). A new formalization of affordances. Technical Report METU-CENG-TR-2006-XX v1, Middle East Technical University, Department of Computer Engineering, Ankara, Turkey.

[Elsaesser and MacMillan, 1991] Elsaesser, C. and MacMillan, R. (1991). Representation and algorithms for multiagent adversarial planning. Technical Report MTR-91W000207, The MITRE Corporation.

[Fikes and Nilsson, 1971] Fikes, R. and Nilsson, N. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4):189–208.

[Firby, 1989] Firby, J. (1989). *Adaptive Execution in Complex Dynamic Worlds*. Technical report, yaleucsdrr #672,, Yale University.

[Gat, 1992] Gat, E. (1992). Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 809–815.

[Gat, 1998] Gat, E. (1998). On three-layer architectures. In D.Kortenkamp, Bonasso, R., and Murphy, R., editors, *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems Studies of Successful Robot System*, MIT Press, pages 195–210.

[Hertzberg et al., 1998] Hertzberg, J., Jaeger, H., Morignot, P., and Zimmer, U. (1998). A framework for plan execution in behavior-based robots. In *Proc. of the 1998 IEEE Int. Symp. on Intell. Control (ISIC-98)*, pages 8–13, Gaithersburg, MD.

[Hertzberg and Schnherr, 2001] Hertzberg, J. and Schnherr, F. (2001). Concurrency in the dd&p robot control architecture. In *Proceedings of The International NAISO Congress on Information Science Innovations*, pages 1079–1085.

[Jaeger and Christaller, 1997] Jaeger, H. and Christaller, T. (1997). Dual dynamics: Designing behavior systems for autonomous robots. In Fujimura, S. and Sugisaka, M., editors, *Proceedings International Symposium on Artificial Life and Robotics (AROB)*, pages 76–79, Beppu, Japan.

[Kautz and Selman, 1992] Kautz, H. and Selman, B. (1992). Planning as satisfiability. In *ECAI Ó2: Proceedings of the 10th European conference on Artificial intelligence*, 0-471-93608-1, pages 359–363, New York, NY, USA. John Wiley & Sons, Inc.

[Konolige et al., 1997] Konolige, K., Myers, K., Ruspini, E., and Saffiotti, A. (1997). The saphira architecture: A design for autonomy. *Journal of Experimental and Theoretical Artificial Intelligence*, 9:215–235.

[Lyons and Arbib, 1989] Lyons, D. and Arbib, M. (1989). A formal model of computation for sensory-based robotics. *IEEE Transactions on Robotics and Automation*, 5(3):280–293.

[Murphy, 2000] Murphy, R. R. (2000). *Introduction to AI Robotics*. MIT Press, Cambridge, MA, USA.

[Murphy and Arkin, 1992] Murphy, R. R. and Arkin, R. C. (1992). Sfx: An architecture for action-oriented sensor fusion. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1079–1086, Raleigh, NC.

[Neisser, 1994] Neisser, U. (1994). Multiple systems: a new approach to cognitive theory. *The European Journal of Cognitive Psychology*, 6:225–241.

[Noreils and Chatila, 1995] Noreils, F. R. and Chatila, R. G. (1995). Plan execution monitoring and control architecture for mobile robots. *IEEE Transactions on Robotics and Automation*, 11(2):255–266.

[Norman and Shallice, 1986] Norman, D. A. and Shallice, T. (1986). Attention to action: Willed and automatic control of behavior. In Davidson, R. J., Schwartz, G. E., and Shapiro, D., editors, *Consciousness and self-regulation: Advances in research and theory*, volume 4, pages 1–18.

[Norman, 2001] Norman, J. (2001). Ecological psychology and the two visual systems: Not to worry! *Ecological Psychology*, 13(2):135–145.

[Russell and Norvig, 2003] Russell, S. and Norvig, P. (2003). *Artificial intelligence: a modern approach*. 0-13-103805-2. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Sacerdoti, 1975] Sacerdoti, E. (1975). *A structure for plans and behavior*. PhD thesis, SRI International, Menlo Park, CA. AAI7605794.

[Saffiotti et al., 1993] Saffiotti, A., Ruspini, E. H., and Konolige, K. (1993). Blending reactivity and goal-directedness in a fuzzy controller. In *Proceedings of the IEEE International Conference on Fuzzy Systems*, pages 134–139, San Francisco, California. IEEE Press.

[Simmons, 1994] Simmons, R. G. (1994). Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, 10(1):34–43.

[Slack, 1992] Slack, M. G. (1992). Sequencing formally defined reactions for robotic activity: integrating raps and gapps. In *Proceedings of SPIE's workshop on Sensor Fusion*.

[Sussman, 1975] Sussman, G. (1975). *A Computer Model of Skill Acquisition*. Elsevier Science Inc., New York, NY, USA. ISBN: 044400159X.

[Wehner, 1987] Wehner, R. (1987). matched filters neural models of the external world. *Journal of Comparative Physiology A: Neuroethology, Sensory, Neural, and Behavioral Physiology*, 161(4):511 – 531.