Control Architectures for Autonomous Robots

Faisal Zubair Qureshi

12 August 2003

1 Introduction

Humans and other animals are nature's autonomous agents. They can operate in oftentimes hostile environments without supervision, performing purposeful actions that accomplish their goals. At each instant, an agent perceives its environment through its sensors, selects an appropriate motor action, and carries out the action through its effectors. Robots are quintessential autonomous agents of the artificial kind. Building robotic autonomous agents has proven to be a formidable challenge. Regardless of the reliability of the available sensors and effectors, the real world defies complete and accurate modeling, nor can a robot anticipate the effects of all its possible actions on the world. These are fundamental problems whose solution requires more than brute force compute power.

To address these problems, researchers have proposed various robot control architectures. The architectures fall into one of three categories: Deliberative, reactive, or hybrid (deliberative/reactive). Deliberative controllers maintain an internal world model and use it to plan a course of action. They typically cannot keep pace with the real world, because all but the most trivial planning takes significant time. By contrast, reactive controllers operate at the speed of the environment, exploiting the interaction of many real-time processes. Unfortunately, they cannot accomplish tasks that require any amount of planning. Humans are sophisticated autonomous agents that combine reactivity and deliberation. This observation has led robotics researchers to propose hybrid, deliberative/reactive control architectures. Hybrid architectures appear to hold the greatest promise for building autonomous robots and other autonomous agents with human-like abilities.

There exists considerable literature on autonomous agents. This document briefly

reviews the relevant background literature with an eye towards building autonomous robots. Section 2 begins by identifying requirements for an autonomous robot. Next, in Section 3, we trace the evolution of robot control architectures. In Section 4, we conclude the report with a summary and we identify some areas that require further investigation.

2 Autonomous Robots

What is an autonomous agent? This question is elusive. Many definitions of autonomous agents exist in the literature, and they can be summarized as follows.

> An autonomous agent inhabits a complex and dynamic world, and it pursues its agenda within this world. It is able to perceive its surroundings through its sensors. It is able to act through its actuators. It is also able to make rational choices and act upon them to fulfill its agenda.

This definition introduces several important notions. First, an autonomous agent is *situated*. It is a part of its world, and it can only be fully realized within its world. For example, robots are understood within the context of the real world. Second, an autonomous agent has an agenda; i.e., it has goals that it wants to accomplish. These goals can be implicit in the agent's design, the agent can set these goals by itself, or an external entity can ask the agent to achieve certain goals. Third, an agent interacts with its world through its sensors and actuators; i.e., an agent is *embodied*. It experiences and affects its surroundings. Fourth, an agent is capable of making rational choices that will further its agenda.

The above definition is loose and does not stand up to mathematical scrutiny; however, it serves our purpose. It works well for the autonomous agents in which we are interested—autonomous robots. An autonomous robot has its own agenda that it pursues on its own. It does so by perceiving its environment, making rational choices, and acting upon these choices. We identify the following requirements for an autonomous robot:

1. A robot should be able to remain functional in the real world for long periods without human assistance.
2. It should be able to handle the challenges posed by its environment.
3. It should be able to respond intelligently to unexpected events.
4. It should operate in real-time.
5. It should assume that it can never have a complete and accurate model of the environment.
6. It should assume that it has only limited control over its surroundings—the existence of exogenous actions.
7. It should assume that it has imperfect sensors and motor-controls.
8. It should be able to perform tasks that require deliberation.
9. It should maintain a task-directed mental model of its surroundings that supports deliberative activity.

The above list is by no means complete. For example, one can also add communication, sociability, self-learning, and adaptability; however, we will not consider these

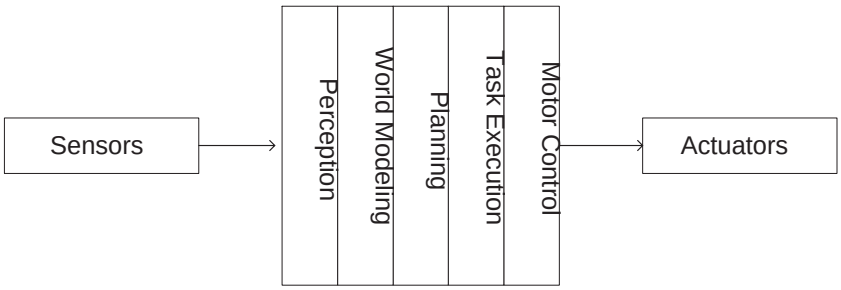


Figure 1: A deliberative controller consists of one module that takes care of perception, world modeling, decision making, and acting needs of a robot.

requirements. In this report we will restrict our discussion to the control structures required for everyday operation of an autonomous agent in its environment.

3 Historical Development of Robot (Control) Architectures

In this section, we examine various robot control architectures. We will, in particular, focus on their respective strengths and weaknesses, and explain why some architectures are superior to other in terms of robustness, performance, and ability.

3.1 Deliberative Robotic Agent Architectures

Early attempts at designing autonomous robotic agents employed a sense-model-plan-act (SMPA) architecture with limited success (Fikes and Nilsson1971). Here, the agent maintains an explicit model of its goals, abilities, and the world. The sensing module keeps the internal model consistent with the external reality, the planning or reasoning module uses the internal model to plan a course of action, and the execution module executes the

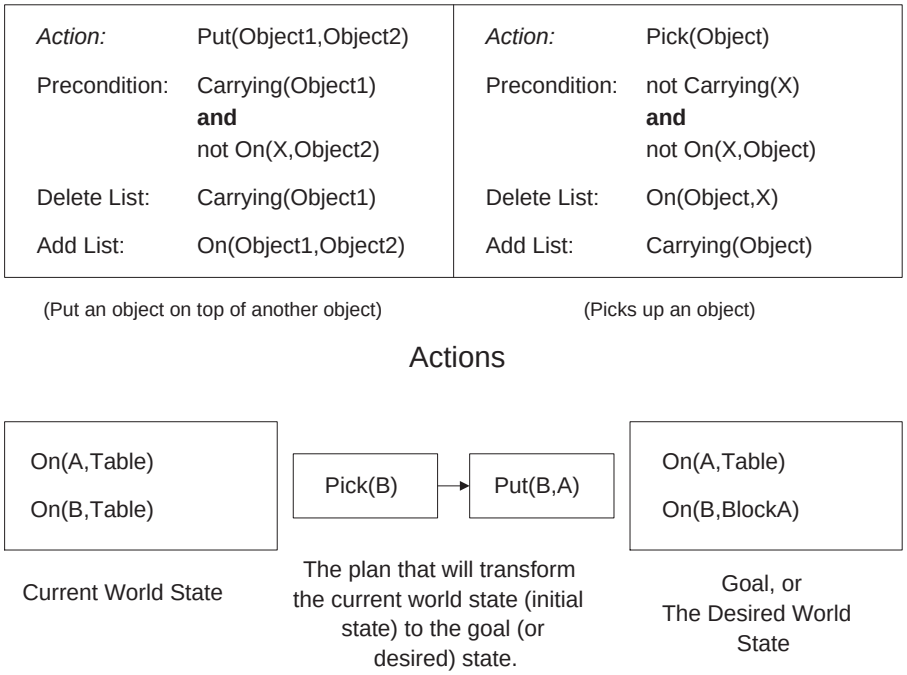


Figure 2: STRIPS maintains a world model that encodes what actions are available to the planner, when these actions are legal (i.e., can be executed), and how these actions affect the world. STRIPS also maintains an abstracted world state (e.g., object A is on the table). The goal is represented as a world state where certain properties hold, e.g., object B is on top of A. Add and delete lists encode the effects of an action on the world. Given an initial world state (in the case of a robot, this would be the current world state), and the desired world state (the goal is also represented as a world state, e.g., object B is on top of object A), STRIPS employs "means-end" analysis to construct a sequence of actions that will transform the initial world state to the desired world state.

plan. These agents do not perform well in the real world, which is hard to model and waits for no one. Their sensing and planning modules fail to keep up, thereby, rendering the agent unsafe. Even though newer SMPA architectures support interleaved planning and execution, they to have difficulty attaining the desired performance (Ambros-Ingerson and Steel1988).

"Shakey", built by the Stanford Research Institute in the early 1970s, is the most famous deliberative robot, and it illustrates the strength and weakness of deliberative architectures (Nilsson1984). Shakey occupied a simple blocks world where, on a good

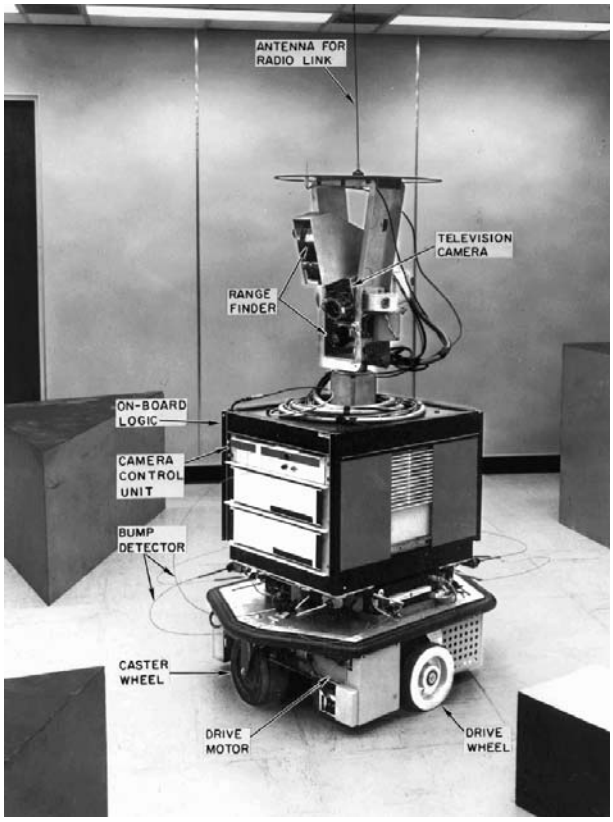


Figure 3: Shakey is the most famous deliberative robot. Shakey can accompalish many tasks that require forethought in the blocks world; however, it does not fare well in the real world, as it is bogged down by the sheer complexity of the real world.

day, it could formulate and execute over a period of hours, plans involving moving from place to place and pushing blocks to achieve a goal. Shakey's sensors consisted of a camera and a laser range-finder, and its reasoning module comprised a STRIPS-based planner (Figure 2) (Fikes and Nilsson1971). The planner represents the current world-state as a set of logical statements that specify what is true in the world. It encodes the effect of the actions by maintaining add-delete lists for every action. The add list specify conditions that become true after executing the action, and the delete list specify conditions that will cease to be true after the execution of the action. It represents a goal as a "desired" world state and employs means-end analysis to plan the sequence of actions that will

achieve the goal by transforming the current world state into the desired world state. At each instant, Shakey (1) constructs a spatial model of its world through edge processing of camera images and laser-range measurements, (2) updates the internal world model by adding or deleting logical statements, (3) computes the plan, and (4) attempts to execute this plan. Shakey's success is limited to its blocks world. Its SMPA cycle is slow and it fails in more dynamic settings.

Learning from the shortcomings of the strict sense-model-plan-act paradigm, researchers proposed to interleave sense-model-plan and act cycles. Integrated Planning and Execution Monitoring (IPEM), proposed by Ambros-Ingerson and Steel (Ambros-Ingerson and Steel1988), implements a control strategy to decide when to plan and when to execute. IPEM is able to detect when the current plan becomes invalid because of a failed execution of an action or due to some unexpected event in the world. It then backtracks to the last decision point that is still valid and replans. If multiple actions are possible, a scheduler is used to arbitrate between them. Even this variation of the sense-model-plan-act paradigm does not yield the desired performance.

Building and maintaining a detailed internal world model is difficult and time-consuming, and often counter-productive. A detailed internal world model will always lag behind the outside world due to perception delays; this situation worsens with the increasing complexity of the internal world model. Planning systems assume that the world behaves in a predictable way. For example, each action has a well-defined effect on the world. It also assumes that the world will not change significantly during the course of planning, as otherwise the plan will become invalid. Both assumptions do not hold in the real world.

There is another, perhaps more fundamental, reason why deliberative robotic agents

Figure 4: A behavior-based controller consists of concurrently running competence modules (or behaviors) that take care of their own sensing and acting needs. The over-all behavior of the controller is due to the interaction of these modules with each other and with the world.

do not perform well in the world. These agents assume that actions and the world states are discrete; this is not true for a world that obeys Newtonian physics. Discrete actions and world states are theoretical constructs that make planning tractable, but they have no real world counterparts. In the real world of continuous activities, these activities blend into each other, and more than one of them can be active simultaneously, which makes it harder to define precisely the effect of an action/activity.[1]

We require the means to abstract away unnecessary details from the real world and present a discrete and predictable world to the deliberative module. Reactive systems provide us with such a mechanism.

3.2 Reactive Robotic Agent Architectures

Animals are natural autonomous agents that are able to make decisions that ensure their survival and achieve their goals by taking into account external factors[2] and internal motivations[3]. To this day no robot can match all the abilities of even simple animals.

[1]The notion of *emergent* effects comes to mind.

[2]Imagine a zebra trying to get away from a lion. It is safe to assume that the zebra's actions are influenced by the proximity of the lion.

[3]A giraffe makes its way to a pond to quench its thirst. In this scenario, the actions of the giraffe are influenced by external factors, such as the location of the pond, and its internal motivation, which

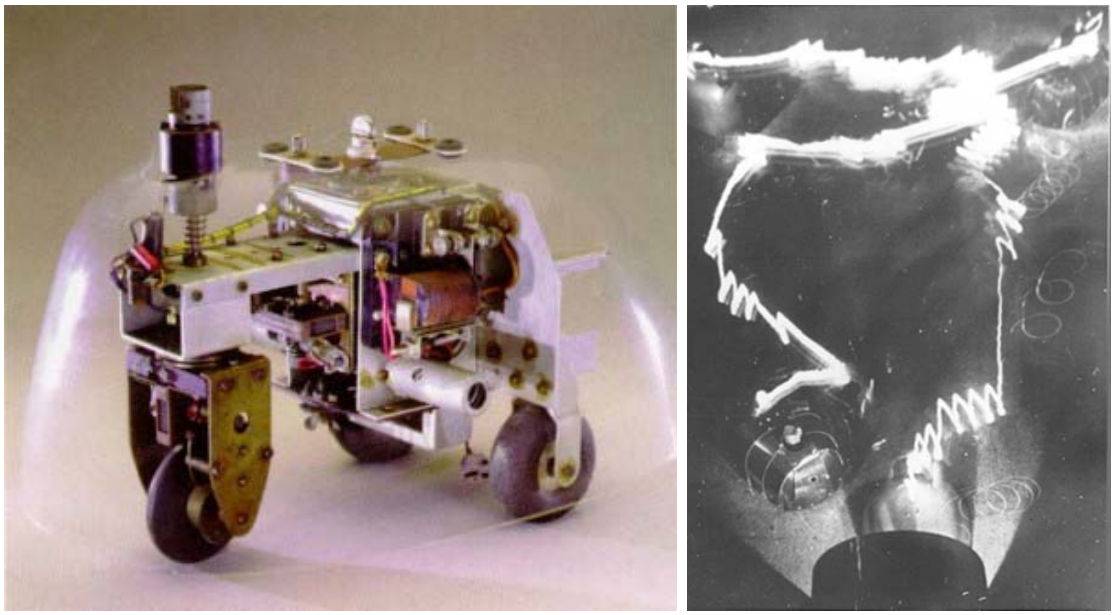
Even insects, such as ants, that do not appear to have deliberative capabilities perform better than any existing robot. This observation, combined with a disillusionment with the traditional AI approach, led researchers to think of alternative approaches for robotic agent design. The reactive or behavior-based approach appeared in the 1980s. These robots operate in real-time and can handle the dynamic events of the world better than their deliberative counterparts. They accomplish this by tightly coupling sensing to action through simple processes, called *behaviors*. Every behavior handles its own perception, modeling, and execution needs, and is only responsible for a small portion of the overall behavioral repertoire of the robot. The interaction of these behaviors gives rise to the desired overall behavior, which is often called the *emergent behavior*.

The earliest instance of a behavior-based agent is the schematic sowbug[4] that was presented by psychologist Tolman in 1939 (Tolman1939). Tolman put forward the theory of purposive behaviorism and claimed that a tight connection exists between stimulus and response. Furthermore, he said that high-level factors, such as motivation, experience, and purpose, affect this connection. He called this connection *behavior*.

Complex behavior of an agent does not necessarily indicate a complex internal structure, so it is possible to realize robots having elaborate behaviors using simple components. For example, Walter's (Walter1950; Walter1951; Walter1953) mechanical turtles, Elmer and Elsie, display complex behavior despite their simple design, which consists of a light sensor, a touch sensor, two vacuum-tube analog computers, a steering motor, and simple behaviors such as `move-towards-light` (Figure 5). The interaction with the world results

includes its thirst.

[4]Tolman's schematic sowbug exhibits phototactic behavior. For an implementation see (Endo and Arkin2000).



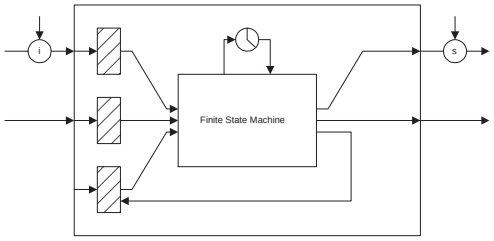
(a) (b)

Figure 5: (a) *Machina Speculatrix*.. (b) The trajectory followed by Walter's turtles during one of their runs. Walter's turtles exhibit complex behaviors inpsite of simple control strategy. This lead to the realization that complex behavior does not need a complex control mechanims; it might just be a reflection of a complex environment. Furthermore, complex behavior does not require an internal model of the environment. Pictures are borrowed from http://www.robotics.com

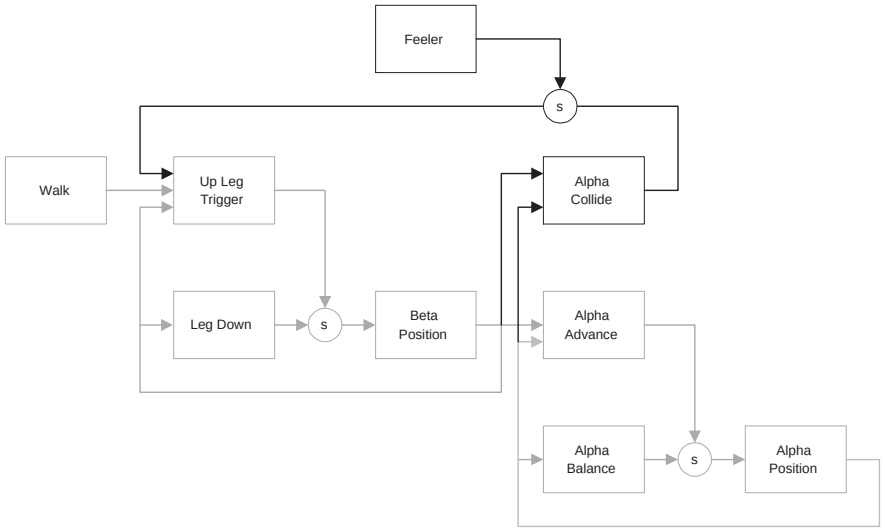
in an interesting and hard to predict behavior. They appear curious as they explore their environment in a speculative manner; Walter therefore named them *machina speculatrix*.

In spite of these early forays, the behavior-based approach did not catch on in the robotics research community until Brooks introduced the *subsumption* architecture, which demonstrated that the interaction of independent components can produce coherent "intelligence" (Figure 6). Brooks claimed that "the world is its own best model," and argued that it can simply be sensed as necessary. He rejected the necessity of a symbolic world model for intelligence. He built many subsumption-based insect-like robots (Brooks1986b; Brooks1986c; Brooks1986a; Brooks1990b; Brooks1990a), which could explore their en-

(a) Robot Genghis



(b) Augmented Finite State Machine



(c) A subset of the full AFSM network that conrols Genghis.

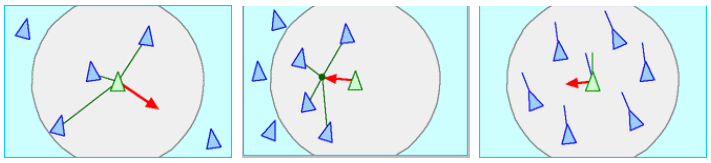
Figure 6: (a) Genghis is a subsumption-based reactive robot. (b) An Augmented Finite State Machine (AFSM) consists of registers, a clock, and a regular finite state machine. Input messages are delivered to the registers, and output messages are generated on the output wires. A new AFSM is added by connecting its wires to the exisiting wires through suppressors (denoted by the circle "s") or inhibitors (denoted by the circle "i"). The new AFSM adds to the overall ability of the AFSM-network. For example, (c) is a subset of the overall AFSM network that controls Genghis. The greyed portion controls one of the six legs of Genghis (it is repeated six times in the overall network). It enables Genghis to walk without any feedback. We can add higher-level of competence by simply adding appropriate AFSMs. The black portion represents the addition to the existing network (i.e., the greyed portion). Now, Genghis can incorporate information from its proximity sensors (i.e., its whiskers) into its walking, and as result it handles rough terrain much better. Notice that the addition does not require any modifications to the existing network. Similarly, an even higher-level of competence, such as prowling and collision-avoidance behaviors, can be achieved just by adding more AFSMs. (a) is borrowed from http://www.ai.mit.edu/projects/genghis/genghis.html.

vironment, build maps, navigate, and interact with people—feats, no robot previously accomplished. The subsumption architecture is a network of message passing Augmented Finite State Machines (AFSMs) that are hierarchically organized into *layers*. There is no central world model and all layers take care of their own perception and world modelling needs. Inhibition and suppression between layers produce the desired overall behaviors. New behaviors can be added to the behavior repertoire by adding more layers.

Around the same time when Brooks was building reactive robots, other researchers found evidence supporting the basic assumption of the behavior-based approach, i.e., interaction of simple rules can produce elaborate and coherent overall behavior. Reynold's virtual *boids* exhibit elaborate global behavior (flocking) while using simple local rules (Reynolds1987). Each boid follow three rules: maintain a safe distance from all visible boids, move towards their centroid, and match their average velocity (Figure 7). The interaction of these rules produce realistic, life-like flocking, schooling, and herding behaviors[5]. Braitenberg's imaginary *vehicles* (Braitenberg1984), which consist of simple components, appear to have emotions like love, fear, and logic. Here too, the interplay of simple components among themselves and with the environment gives rise to elaborate behavior. In his book, *The Society of Mind*, Minsky suggests that our mind is organized as a collection of specialists, and the competition and cooperation of these specialists produce the overall behavior (Minsky1985). Ethologists, who have independently reached the same conclusion, propose that the behavior of an animal is an outcome of competition and interaction among many behaviors (Lorenz1973). Ethology, especially, has proven

[5]Reynold's model belongs to the class of individual-based models where local interaction of the members of a population generate the overall behavior for that population.

useful for designing behavior-based agents, as it provides the necessary blueprint required for identifying and designing behaviors and managing their interaction.



(a) Separation (b) Cohesion (c) Alignment

Figure 7: Each boid follows three rules: (a) separation, (b) cohesion, and (c) alignment. The interaction of these *local* rules gives rise to schooling, herding, and flocking behaviors. The images are borrowed from Reynold's website http://www.red3d.com/cwr/boids.

An argument in favor of behavior-based AI is that most of the time most robots just perform routine tasks, such as recharging their batteries and moving around safely (Agre and Chapman1987).[6] These tasks require little or no abstract reasoning. Deliberation is only required to perform novel tasks, and most tasks, once learned, can be accomplished using purely reactive means. This might be why deliberative agents have failed to match the success of behavior-based agents.

In the 1990s, several researchers demonstrated impressive examples of ethologically-based autonomous agents (see, e.g., (Tu and Terzopoulos1994), (Blumberg1997), and (Arkin, Fujita and Takagi T.2001)). Tu and Terzopoulos established that it is possible to build complex ethologically-inspired behavior-based agents by developing virtual, artificial fishes that "live" in a physics-based virtual sea. These biomimetic agents exhibit a broad behavioral repertoire, including behaviors such as foraging, predation, schooling, and mating. They are a departure from the strict subsumption style agents, as they in-

[6]This also holds for humans.

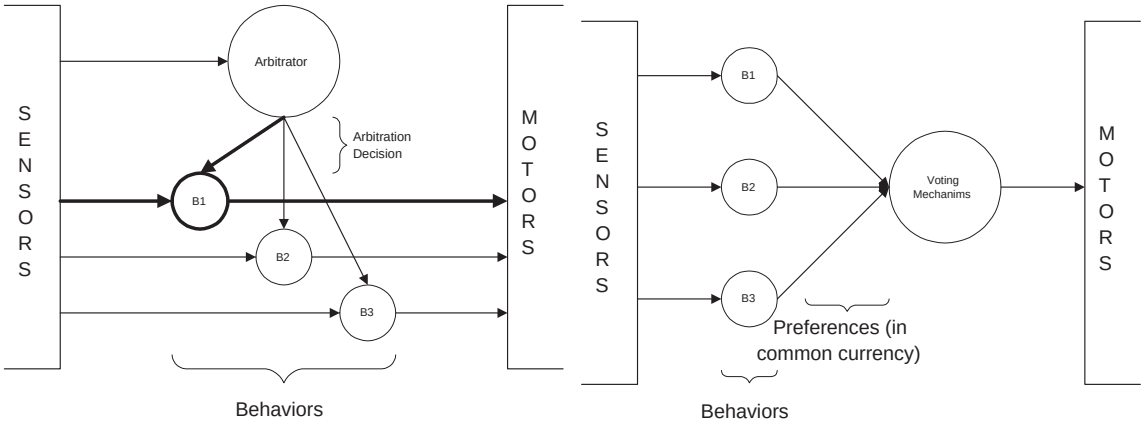
clude a simple mental state, an attention-based perception mechanism, and an elaborate action-selection scheme. The fishes are driven by motivations, such as fear, and desires, such as hunger and libido, which are recorded in their mental state. The action-selection mechanism, which is implemented as a decision tree, takes into account sensory information, motivations, and desires to choose an appropriate behavior. The strict hierarchy among behaviors, the continuity of mental state, and the constancy in habits provides persistence among behaviors.

Designing behavior-based controllers is a tricky engineering endeavour, as we must deal with emergent properties that defy formal modeling. Some of the issues faced when designing behavior-based system are:

- Behavior arbitration
- Behavior coordination
- Persistence among behaviors
- Behavior dither
- Transitions between behaviors
- Behavior preference unification
- Working memory (or short-term memory).

In the remainder of this section, we will discuss how various architectures found in the literature address these issues.

Managing behavior interaction to produce the desired overall behavior is quite challenging, as there are possibilities of dead-locks, race-conditions, and the ubiquitous behavior-dither. Behavior interaction involves both behavior selection and behavior preferences unification.



(a) Centralized arbitration (b) Distributed arbitration

Figure 8: (a) In centralized behavior arbitration scheme, the arbitration module activates appropriate behavior(s)—in the figure, only behavior B1 is active. (b) In distributed behavior arbitration scheme, all behaviors compute their relevance in a common currency, and the motor controller give preference to motor commands from more relevant behaviors.

A *behavior arbitration* scheme selects appropriate behaviors given the current sensory inputs and internal motivations. It can be implemented in a single specialized behavior arbitration module, or it can be the consequence of the design of individual behaviors. In the first approach, the behavior arbitrator takes into account the sensory inputs and internal motivations and activates appropriate behaviors. Here, the design of individual behaviors is simpler; however, the behavior arbitrator becomes increasing complex with the addition of new behaviors, which can adversely affect the real-time performance of the robot. In the second approach, all behaviors compute their own "relevance" (in a

predetermined common currency) to the current situation. Here, adding a new behavior is easier, as it does not effect other behaviors; however, the new behavior must be able to compute its relevance in the common currency. Hierarchy among behaviors (Tyrrell1993; Tu and Terzopoulos1994) and behavior-groups (Minsky1985; Blumberg1994) simplify behavior interactions, thus making behavior-arbitration easier. Tyrell uses a fixed hierarchy among behaviors that is encoded as a decision tree, so selecting an appropriate behavior is straightforward, i.e., a single pass through the decision tree (Figure 9). Blumberg groups similar behaviors together (i.e., behaviors that share resources, for example, behaviors `goto-loc` and `avoid-collision` belong to one group, and behaviors `talk` and `eat` belong to another group), which reduces interaction among behaviors that belong to different groups. In this scheme, behaviors compete with other behaviors within the same group. Another commonly used scheme for behavior arbitration is the "homeostasis regulation rule." Here, a number of internal (usually motivational) variables along with their allowable ranges are specified, and the behavior arbitrator selects behaviors to keep the values of interval variables within the allowable ranges (Arkin1988).

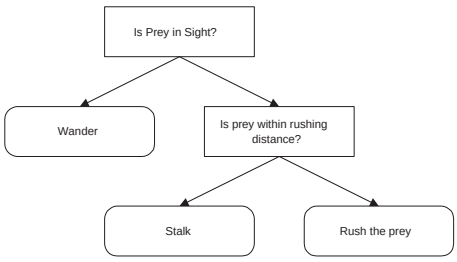


Figure 9: A decision tree encodes heirarchy among behaviors, and can be used for behavior arbitration.

In the subsumption architecture only one behavior issues preferences for the motors (for example, either the `goto-loc` or the `avoid-collision` behavior can issue motor

commands at any instant), so it does not fully utilize the interaction of multiple behaviors. Payton remedied this problem by allowing multiple behaviors to issue preferences for a common control (Payton1990). The net preferences is then the sum of the active behaviors' preferences weighted by their respective relevances. Tyrell modifies Payton's approach, and proposed that preferences for a given control should be calculated as a weighted average of the highest single preference and the sum of all preferences for that control (Tyrrell1993). Combining behavior preferences is especially easy in Arkin's motor-scheme framework (Arkin1998). Here, a behavior is a mapping from sensor vectors to motor vectors. Behavior preferences, which are vectors, are combined through simple vector addition. We need to keep in mind that in all of the strategies for combining behavior preferences, the most difficult element is to weigh different preferences correctly. Unfortunately, there are no fixed rules/algorithms to resolve this issue, and the designers have to rely upon their intuition and experience to come up with an appropriate set of weights that result in the desired overall behavior.

Persistence among behaviors is necessary for a coherent and useful overall behavior; a robot that constantly alternates between two goals appears undecided, and it might never achieve either goal. Behavior-dither is a consequence of a poor behavior-arbitration scheme—one that swings back and forth between different behavior choices. The careful use of internal mental states can avoid this undesirable situation. For example, in the case of Tu's artificial fishes (Tu and Terzopoulos1994), the continuity of mental state and the constancy in habits provides persistence among behaviors. Another approach is that the currently active behavior inhibits a non-active behavior by a factor that varies inversely as the duration of the active behavior. For example, behavior A becomes active at time

t_0, then at time t, where $t > t_0$, the inhibition for behavior B will be proportional to $\frac{1}{f(t-t_0)}$. This simple scheme ensures that once activated, a behavior will remain active for some minimum duration.

A related problem is how to avoid one behavior from taking over and never relinquishing control. This is a common occurrence in situations where the robot forever tries to achieve some goal. The solution here is to associate a boredom factor with the active behavior that increases with its duration. The boredom factor decreases the relevance of the active behavior over time, so even if some goal is not achieved, the behavior-arbitrator will activate another behavior. This way, a behavior can not be active for more than some maximum duration.

Transitions between behaviors require extra care, as the controller may exhibit choppy, or worse, incorrect overall behavior. This is especially true for behaviors that share resources, such as motors, which is a common situation for low-level behaviors. One solution is to force the active behavior to return the shared resource in a consistent state before shutting off (Blumberg1997), which can be achieved through "action buffering" (Perlin and Goldberg1996). Another, more involved scheme is to implement fuzzy transitions where the state of the shared resource depends upon both the old and new behaviors. See (Minsky1985) and (Blumberg1997) for further details.

The biggest weakness of the subsumption architecture is that it does not maintain any mental state, without which it is impossible to (1) realize goal-directed reactive behavior, (2) avoid behavior dither, and (3) implement behavior persistence. More recent behavior-based architecture allow for a small working memory, or short-term memory. Tu's artificial fishes have a single element behavior memory that remembers the last active behavior.

In case a behavior, such as `follow`, is interrupted by the invocation of another (more relevant) behavior, say `avoid-collision`, the arbitration module reinvokes the last active behavior, `follow` once the danger of collision has passed. An internal mental state is also necessary to deal with perception delays—modern behavior-based robots are equipped with vision-based sensors that require considerable processing time. Moreover, Internal mental state is also necessary for tracking motivational variables.[7]

It is, however, important for the internal mental state within the reactive module to be able to operate in real-time, as otherwise it defeats the main purpose of the reactive module: real-time response. One way to do that is to keep the internal mental state simple, i.e., it should only store information relevant to the task at hand. For example, Tu's fishes use an attention-based perception system that filters out irrelevant details. Attention-based perception systems are biologically plausible, and they have found broad acceptance in the behavior-based AI community.

As stated earlier, ethological models provide a basis for the kinds of behaviors we should realize within a robot. A testament to this statement is Sony Corporation's life-like robot dog Aibo (Arkin, Fujita and Takagi T.2001). Aibo can operate competently in the real-world. It exhibits interesting behaviors that are commonly associate with a pet dog: it appears to yearn for its master's approval, it gets hungry, it exhibits moodiness, and it likes to play with bright balls (or things that look like bright balls). Aibo's behavior controller implements a subset of the complete ethogram (categorization of behavioral patterns that span the range of an animal) of a dog. Its behavior arbitrator takes into

[7]One aspect of behavior-based architectures that we have deliberately avoided is *learning*. Various researchers have studied behavior learning.

account external stimuli (such as the proximity of food) and internal motivations (the urge to play with a bright ball) and uses a homeostasis regulation scheme to choose appropriate behaviors (Arkin1988). For behavior coordination (i.e., to resolve behavior-dither and implement behavior-persistence), it uses Ludlow model of lateral inhibition (Ludlow1976).

Tsotsos cogently argues about the limitations of a strict behavior-based controller (Tsotsos1995)—one that does not allow any internal mental model. He shows that strict behavior-based controllers cannot exhibit human-level intelligence. In the strict behavior-based controller, a behavior is a stimulus-action pair, and at each instant, the controller searches for relevant behaviors to activate. Tsotsos shows that unbounded stimulus-action search is NP-complete; thereby showing that strict behavior-based architecture is not enough. Furthermore, he shows that internal world model, explicit goal representation, and hierarchy among behaviors is required to reduce the complexity of the stimulus-action search. He also shows that unbounded vision is intractable, which essentially means that no strict behavior-based agent can use vision as its sensor. This too excludes the possibility of designing a subsumption style agent with human-level capabilities.

Tsotsos addresses the shortcomings of the subsumption architecture in the S* framework (Tsotsos1997). S* allows an agent to have an internal mental model, and it generalizes the notion of a behavior by adding modeling and planning components to each behavior. The behaviors can read from sensors (i.e., the real world) or from the internal world model. Similarly, the behaviors can write to actuators or to the internal world model. S* provides general guidelines about how to design a controller, but its main contribution is to point out the importance of perceptual attention, internal models, and

explicit goal representations when designing an intelligent agent.

To summarize, behavior-based agents can survive in a complex and unpredictable environment for prolonged periods without human intervention, but they can not reason about the goals which are implicit in their designs. Behavioral agents cannot "think" ahead and perform tasks that require deliberation, such as path planning. Also, they are not easily amenable to formal analysis, which is a big concern in safety-critical applications. It appears unlikely that a purely behavior-based agent can ever demonstrate human-level capabilities.

3.3 Hybrid, Reactive/Deliberative Architectures

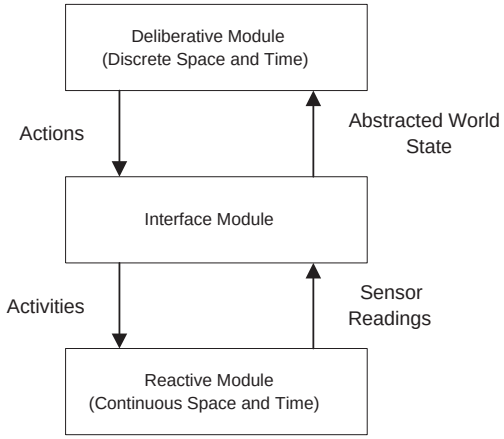


Figure 10: Hybrid controllers combine deliberative and reactive components. Here, the reactive module mediates between the world and the deliberative module.

Hybrid robotic controllers promise greater utility, reliability, and performance through the combination of reactivity and deliberation. The common theme among hybrid architectures is that the reactive component deals with the continuous, detailed, and uncertain aspects of the world, and that it mediates between the world and the deliberative component. The deliberative component experiences the outside world as a well-behaved,

discrete phenomenon—actions are discrete and have well-defined effects, and world states, which are represented by a set of symbols (fluents), are akin to snapshots in time. It maintains a detailed mental model of the agent and its environment and it uses this model to form strategies that will achieve the long-term goals of the agent.

Imagine two tasks: (1) following a person and (2) travelling from one room to another within a building. A deliberative robot cannot perform both tasks satisfactorily; it will have trouble following the person. No amount of deliberation can yield a correct plan in advance, as the trajectory of the person is not known *a priori*. The only other option is to re-compute a new plan at every instant, which is clearly inefficient and cannot be realized in real-time. Similarly, a reactive robot will also have trouble performing both tasks. It will have no way of finding a route between the two rooms, except through serendipity, since it lacks an internal world model necessary for this purpose. The good news is that a hybrid robot can perform both tasks easily. It can employ the deliberative module to form plans when required, as when it needs to go from one room to the other, and use the reactive module to take care of time-critical tasks, such as following a person and moving between two locations safely.

Hybrid controllers can be divided into two classes: (1) hybrid controllers that do not distinguish between deliberative and reactive activities and use uniform computational structures throughout, and (2) those that treat deliberation and reactivity as two separate activities that require different computational mechanisms. In practice, the controllers that belong to the second class outperform their counterparts, because they can take full advantage of specialized deliberative (e.g., a symbolic planner) and reactive (e.g., a behavior-based controller) computational structures. It is no surprise that most successful

hybrid controllers consist of asynchronous, dissimilar deliberative and reactive components (see the discussion on RHINO and MINERVA on Page 32).

3.3.1 Homogeneous Hybrid Architectures

We begin by examining "Soar", which uses homogeneous computational structures for both deliberation and reactivity.

Soar (Laird and Rosenbloom1990) is a cognitive architecture that combines deliberation and reactivity. It has been used successfully to control virtual characters in simulated worlds[8]; however, it has only met with limited success in the domain of real robots. Soar makes no distinction between deliberative and reactive activities. It stores all knowledge as *if-then* rules (*productions*), and its problem-solving cycle consists of activating the relevant productions—an activity that continues till there no more productions fire; i.e., when Soar reaches *quiescence*. Soar embeds deliberative activity within the productions (some production may lead to planning), so reaching quiescence can take an arbitrarily long time, which rules out reactive performance. Soar performs deliberation whenever it fails to reach quiescence and it saves the result of deliberation as a production, thus avoiding deliberation in similar situations in the future. This built-in learning mechanism is called *chunking*, and it increases the ability of the agent over time. This, however, has an undesirable side-effect: now Soar has to consider more productions, so its performance decreases.

[8]SOAR has successfuly controlled Quakebot—the main character in the popular first-person shooter game Quake by Id Software. SOAR is also used to design a realistic arial war simulation.

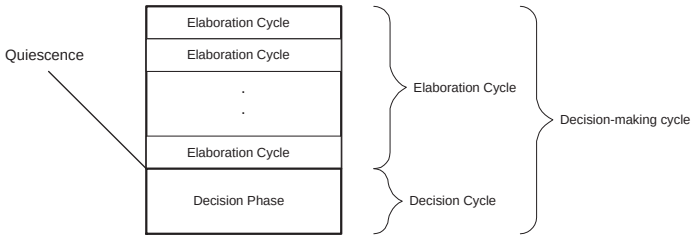


Figure 11: Soar's decision-making cycle consists of an elaboration phase and a decision phase. The elaboration phase in turn consists of one or more elaboration cycles. Each elaboration cycle is analogous to firing a production. The elaboration phase continues till no more changes are made to the working memory—Soar is said to reach quiescence at this stage—and at this point the decision phase chooses an operator (action). It uses automatic subgoaling to resolve impasse during problem solving activity. Conceptually, this is thought of as Soar not being able to do automatic processing and having to resort to some sort of deliberative decision making for what to do next. Soar saves the result of this deliberative process as a production through chunking; thereby, avoiding the impasse that led to subgoaling in the future.

3.3.2 Heterogeneous Hybrid Architectures

Heterogeneous hybrid architectures found in the literature usually consist of two specialized modules: a deliberative and a reactive module. We begin by discussing the relationship between discrete actions and continuous activities. We will also make an argument in favor of abstract, or so called "sketchy," plans.

A deliberative module (planner) constructs plans, which are usually construed as lists of actions. The quality of a planner (i.e., its efficiency and the reliability of its plans) depends upon the actions that are available to it. These actions are called *primitive actions*, and they should have well-defined effects and unambiguous success and failure conditions. In practice, planners fare better when dealing with more abstract primitive actions. Abstract primitive actions hide much of the complexity of the world, so planning takes less time.[9] Moreover, they can also make a plan more robust to the changes in the

[9]When dealing with low-level actions, the search-space for possible plans is larger.

world by handling "routine" error situations (at run-time) that would otherwise invalidate the plan.[10] For instance, both set of actions are valid within the context of the example on page 23:

$$Set1 = \{\texttt{forward(x)}, \texttt{turn-left}(\theta), \texttt{turn-right}(\theta), \texttt{break}\},$$

and

$$Set2 = \{\texttt{follow-person}, \texttt{goto-loc(l)}\}.$$

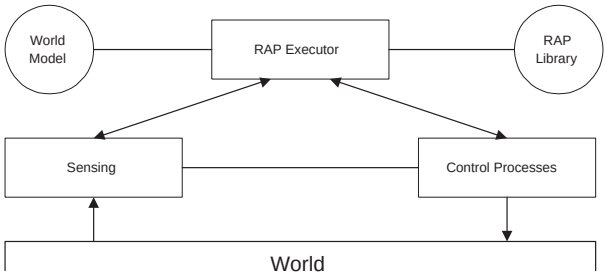
$Set1$ is clearly inferior to $Set2$, however, both in terms of the time it takes to formulate the plan and the robustness of the plan. As stated earlier, primitive actions do not exist in the real-world; however, we can define them in terms of activities, which are real-world equivalents of actions. The associated difficulty, which is considerable to begin with, is greater for more abstract primitive actions. The upside is that the reactive approach is well-suited for constructing these actions correctly and robustly.

From the reactive module's point-of-view, primitive actions are just high-level behaviors. In simplest terms, actions of the deliberative module are tied to appropriate behaviors in the reactive module. This connection forms the basis of the interface between the two modules with vastly different properties and structures. For any hybrid architecture, this interface is a crucial design decision that affects both its ability and its performance. Too much reliance on the deliberative module renders it incapable of handling time-critical situations; whereas, a bloated reactive module makes it fragile, as reactive modules tend to get stuck in "local minima."

[10] In general, low-level primitive actions fail, and as a result, invalidate the plan more often than their more abstract counterparts.

Firby's Reactive Action Package (RAP) framework provides an elegant scheme for constructing abstract primitive actions on top of a behavior-based reactive module (Firby1989; Firby1992; Firby1994). It consists of two components: (1) a library of RAPs and (2) an execution module (Figure 12). Each primitive action is encoded as a RAP, which is a control structure that can encode multiple situation-dependent execution, error-handling, and termination-evaluation strategies for that action by specifying the interaction of various activities. The execution of a RAP (and hence the associated primitive action) is handled by the RAP executor, which chooses the most appropriate method for execution at run-time. This situation-dependent execution paradigm makes a primitive action more robust to the changes in the environment than is otherwise possible. Moreover, it removes the burden from the deliberative module of handling every contingency in the environment, so the deliberative module can construct sketchy plans—plans that constitute high-level actions—and let the actions figure out the details at run-time. The ability of describing a RAP in terms of other existing RAPs is especially powerful, and it makes this framework ideally suited for describing hierarchical task networks.

Firby mentions how the RAP framework can provide the necessary interface between the deliberative and reactive modules, but he does not describe any specific planner (Firby1992). He views RAPs as pre-coded hierarchical plans that can be expanded into subgoals, other RAPs, and activities at runtime. Within this view, the executor activates the RAP that will achieve the current goal of the robot, and the RAP remains active until its success (the goals for this RAP have been achieved) or failure (the goals for this RAP can never be achieved) conditions are met. No on-the-fly planning is performed, which severely limits a RAP-based robot's ability to handle novel situations; Gat addresses this



(a)

```
(DEFINE-RAP
    (INDEX (goto-loc ?pos))
    (SUCCESS (mypos ?pos))
    (METHOD
        (CONTEXT (not danger-collision))
        (TASKNET
            (t1 (turn-toward-pos)
                (pos ?ahead for t2))
            (t2 (go-straight))))
    (METHOD
        (CONTEXT (danger-collision))
        (TASKNET
            (t1 (avoid-collision)
                (not danger-collision for t2))
            (t2 (turn-toward-pos)
                (pos ?ahead for t3))
            (t3 (go-straight)))))
```

(b)

Figure 12: (a) RAP Architecture. (b) An example RAP that implements a robust high-level behavior `goto-loc` using low-level activities (`turn-toward-pos` and `go-straight`) and another RAP `avoid-collision` .

limitation and shows how RAPs can connect a classical AI planner with a control theory based reactive module (Gat1992).

Gat proposes a three-tiered control architecture called ATLANTIS[11] that consists of three heterogenous, asynchronous modules: controller, sequencer, and deliberator (Gat1992). The controller uses classical control theory techniques to implement motor controls, such as `move-forward`, and low-level behaviors, such as `follow-path`. The sequencer is a RAP module that interfaces the other two modules. Finally, the deliberator consists of a classical AI planner, and is responsible for planning to perform high-level tasks of the robot. A note-worthy feature of ATLANTIS is the relationship between the deliberator and the sequencer. It follows Agre and Chapman's theory of plans-as-communications (Agre and Chapman1987), and the deliberator module merely advises the sequencer. This proves to be a powerful mechanism for combining reactivity and

[11]ATLANTIS stands for "A Three-Layer Architecture for Navigating Through Intricate Situations."

deliberation, as it allows deliberation without affecting reactivity—a feature previously missing from hybrid architectures. An ATLANTIS-based robot can boast reactive performance similar to that of a subsumption-based robot and deliberative capabilities similar to those of Shakey.

Connell's SSS, which stands for "servo, subsumption, symbolic," architecture combines a servo-control layer, a subsumption layer, and a symbolic layer (Connell1992). The lower-most layer implements servo-loops for various actuators. The middle layer is a subsumption style controller, which consists of behaviors that handle "events of interests" by choosing appropriate *setpoints* for the servo-loops, such as the desired speed for the wheel-speed servo-loop. The symbolic layer maintains a coarse geometric map of the world, which is used to plan a route between the source and destination. To traverse this route, the symbolic system enables/disables appropriate behaviors present in the subsumption layer before and after each segment of the route. The subsumption layer has considerable freedom for following the current segment; however, it requires immediate response from the symbolic system for the events that it can not handle (for example, the end of the current segment, or if it is stuck). This imposes an undesirable constraint on the symbolic layer, which now must perform in real-time. The symbolic layer accomplishes this by using special-purpose data structures called *contingency tables*, which contain pre-compiled responses to various events that the symbolic layer must handle. Here, the decoupling between the symbolic layer and the subsumption layer is not sufficient; a better approach is to use the deliberative layer only as an advisor.

Autonomous Robot Architecture (AuRA) is a hybrid architecture for autonomous robots capable of both deliberative planning and reactive control (Arkin1998; Arkin1992;

Arkin1990). AuRA principles has been applied in a variety of domains like navigation, robot competitions, multi-robot teams, and vacuuming. A multi-agent implementation of AuRA won the 1994 robotic competition. Aura consists of four modules: a mission planner, a spatial reasoner, a plan sequencer, and a schema-based reactive module. The mission planner sets high-level goals for the robot, and the spatial reasoner constructs plans that will achieve those goals. The plan sequencer executes these plans over the reactive module by invoking appropriate behaviors. Aura does not have true deliberative capabilities, i.e., it does not have a planning module. Aura's plans, which are represented as Finite State Acceptors (FSA), are hand-coded. Each state represents a specific combination of behaviors that accomplish one step of the task (i.e., on path leg). The reactive module is a schema-based control system where all active behaviors contribute towards the overall motion. Each motor-schema (essentially a behavior) receives sensory data and generates its response as a vector. The vectors are then combined to get the actual response of the agent.

PLAYBOT is a visually-guided robot that helps physically disabled children in play (Tsotsos et al.1998). PLAYBOT uses its robotic arm to interact with the environment, e.g., picking and dropping objects. Its vision system (Dickinson et al.1993), which combines reactive behavior and planning, is capable of performing complex visual tasks, such as object search, recognition, and localization. The vision system consists of two layers that run independently of each other. The lower layer, which consists of reactive behaviors, is always active and performs object recognition. It continuously extracts information about the world from the images and uses this information to construct and update the internal world model. The top layer consists of a planner that maintains the high-level

goals of the vision system. It uses the world model to reason about the task at hand and directs the recognition layer. For example, it can direct the recognition layer to look in a particular direction, to look for a particular object, or to look for objects in a particular location etc. In addition to that, it can help the recognition layer in the task of recognizing objects, e.g., by suggesting a better view point so as to disambiguate an object. Apart from showing how perception systems capable of complex visual tasks can be realized by combining reactivity and deliberation, this work proposes a straightforward scheme of injecting high-level advice into the reactive modules; the planner controls the recognition layer through its state variables. It also furthers the idea that hybrid architectures should implement an advisor-client relationship between the deliberative and reactive module.

ARK project has designed a series of autonomous mobile robots capable of safely navigating within industrial environments. These robots employ a sensor called *Laser Eye* that combines vision and laser ranging for navigation and self-localization (Jasiobedzki1993). Indoor navigation and localization is accomplished by laser ranging, while navigation in open areas is carried out by visually detecting landmarks. The control architecture consists of two modules. The high-level module is responsible for planning robot actions, path planning, selecting landmarks for sighting, and user interactions. The low-level is a subsumption style reactive module that implements motion commands, such as go-straight and turn-left. To ensure that the robot moves around safely, the reactive modules implements a collision avoidance behavior that detects obstacles and navigate around them. ARK project illustrates how vision can be used to navigate a robot in an unstructured environment; thereby showing that vision is indeed a viable sensor for physical robots.

RHINO and MINERVA, two recent examples of robots that combine deliberation and

reactivity, have garnered kudos. Both have gained popularity as museum tour-guides. RHINO conducted tours at Deutsches Museum Bonn (Germeny) in mid-1997 (Burgard et al.1999). It successfully interacted with visitors, planned tours, and navigated at high-speeds through dense crowds. Its control architecture consists of two modules: (1) a high-level module that uses symbolic logic to perform planning, and (2) a low-level module that takes care of the reactive needs of the robot (i.e., sensing, motor-skills, and low-level behaviors). The reactive module uses probabilistic techniques to perform localization (determining the current position of the robot in the world) and mapping (the current location of the obstacles).[12] GOLEX, or GOLOG executor (Hdhnel, Burgard and Lakemeyer1998; Lespérance et al.1997), connects the two modules. Similar to RAP, GOLEX can expand each primitive action of a linear plan returned by GOLOG into a pre-specified sequence of commands to be executed over the reactive module. GOLEX can also expand GOLOG actions into pre-specified conditional plans, and it has limited capability of handling such plans. It also monitors the execution of these plans, and can request the top-level module to replan upon failure.

MINERVA, which conducted tours at Smithosonian's National Museum of American History in Washington (USA), is more capable than RHINO (Thrun et al.2000). Unlike RHINO, it can learn maps from scratch, and within these maps, it uses a *coastal planner* algorithm for path-planning. Such paths take into account the amount of information the robot is expected to receive at different location in the environment, which helps in localization. In addition to occupancy maps, it uses ceiling-mosaic maps to perform

[12]RHINO's uses modified *Markov Localization* method to find the current pose of the robot, and uses *occupancy grid algorithm* to find the locations of nearby obstacles.

Figure 13: Minerva is a very successful museum tour guide.

localization. Minerva uses facial expression, gaze direction, and voice features to interact with people. This interaction depends upon, and conveys, its current "emotional state," or mood, which ranges from happy to angry. A four state stochastic finite state automaton controls its mood by taking into account several external factors, such as proximity of people and whether people are in the way. In MINERVA, the high-level controller is developed using RPL, and is able to compose tours on-the-fly (for example, it can choose not to visit all the exhibits if time is running out, say, because it spent more time than expected on one exhibit). GOLEX reactively executes the plans returned from the high-level controller.

Funge combines deliberation and reactivity to construct quasi-intelligent autonomous virtual characters (Funge, Tu and Terzopoulos1999). These characters inhabit complex virtual worlds, such as the undersea world of Tu's fishes and the prehistoric world that is inhabited by a Tyrannosaurus and Velociprators, and perform high-level tasks using their deliberative/cognitive abilities. For example, in one of the scenarios, the merman Duffy

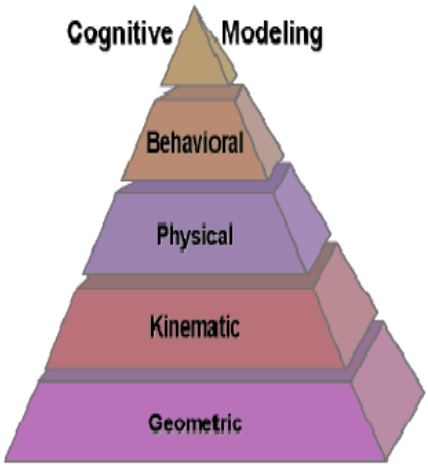


Figure 14: Funge *et al.* showed that deliberative/cognitive controller on top of an ethologically-inspired behavior-based substrate can yield powerful high-level controllers.

evades a shark using his superior intelligence even though the shark is faster and stronger than him.

Here, the deliberative module manages the knowledge that a character possesses, such as the position of the nearest rock, and it uses this knowledge to plan. It uses interval valued epistemic fluents to represent the uncertainty about this knowledge, and when this uncertainty increases beyond a certain threshold, it can plan to (1) decrease the uncertainty through sensing and (2) accomplish the task using the newly acquired knowledge. The controller can automatically initiate re-planning when the current knowledge becomes out-dated. Unlike the hybrid architectures discussed so far, here the deliberative module constructs *partial* plans (it only plans a limited number of moves in the future). Funge proposes Cognitive Modeling Language (CML) and uses it to implement the deliberative module. CML can be seen as Golog on steroids; it provides Golog-like syntax, structures, and facilities. The only difference is that it compiles the high-level controller into C lan-

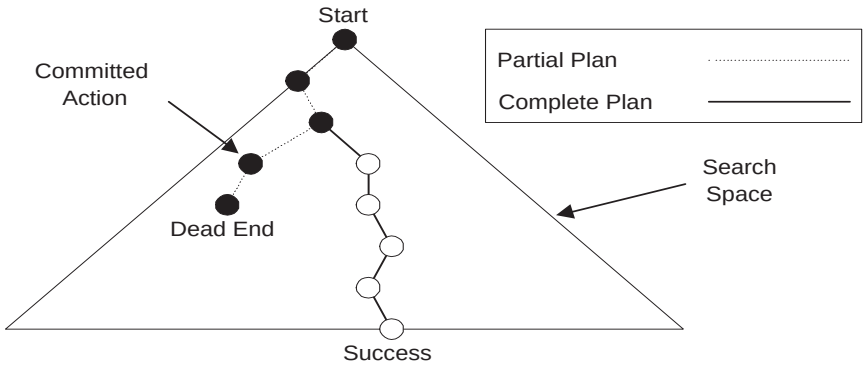


Figure 15: Hybrid architectures discussed so far, with the exception of Funge's work, share an important feature: these formulate the entire plan before beginning execution (of course these can re-plan when an action fails). One can not guarantee the correctness of a partial plan; committing to a partial plan excludes backtracking, so the planner might never find a valid plan. A robot might be worse off following a partial plan (as it becomes available) than waiting for the entire plan. One can also argue that longer duration plans degrades the real-time performance, and that these are rarely useful in a dynamic environment.

guage. The reactive module implements high-level behaviors using the scheme proposed by Tu (Tu and Terzopoulos1994). For example, the reactive module of Duffy implements behaviors `following` and `evading`. The reactive module is competent on its own and prevents the character from doing anything stupid in the absence of the advice from the deliberative module; an adviser-student relationship exists between the deliberative and reactive modules.

Funge's implementation is geared towards graphics and animation applications and can not be used to control a physical robot: 1) it does not use real-time control structures, and 2) the interface between the deliberative and reactive modules is ill-defined—it appears as if deliberative activity is embedded within the reactive control-loop[13]. It has nevertheless shown how a deliberative/cognitive module on top of an ethologically-inspired

[13]These are limitations of the implementation, and not of the theory itself.

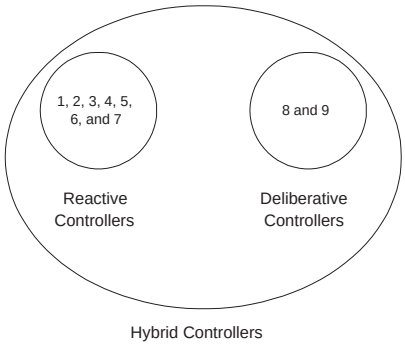


Figure 16: Hybrid controllers can meet all the requirements for auonomous robots that were put forward in Section \ref{sec:autonomous-robots}.

behavior-based substrate can yield powerful high-level controllers for autonomous characters (and robots). It also serves another purpose; it shows that simulated environments can prove useful for designing, implementing, and testing controllers for physical robots, as a designer can focus on the design of the controller itself rather than tackling with the hardware issues[14].

Hybrid control architectures seem to be more suitable for designing autonomous robots. It also appears that controllers that consist of heterogeneous modules fare better than homogenous hybrid controllers. Furthermore, the advisor-student relationship between the deliberative and reactive modules yields the best results.

4 Discussion

The success of Sony's Aibo robot has clearly shown that it is possible to construct a life-like robot using purely reactive (or behavior-based) control strategies. Reactive robot

[14]We are not saying that hardware issues are irrelevant. We are just saying that high-fidelity simulation environments are useful for designing and testing physical devices.

controllers model biological processes, such as low-level motor facilities, perception, attention, memory, and emotion, whose combination comprises a reactive module. The biological subfield of ethology provides the necessary guidance for designing reactive robot controllers. Although behavior-based robots can operate in the real world for long periods without human intervention, they are hard to formalize. Researchers have yet to devise a mathematical theory of reactive systems. Reactive systems exhibit emergent functionality that is hard to model formally, making it difficult to come up with a theory within which one can prove properties about them.

In the absence of such a theory, one can only study and test these systems empirically. One cannot prove that a reactive controller is "safe"; i.e., there are no guarantees that it will perform its function satisfactorily. Aside from their success in the domain of entertainment robotics, behavior-based systems are, therefore, not in widespread use.

Moreover, reactive systems are inherently limited because they are incapable of deliberative planning. Unlike reactive controllers, deliberative controllers are amenable to formal analysis, so we can prove a priori that a deliberative module is correct. However, experience has shown that deliberative controllers perform poorly in real-time situations.

Despite initial speculations in the domain of behavior-based robotics that purely reactive controllers are sufficient, it is now becoming clear that they are not, and that control architectures that combine reactive and deliberative strategies fare better. For example, behavior-based systems are inherently myopic; therefore, they tend to get stuck in local minima. A solution is to endow the robot with deliberative abilities such that it can reason about its goals and guide its reactive module, which provides the necessary interface between the deliberative module and the world, towards achieving them.

Existing deliberative modules are designed either as finite state machines, which represent pre-coded plans, or as a single monolithic planner. The plans that are computed offline are in general less robust to unanticipated situations; however, they can boost the performance of an agent, as now the agent need not waste any time computing the plan. Perhaps a better approach is to combine these two strategies within a single framework. Here, the agent will have access to a suite of pre-coded plans to handle the frequent situations encountered by the agent. The agent will also have a planning module that can jump in whenever none of the pre-coded plans can handle the current situation satisfactorily.

The interface between the two modules is a critical component that determines the overall performance of the controller. It must allow the deliberative module to advise the reactive module without affecting the real-time properties of the agent. Reactivity and deliberation have different characteristics and hence require different computational structures, so it is not surprising that heterogeneous hybrid architectures fare better than those that use the same computational mechanism for reactivity and deliberation.

Thus, our hypothesis is as follows: A heterogeneous hybrid architecture that implements an advisor-client relationship between the deliberative and reactive modules is well suited for designing intelligent controllers for autonomous robotic agents.

References

Agre, P. and Chapman, D. (1987). Pengi: An implementation of a theory of activity. In *Proceedings of American Association of Artificial Intelligence*, San Mateo, CA. Morgan Kaufmann.

Ambros-Ingerson, J. and Steel, S. (1988). Integrating planning execution and monitoring. In *In Proceedings of the seventh national conference on artificial intelligence (AAAI 88)*.

Arkin, R. (1988). Homeostatic control for a mobile robot. dynamic replanning in hazardous environments. In *Proceedings SPIE Conference on Mobile Robots*.

Arkin, R. (1990). Integrating behavioural, perceptual and world knowledge in reactive navigation. *Journal of robotics and autonomous systems(1-2), June 1990*, 6.

Arkin, R. (1992). AuRA: A hybrid reactive/hierarchical robot architecture. In *Proc. of the IEEE Int. Conf. on Robotics and Automation, Workshop on Architecture for Intelligent Control Systems*.

Arkin, R. (1998). *Behavior-Based Robotics*. The MIT Press, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Arkin, R., Fujita, M., and Takagi T., Hasegawa, R. (2001). Ethological modeling and architecture for an entertainment robot.

Blumberg, B. M. (1994). Action selection in hamsterdam: Lessons from ethology. In *Proceedings of the 3rd International Conference on the Simulation of Adaptive Behaviour*, Brighton. The MIT Press, Cambridge, MA.

Blumberg, B. M. (1997). *Old Tricks, New Dogs: Ethology and Interactive Creatures*. PhD thesis, Massachusetts Institute of Technology.

Braitenberg, V. (1984). *Vehicles*. MIT Press, Cambridge MA.

Brooks, R. (1986a). Achieving Artificial Intelligence through building robots. AI Memo 899, MIT, Cambridge, MA.

Brooks, R. (1986b). A robust layered control system for a mobile robot. *IEEE Journal of Robotics & Automation, March 1986, Also, MIT AIM 864, Sept.1985 - draft form*, RA-2(1).

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

Brooks, R. (1990a). The behavior language; user's guide. Technical Report A. I. MEMO 1227, Massachusetts Institute of Technology, A.I. Lab., Cambridge, Massachusetts.

Brooks, R. (1990b). A robot that walks: Emergent behaviors from a carefully evolved network. In Winston, P. H. and Shellard, S. A., editors, *Artificial Intelligence at MIT, Expanding Frontiers*. MIT Press, Cambridge, Massachusets.

Burgard, W., Cremers, A. B., Fox, D., Hahnel, D., Lakemeyer, G., Schulz, D., Steiner, W., and Thrun, S. (1999). Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, 114(1-2).

Connell, J. (1992). SSS: A hybrid architecture applied to robot navigation. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*.

Dickinson, S., Stevenson, S., Amdur, E., Tsotsos, J., and Olsson, L. (1993). Integrating task-directed planning with reactive object recognition. *SPIE Intelligent Robotics and Computer Vision XII*, 2055:212–224.

Endo, Y. and Arkin, R. (2000). Implementing tolman's schematic sowbug: Behavior-based robotics in 1930's. Georgia Tech.

Fikes, R. and Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 5(2).

Firby, R. (1989). *Adaptive Execution in Complex Dynamics Worlds*. PhD thesis, Yale University.

Firby, R. (1992). Building symbolic primitives with continuous control routines. In *Proceedings of the First International Conference on AI Planning Systems*.

Firby, R. (1994). Task networks for controlling continuous processes. In *Artificial Intelligence Planning Systems*.

Funge, J., Tu, X., and Terzopoulos, D. (1999). Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. In Rockwood, A., editor, *Proceedings of the Conference on Computer Graphics (Siggraph99)*, N.Y. ACM Press.

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

Hdhnel, D., Burgard, W., and Lakemeyer, G. (1998). GOLEX—Bridging the gap between logic (GOLOG) and a real robot. In *German AI Conference*.

Jasiobedzki, P. (1993). Active image segmentation using a camera and a range finder. In *SPIE Proceedings. Application of Machine Vision and Artificia Intelligence*.

Laird, J. E. and Rosenbloom, P. S. (1990). Integrating execution,planning,learning in SOAR for external environments. In *AAAI-90, Proceedings of the 8th National Conference on AI*, volume 2.

Lespérance, Y., Reiter, R., Lin, F., and Scherl, R. (1997). GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31.

Lorenz, K. (1973). *Foundations of Ethology*. Springer-Verlag, New York.

Ludlow, A. (1976). The behavior of a model animal. *Behavior*, LVIII(1-2).

Minsky, M. (1985). *The Society of Mind*. Simon and Schuster, New York, NY.

Nilsson, N. (1984). Shakey the robot. Technical Report 323, Stanford Research International.

Payton, D. (1990). Internalized plans: A representation for action resources. *Int. Journal of Robotics Systems*, 6(1-2).

Perlin, K. and Goldberg, A. (1996). IMPROV: A system for scripting interactive actors in virtual worlds. In Rushmeier, H., editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series. ACM SIGGRAPH, Addison Wesley. held in New Orleans, Louisiana, 04-09 August 1996.

Reynolds, C. (1987). Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4). Description of Reynold's bottom-up system for the study of flocking behavior.

Thrun, S., Beetz, M., Bennewitz, M., Burgard, W., Cremers, A., Dellaert, F., Fox, D., ahnel, D., Rosenberg, C., Roy, N., Schulte, J., and Schulz, D. (2000). Probabilistic algorithms and the interactive museum tour-guide robot minerva. *International Journal of Robotics Research*.

Tolman, E. (1939). Prediction of vicarious trial and error by means of the schematic sowbug. *Psychological Review*, 46.

Tsotsos, J. (1995). On behaviorist intelligence and the scaling problem. *Artificial Intelligence*, 75.

Tsotsos, J. (1997). Intelligent control for perceptually attentive agents: The s* proposal. *Robotics and Autonomous Sytems*, 21:5–27.

Tsotsos, J., Verghese, G., Dickinson, S., Jenkin, M., Jepson, A., Milios, E., Nuflo, F., Stevenson, S., Black, M., Metaxas, D., S. Culhane, Y. Y., and Mann, R. (1998). PLAYBOT: A visually-guided robot for physically disabled children. *Image and Vision Computing*, 16:275–292.

Tu, X. and Terzopoulos, D. (1994). Artificial fishes: Physics, locomotion, perception, behavior. In *Proceedings of SIGGRAPH '94*, Computer Graphics Proceedings, Annual Conference Series. ACM SIGGRAPH, ACM Press. ISBN 0-89791-667-0.

Tyrrell, T. (1993). *Computational Mechanisms for Action Selection*. PhD thesis, Centre for Cognitive Science, University of Edinburgh.

Walter, W. (1950). An imitation of life. *Scientific American*, 182(5).

Walter, W. (1951). A machine that learns. *Scientific American*, 185(2).

Walter, W. (1953). *The Living Brain*. W. W. Norton, New York, NY.