

Control engineering of autonomous cognitive vehicles - a practical tutorial

Sandor M Veres, Nick K Lincoln and Levente Molnar

Faculty of Engineering and the Environment, University of Southampton, Highfield, SO17 1BJ, UK

April 2011

Abstract. *An introduction is provided to artificial agent methodologies applicable to control engineering of autonomous vehicles and robots. The fundamentals that make a machine autonomous are considered: decision making that involves cognitive modelling the environment and forming data abstractions for symbolic processing and logic based reasoning. Capabilities such as navigation, path planning, tracking control and communications are treated as basic skills of cognitive agents. The ANSI standard of intelligent systems is used followed by the fundamental types of possible agent architectures for autonomous vehicles are presented, starting from reactive, through layered, to advanced architectures in terms of beliefs, goals and intentions. Cognitive capabilities of agents can fill in missing links between computer science results on discrete agents and engineering results of continuous world sensing, actuation and path planning. Design tools for “abstractions programming” are identified as needed to fill in the gap between logic based reasoning and sensing.*

Contents

1. Introduction
2. Artificial intelligence of autonomous robots
 - 2.1 Classification of mobile robots
 - 2.2 Autonomous vehicle/robot components
3. Software for autonomy and agents
 - 3.1 Definitions of agents
 - 3.1.1 Logic based architectures
 - 3.1.2 Behaviour-based and reactive architectures
 - 3.1.3 Situation calculus
 - 3.1.4 Beliefs-desire-intention (BDI) agents
 - 3.1.5 Layered architectures
 - 3.2 Markov decision processes for attention and action selection learning
 - 3.3 Agent oriented programming (AOP)
 - 3.4 Multi-agent decision processes
 - 3.5 Levels of autonomy
- 4 Abstractions for robotic agents
 - 4.1 Review of engineering abstractions
 - 4.2 Formal verification and certification
- 5 APPLIED AGENT ARCHITECTURES
 - 5.1 Layered architectures
 - 5.2 Hierarchical architectures
 - 5.3 Behavioural and reactive architectures
 - 5.4 BDI architectures
 - 5.5 Hybrid architectures

1 INTRODUCTION

Autonomous vehicles and robots need thorough testing or formal verification in case of safety critical systems. Decision making mechanism for switching among feedback control mechanisms, to control an autonomous system is invariably some kind of, perhaps naïve, “intelligent agent” implementation. The only methodology that has a promise to achieve these goals, and is currently to some degree available to engineers, is the area of “intelligent agent” theory and programming. An crucial part of reliability is the quality of cognitive capabilities of agents in terms of computer vision, audio processing, tactile, inertial, thermal, magnetic, radiation and other types of sensing. In many ways cognitive capabilities and abstractions of the world for modelling are the most important ingredients of an agent and rational decision making is an icing on the cake intelligence.

Despite the importance of cognitive capabilities for intelligence, the goal of this paper is to provide and introduction of decision making methodologies for cognitive systems and thereby encourage their use in practice. Our aim is to complement the cognitive science research under EuCog II with the top level decision making methods provided by agents that handle discrete events by logic such as temporal logic.

Until recently it was the privilege of biological beings: insects, mammals, birds, reptiles, and also plants, to command a system that is able to cope with an infinitely complex environment: to learn, develop their abilities and to ultimately fight with circumstances or with other beings. It has been unimaginable that artificial beings, i.e. man made machines, could have any comparable abilities to the well developed animals with regards to their autonomy in their environment. As the digital computer’s computational speed and capacity surpasses that of natural beings in some ways, similarly, the control capability of our machines can potentially surpass that of human beings in specific areas of knowledge. An existing example of this is the control avionics of a passenger aircraft that maintains the flight path of the plane autonomously.

This paper aims to guide the reader in the area cognitive software agent based intelligence and decision making with regards to how they achieve goals and how they act in face of sudden developments in thee environment. We start with ingredients of autonomy, programming techniques, definition of agents, followed by h the all important cognitive abstraction processes of agents and complete the paper with methods of formal verification that are relevant for industrial certification. The paper follows the following line of thoughts:

- i. Artificial intelligence of machines in general.
- ii. Standard architectures of intelligence.
- iii. Ingredients of autonomy.
- iv. Programming of agents: object oriented versus agent oriented.
- v. Definitions of the main types of agents for decision making.
- vi. Cognitive processes for agent decision making, formal verifiability and certification.

Agent-based approaches aim to enable autonomous vehicle or robots to determine and manage their health and adapt their behaviour accordingly. Ideally, the autonomous robot needs to acquire the fault detection and decision-making traits of the human supervisor or become even more capable in terms of precise planning and timely execution. Real-time system diagnostic and prognostics are capabilities that agents can integrate beside their ability to achieve mission goals. An agent framework provides intelligence in contingency management in military and civilian applications alike.

The next section provides an introduction into some of the basic concepts of intelligence, the ANSI reference architecture, autonomous vehicles and robots. Section 3 is a theoretical overview of agent architectures. The essence of reactive, behavioural, logic based, belief-desire-intention (BDI) and layered agent architectures are described. Section 3 concludes with some recent results on Markov decision

processes and agent oriented programming that can play a crucial role in cognitive processes. The whole of Section 4 is devoted to the cognitive processes of abstractions. Discrete abstractions of continuous systems are important for the creation of formally verifiable agent architectures that is the basis of *trustworthy autonomous vehicles*. A lot of acronyms are in use as listed here:

Acronyms used

AAV	autonomous aerial vehicle	KRONOS	realtime system model checker in RLTL
ADA	a statically typed, imperative, object-oriented and wide-spectrum programming language	LTL	linear temporal logic
AGV	autonomous ground vehicle	LIDAR	light detection and ranging
AI	Artificial intelligence	LTS	labelled transition systems
AMR	autonomous mobile robot	MA	manoeuvre automaton
ANSI	American National Standards Institute	MATLAB	a high level computer language
ASC	autonomous spacecraft	MCMAS	model checker for multi-agent systems
AV	autonomous vehicle	MDP	Markov decision process
AVDS	autonomous vehicle driver system	METATEM	an agent programming language
AVM	autonomous vehicle manager	NLP	natural language programming
AOP	agent oriented programming	NuSMV	model checker for timed automata
BDI	belief desire intention	OOP	object oriented programming
BG	behaviour generation	POMDP	partially observable Markov decision process
CAT	cognitive agent toolbox	RLTL	realtime temporal logic
CONGOLOG	a robot programming language	PRS	procedural reasoning system
CTL	computational tree logic	RA	remote agent
DAMN	distributed architecture for mobile navigation	RBF	radial basis function (neural network)
FDI	fault detection and isolation/identification	RNDF	route network definition file
CTL	computational tree logic	SBSP	situation based strategic positioning
DAMN	distributed architecture for mobile navigation	SMV	model checker in CTL
FDI	fault detection and isolation/identification	SOM	self-organising map (neural network)
FPGA	field programmable gate array	SP	sensory processing
FSM	finite state machine	SPA	sense process act (cycle)
GPS	global positioning system	SPIN	an LTL model checker for LTS
HYTECH	linear hybrid system verifier	TAXYS	extension of KRONOS with compiler
HS	hybrid system	T-REX	teleo-reactive eXecutive
INTERRUP	a vertical agent architecture	UAV	unmanned aerial vehicle
ITOCA	intelligent task oriented architecture	UPPAAL	model checker for timed automata
JACK	a Java based BDI agent programming environment	UGV	unmanned ground vehicle
JADE	a Java based agent programming language	VIS	model checker for interacting FSM of hardware systems
JASON	a Java based agent programming language	VJ	value judgement
		WM	world model

2 ARTIFICIAL INTELLIGENCE OF AUTONOMOUS ROBOTS

Why one would raise the issue of ‘intelligence’ in connection with autonomous vehicles? This is because one would like to have reliable vehicles operating in constantly changing and complex environments, for instance an urban environment, within a residential property, on the surface of a non terrestrial planet, or in deep space. In these applications unexpected events may happen and the vehicle must deal with these appropriately. Some of the decisions that a vehicle must make (especially in urban environments), can only be based on higher levels of knowledge of an educated adult familiar with the social habits, the legal consequences, dangers to health and life of an action taken – these are *the AI complex problems*. In brief: AI complex problems are those that need human levels of intelligence. At the initial stages of autonomous robotic research and development, as we are today, it is clearly important to avoid AI complex problems. This can be achieved by some *infrastructure* provided to autonomous robots that reduces their need for higher levels of intelligence. “Intelligence” is one of the hardest concepts to grasp and formalise. For the purpose of this tutorial we accept the following “definition” of intelligence [1]:

“Intelligence is the ability of a system to act appropriately in an uncertain environment, where an appropriate action is that which increases the probability of success, and success is the achievement of behavioural sub-goals that support the system’s ultimate goal.”

A general ANSI reference architecture [1] for intelligent systems contains the main processing units of Sensory Processing (SP), World Model (WM), Behaviour Generation (BG) and Value Judgement (VJ) with information flow as shown in Fig. 2.

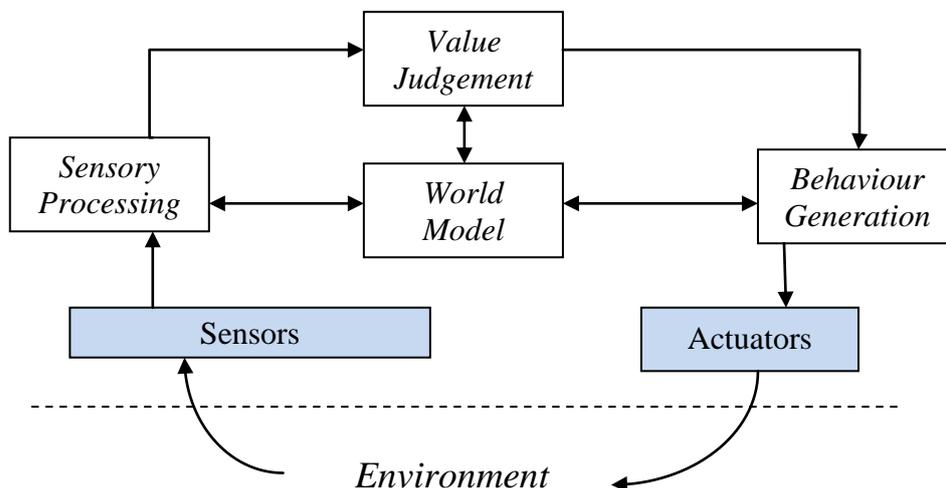


Figure 2. Reference architecture (ANSI) of intelligent systems [1] in the most generic form [2].

Some practical autonomous systems in operation today fall short of such fundamental requirements due to speed and computing capacity limitations. However, many of the current applications of agent based techniques to control systems have intelligence, in a similar sense. For instance a search and rescue AAV may have the goal to discover and report about trapped personnel in a hostile terrain. To achieve this the AAV may perform an initial rough survey of an area (SP,WM) and then select a set the “sub-goal” areas (based on VJ) for closer examination of some likely targets to take appropriate actions (BG) of signalling, relaying images and sending messages etc. This task can also be carried out by a group of AAVs. In this case intelligence emerges as the ability of the group of agents to collaboratively serve the ultimate goal of the system that is expressed in quality of performance, quality of output, energy efficiency and reliability. Table 1 points out parallel concepts in cognitive systems with those of the .

Table 1. Concepts of the ANSI reference architecture for intelligent systems as compared to their location in the human brain.

ANSI Intelligent Systems	Cognitive science	Brief description of associated digital signal processing
Sensory Processing	<i>Perception:</i> frontal lobe (attention), parietal lobe (tactile, heat, pain), occipital lobe (vision), temporal lobe(hearing)	Data and image filtering, <i>abstraction processes</i> by cognitive signal processing using clustering, self-organising artificial neural networks, reinforcement learning, etc., to recognize signal patterns, objects and events in the environment.
Behaviour Generation	<i>Abstract planning:</i> frontal lobe <i>Motor responses:</i> cerebellum	Behaviour generation can be more complex than running a simple feedback/feedforward control algorithm. It can also include <i>complex composition of switching and realtime adaptation</i> of controllers supervised by symbolic computation.
World Model	<i>Coordination of sensations:</i> middle cortex sections, <i>Maps:</i> parietal lobe (precuneus)	World model can include static geometric models, visual models, memory organisation, <i>abstract models of the present and past</i> , dynamical modelling and parameter estimation, etc.

	<i>Knowledge: frontal lobe</i>	
Value Judgement	<i>Decision making: frontal lobe</i> <i>Sensual assessment: middle cortex regions</i>	Value judgement can be much more than evaluating a fixed performance function. Goal oriented behaviour of agents means that <i>the agent selects its own performance functions</i> which can be constrained by behaviour rules and can require to make a compromise between partially conflicting goals.

2.1 Classification mobile robots

Autonomous mobile robots (AMRs) include autonomous unmanned aeronautical vehicles, autonomous underwater vehicles, autonomous unmanned ground vehicles and autonomous spacecraft. These latter four will be collectively called *autonomous vehicles* (AVs). AMRs are however a wider class than the vehicle classes mentioned. Autonomous vehicles are autonomous robots whose task repertoire is dominated but not exhausted by the goal of travelling along a route. AVs still can perform other tasks such as photographing or sampling the environment with robot arms, but they share the main capability to plan and follow tracks in their environment without collisions and serve some purpose by their travel that they need to “understand”. AMRs also include walking and climbing robots, manufacturing and humanoid robots that may be dominated by other tasks such as manipulation tasks, movements of objects, executing gardening, manufacturing or problem solving tasks. The boundaries are not rigid between these classes. Nevertheless, this paper will mainly focus on autonomous vehicles with the four subclasses of AGVs, AUVs, ASCs and AAVs. The development of these can however greatly benefit from computational and architectural achievements in the broader area of AMRs.

2.2 Autonomous vehicle/robot components

Table 2 clarifies the essential features that make a system autonomous. All five areas are important for autonomy but four of them are also needed for *remotely operated vehicles* (ROVs) and not specifically for autonomous vehicles only. Autonomous software is an indispensable component of an autonomous vehicle. There are however hardware components that support autonomy in their respective areas:

- (1) energy harvesting (<http://www.energyharvestingjournal.com>) [3, 4];
- (2) self repairing or self reproducing structures [5];
- (3) self reconfiguring computing hardware [6];
- (4) self diagnostic sensors and actuators [7].

3 SOFTWARE FOR AUTONOMY AND AGENTS

This section first present programming approaches to controlling autonomous vehicles followed by discussion on what agents are and some formal definitions of major agent classes. Markov decision processes are outlined due to their general use today. Finally agent oriented programming, multi-agent systems and levels of autonomy are discussed.

Software agents have been defined in various ways and it is difficult to pick a perfect definition, as they grasp different aspects of agents. Here is a list of the most common descriptions of agent features, i.e. what makes a software an “agent”:

- (1) *System theory definition*: agents are independent, situated, self contained systems with reflective and reflexive capacities (i.e. self-awareness).

- (2) *Software system definition*: software agents are extensions of objects (as in OOP) that can select their methods to be executed in response to inputs in order to achieve some goals.
- (3) *Robotics definitions*: agents are software that controls an autonomous robot to sense and make decisions on what action to take.
- (4) *Behaviour centric definition*: an agent is a software module that is capable of exhibiting reactive, proactive and social behaviour while using communication inputs and outputs.

The concept of a *physical agent* that has sensors and actuators is well summarised in [8]: “Intelligent agents are autonomous computational entities that can be viewed as perceiving their environment through sensors and acting upon their environment through effectors... ‘intelligent’ indicates that the agents pursue their goals and execute their tasks in such a way that they optimise some given performance measures. To say that agents are intelligent does not mean that they are omniscient or omnipotent, nor does it mean that they never fail. Rather, it means that they operate flexibly and rationally in a variety of environmental circumstances, given the information they have and their perceptual and effectual abilities.”

3.1 Definitions of agents

A formal description of agents [2], [9-12] is as follows.

Let the set of *states of the agent’s environment* be defined by $S = \{s_1, s_2, \dots\}$. The effector capability of an agent is represented by a set $A = \{a_1, a_2, \dots\}$ of *actions*. For any set X let X^* denote the set of all finite sequences of elements in X .

An agent can then be viewed as a function

$$\text{action} : S^* \rightarrow A$$

which maps finite sequences of environment states to actions. Intuitively speaking, an agent decides what action to perform on the basis of its history i.e. its experience to date. This basic model is called a *standard agent*. For any set X let $\wp(X)$ denote the set of subsets of X .

The non-deterministic behaviour of the environment can be modelled as a function

$$\text{env} : S \times A \rightarrow \wp(S)$$

which maps the current state of the environment and the agent’s action to a set of environment states that could result from performing the action. If all the sets in the range of *env* are singletons then the environment is called deterministic.

A *history* of the agent/environment interaction is a sequence:

$$h : s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_3 \xrightarrow{a_4} s_4 \dots$$

$\text{hist}(\text{agent}, \text{env})$ denotes the set of all histories of *agent* and the environment *env*. If some property ϕ holds for all these histories, this property is called an *invariant property* of the agent and the environment.

Two agents are *behaviourally equivalent* with respect to an environment, if their sets of histories are equal with respect to that environment. They are also called simply *behaviourally equivalent* if they are equivalent with respect to any environment.

An agent is *purely reactive* if its action only depends on the current state of the environment, so its action can be described by a function $\text{action} : S \rightarrow A$. This state-space based definition of reactive agents is a bit simplistic to be useful and the first architectural issue is to split the action mapping into perception and action:

$$\text{see} : S \rightarrow P, \quad \text{act} : P^* \rightarrow A$$

where P is a non-empty set of percepts and act maps sequences of percepts to actions. So far the agent's decision was modelled as a mapping from sequences of environment states to actions. Let's now introduce a set of agent states K . Then the next state is defined by a function $next : K \times P \rightarrow K$ and the action is decided by a new function $action : K \rightarrow A$ that maps the set of agent states into actions. State-based agent description assumes that an initial state exists.

Six types of agent architectures will be considered briefly:

- (1) reactive agents
- (2) behavioural agents
- (3) logic based agents
- (4) situation calculus
- (5) layered architectures
- (6) belief-desire-intention agents

There is no linear ordering of these approaches in terms of capabilities or speed of response or practical usefulness. The graph illustrates the connections and overlaps of these architectures.

Most approaches to autonomous vehicle engineering usually either fit one of these architectures or can be a combination of these. For instance there are approaches to BDI agents that rely, to various degrees, on logic inference. Situation calculus is a logic based approach but does not fit well the general framework of other logic based approaches to decision making by agents; behavioural agents can be reactive or not reactive depending on whether decisions are based on memory of the past. Adaptive fuzzy logic and neuro-fuzzy approaches however do not necessarily create a new class relative to the above approximate classification of agents (1)-(6). Also layered architectures can borrow methods of logical inference as well, can use behaviour definitions and can also exhibit reactive behaviours at various layers of abstractions of the environment.

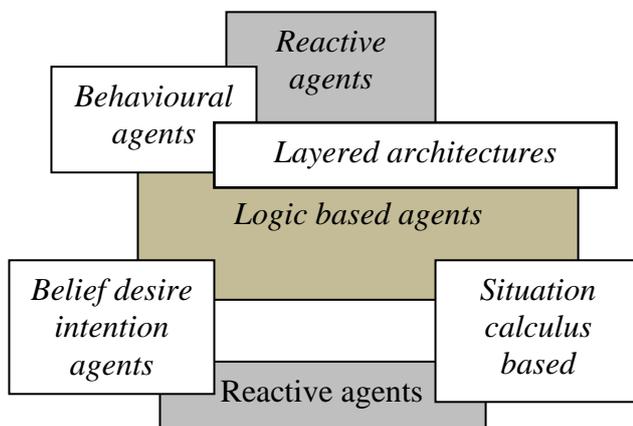


Fig. 3 A map of decision methods for autonomous agents. Most approaches to decision making in current autonomous vehicles can be placed on this map [2].

With regards to real-time performance the picture is not linearly ordered either as there are overlaps within these approaches. For instance it is not true that either of BDI agents or logic inference based approaches are the slowest, it all depends on the type of implementation and concurrency available.

An early analysis of autonomous planning and activity can be found in [13, 14]. The tutorial paper [15] calls the architecture “the backbone of a robotic system”. [16] is a systematic analysis of what is needed for cognitive agent architectures. Levels of autonomy in vehicles, that is a function of these agent architectures, is reviewed in [17]. In the rest of this section we introduce the fundamentals of each of these agent decision making mechanisms. By convention, decision making mechanisms will be called “agent architectures” as they are the essence of how agents operate.

3.1.1 Logic based architectures

Let L be a set of sentences of first-order temporal logic, and let $D = \wp(L)$ be the set of L databases, i.e. the set of sets of all L -formulae. The *internal state* of an agent is then an element of D . Notations $\Delta, \Delta_1, \Delta_2$ will be used for members of D . The logic-based agent’s perception function, *see*, remains unchanged as $see : S \rightarrow P$. The *next* function is an updating of D by the new experience via perception: $next : D \times P \rightarrow D$ which maps a database and a percept to a new database.

An agent’s decision making process is modelled through a set of *deduction rules* denoted by ρ . These are rules for logic inference by the agent. We write $\Delta \vdash_{\rho} \phi$ if the formula ϕ can be proven from an internal state $\Delta \in D$ of the agent, using only the deduction rules ρ .

We use the notation $\rho(D) \in \wp(L)$ for a set of deduced formulas after perception updating by *next*. Let $G \in D$ be the set of current goals. The agent’s action selection function

$$action : \rho(D) \times G \rightarrow A$$

is defined in terms of its deduction rules and goals. There is a variety of techniques in the literature regarding how *action* can be done. An essential simplification is that instead of *action* directly mapping into A , it can first map into a set of plans and to a pointer in the execution process of the plan. Let the set of possible ‘pointed’ plans be $\Pi \subseteq L^* \times I$ (where $I = \{1, 2, 3, \dots\}$ is the set of natural number pointers). The *planner*: $D \times G \rightarrow \Pi$

is a decision about the plan to be applied. Then $exec : \Pi \rightarrow A$ is an executor of a single step in the plan. Finally, goals require the goal update $g\text{-update} : \rho(D) \times G \rightarrow G$ function to be performed during each reasoning cycle to develop sub-goals and cancel unachievable goals.

Numerous pure logical approaches have been developed and applied to agents since the early 80s and throughout the 90s, for instance well known systems are CONGOLOG [18], [19] or METATEM [20]. A more practical approach in [21] use various performance measures, including utilities, costs and fitness, and probability theory for decision making of rational agents. [21] generalizes decision theory as performance measure theory and uncertainty theory. Rational agents with bounded resources look for approximate optimal decisions under bounded resources and uncertainty.

3.1.2 Behaviour-based and reactive architectures

Problems with the computational complexity of logic based architectures inspired the development of behaviour based reactive architectures. These approaches are also called *behavioural* or *situated* because their action is directly determined by the environmental situation via $action : S \rightarrow A$. Such agents are often perceived as simply reacting to the environment. Itself the method of choice for the action to be taken can vary. This section will describe a simple form of one of the best known behaviour-based reactive agent architectures, the *subsumption architecture* [10].

The function *see*, that defines the agent’s perceptual ability, is the same as before: $see : S \rightarrow P$. A *behaviour* is a pair (c, a) where $c \subseteq P$ is a set of percepts, also called the set of *conditions*, and $a \in A$ is an action. One says that a behaviour (c, a) *fires* in state $s \in S$ iff $see(s) \in c$. Let $R_b = \{ (c, a) \mid c \subseteq P, a \in A \}$

be the set of all behaviours. The agent's *behaviour rules* will be defined as some subset $B \subseteq R_b$. A binary *inhibition relation* $\triangleleft \subseteq B \times B$ is defined as a transitive, irreflexive and anti-symmetric relation. The inhibition relation \triangleleft is that is commonly referred to as the *subsumption hierarchy*, which is defined over the set of behaviours B .

The decision function *action* is defined through a set of behaviours and the inhibition relation that is defined over the set of behaviours, as follows. Let a percept $p \in P$ be given. The problem is to define $action(p) \in A$. Define the set of reaction behaviours as $fired(p) = \{ (c,a) \mid (c,a) \in B \ \& \ p \in c \}$. For each $(c,a) \in fired(p)$ it is found out whether there is a $(c',a') \in fired$ such that $(c',a') \triangleleft (c,a)$ that inhibits (c,a) . When a behaviour (c,a) is found that is not inhibited in any way then that action can be selected for $a = action(p)$. This a may not be unique in which case some ordering is introduced to prioritise.

Computational simplicity is a great strength of the subsumption architecture: the overall time complexity is not greater than $O(n^2)$ where n is the number of behaviours or percepts, whichever is greater. This allows strictly bounded and small decision time in realtime applications.

There are obvious advantages of reactive approaches: simplicity of behavioural rules, computational tractability and elegance. But there are some fundamental problems with these reactive architectures. As agents do not employ models of their environment, they must have sufficient information available to them in their local environment to determine their actions. As decisions only depend on current states of the environment, it is difficult to see how they could take into account non-local information in time and space. It is also difficult to see how purely reactive agents can be made to learn from their experience and to improve their performance over time when they do not have memory.

An interesting feature of reactive agent groups is that overall group behaviour *emerges* from the interactions of member behaviours. The relation-ship between individual behaviours, environment and overall behaviour is difficult to understand for a complex subsumption hierarchy. It is difficult to obtain an agent by a design procedure up to a given specification. Hence the subsumption architecture is often obtained by trial and error experimentation [10]. A similar approach is the *agent network architecture* [22, 23]. The related work on *situated automata* [24] is interesting in that it provides a method to compile agents, specified in a logical framework, into computationally efficient and very simple machines.

[25] presents a system for behaviour-based control of an autonomous underwater vehicle for the purpose of inspection of coral reefs. The behavioural scheme modifies the above subsumption by using fuzzy logic and utility function for behaviour selection. Behaviours are defined for guiding the robot along its intended course using sonar and vision-based approaches, for maintaining a constant height above the sea floor, and for avoiding obstacles.

For decision making during missions, [26] uses voting mechanisms. With regards to higher levels of autonomous behaviours, Bryson [27] presents different architectures and shows that the hierarchical organization of behaviours is one of the most advantageous ways of organizing them, since the combinatorial complexity of control is reduced. In these types of architectures, each behaviour has been modelled as a module that can communicate with others. [27] presents behaviour oriented design of agents for engineering agents with complex mission capabilities. [28] provides a systematic descriptions and an overview of decision making methods in behaviour-based robotics. [29] provides experimental comparison of hierarchical and subsumption software architectures for control of an autonomous underwater vehicle. [30] describes experiments to enhance the subsumption approach to behaviour-based control, as applied to a wheeled autonomous mobile robot. It uses a modified subsumption control approach, which reverses the behaviour priority ordering between layers. Modification of behaviours employing fuzzy logic for command fusion has also been used in [30]. [31] describes the development of an autonomous mobile robot that is a test bed for low level control experimentation, primarily for testing the robustness of behavioural and subsumption based control.

3.1.3 Situation calculus

The *situation calculus* is a second order logic with equality that allows for reasoning about actions and their effects. The world evolves from an initial situation due to primitive actions; possible world histories are represented by sequences of actions. Situation calculus distinguishes three sorts: actions, situations, and domain dependent objects. A special binary function symbol $do(a,s)$, with argument a for an action and s for a situation, is used to construct *histories of actions* that are called *situations*. For instance $do(move(2,3),S_0)$ or

$$do(pickup(Ball),do(move(2,3),S_0))$$

are situations starting from the initial situation S_0 .

Let the set of actions denoted by A and the set of situations by S . The do function is a mapping

$$do : A \times S \rightarrow S .$$

A binary predicate symbol $\sqsubset \subseteq S \times S$ defines an order on the set of situations. $s_1 \sqsubset s_2$ means that s_1 is a proper sub-history of s_2 . A binary predicate symbol $Poss \subseteq A \times S$. $Poss(a,s)$ means that it is possible to perform action a in situation s .

The domain independent *foundation axioms* of the situation calculus are:

$$do(a_1,s_1) = do(a_2,s_2) \Rightarrow a_1 = a_2 \wedge s_1 = s_2$$

$$\forall P: P(S_0) \wedge (\forall a \forall s)[P(s) \Rightarrow P(do(a,s))] \Rightarrow \forall s P(s)$$

$$\neg s \sqsubset S_0$$

$$s_1 \sqsubset do(a,s_2) \Leftrightarrow s_1 \sqsubset s_2$$

where the second axiom is in second order logic calculus because of the quantisation of predicate P .

A formula of the situation calculus is called *uniform in a situation s* iff it is first order, does not mention the predicates $Poss$ and \sqsubset , does not quantify over variables of situations, does not mention equality on situations and, finally, whenever it mentions a term in the situation arguments position of a fluent, then that term is s .

An action precondition axiom is a logic statement of the form

$$Poss(A(x_1, \dots, x_n), s) \quad P_A(x_1, \dots, x_n, s)$$

Where A is an n -ary action function symbol and $P_A(x_1, \dots, x_n, s)$ is a formula that is uniform in s and whose free variables are among x_1, \dots, x_n, s . The uniformity in s requirement on P_A is needed to ensure that the preconditions for the executability of the action A are determined only the current situation s and not by any other situation. For an autonomous vehicle we may for instance have

$$Poss(stopping(x), s) \Leftrightarrow moving(x, s)$$

A *successor state axiom* for an $(n+1)$ -ary relational fluent F is a sentence of the form

$$F(x_1, \dots, x_n, do(a, s)) \quad \Phi_F(x_1, \dots, x_n, a, s)$$

where Φ_F is a formula uniform in s , all of whose free variables are among x_1, \dots, x_n, a, s . A *successor state axiom* for an $(n+1)$ -ary functional fluent f is a sentence of the form

$$f(x_1, \dots, x_n, y, do(a, s)) \Leftrightarrow f_F(x_1, \dots, x_n, y, a, s)$$

where f_F is a formula uniform in s , all of whose free variables are among x_1, \dots, x_n, y, a, s .

A *basic action theory* D is the union of sets of axioms: $D = \Sigma \quad D_{ss} \quad D_{ap} \quad D_{una} \quad D_{so}$ where

Σ are the foundational axioms, D_{ss} is a set of successor state axioms for each fluent, D_{ap} is a set of action precondition axioms for each action function, D_{una} is the set of unique names axioms for action functions

and D_{so} is a set of sentences that are uniform in S_0 . A basic result about the situation calculus is that, under functional fluent consistency, a basic action theory D is satisfiable if $D_{una} \cup D_{so}$ is satisfiable. Further details of the properties of the situation calculus can be found in [32].

GOLOG [33] is a robot action programming language based on the situational calculus. [34] presents a mobile robot programming and planning language READYLOG, a GOLOG dialect, which was developed to support the decision making of robots acting in dynamic real-time domains. There is no or very limited use of any deduction rules to achieve realtime decision making. The formal framework of READYLOG is based on the situation calculus, with control structures of feedback loops and procedures that allow for decision-theoretic planning and account for a continuously changing world.

3.1.4 Beliefs-desire-intention (BDI) agents

These architectures have their roots in efforts trying to understand practical reasoning of how humans decide which momentary action to take to further their goals [11]. Practical reasoning involves two important phases: (1) what goals a human want to achieve and (2) how she is going to achieve them. The first is also called the *deliberation* process and the latter is called *means-end* reasoning. The functional architecture of BDI agents is outlined in the block diagram in Fig. 3. A formal description is provided as follows [9, 35].

Let Bel , Des and Int denote large abstract sets from which beliefs, desires and intentions can be taken. The state of a BDI agent is at any moment a triple (B,D,I) where $B \subseteq Bel$, $D \subseteq Des$ and $I \subseteq Int$.

An agent's *belief revision function* (*brf*) is a mapping from a belief set and percept into a new belief set:

$$brf: \wp(Bel) \times P \rightarrow \wp(Bel)$$

The options generation function (*options*) maps a set of beliefs and a set of intentions to a set of desires:

$$options: \wp(Bel) \times \wp(Int) \rightarrow \wp(Des)$$

The main function of *options* is means-end reasoning, and this must be *consistent* with beliefs and current intentions as well as *opportunistic* to recognise when environmental circumstances change advantageously.

The deliberation process, i.e. deciding what to do, is represented by the filter function

$$filter: \wp(Bel) \times \wp(Des) \times \wp(Int) \rightarrow \wp(Int)$$

which updates the agents intentions on the basis of its previously-held intentions and current beliefs and desires. It must *drop* intentions that are no longer achievable, *retain* intentions that are not yet achieved and it should adopt new intentions to achieve existing intentions or to exploit new opportunities. A constraint on *filter* is that it must satisfy $filter(B,D,I) \subseteq I$, i.e. current intentions must be either previously held intentions or newly adopted ones.

Finally the function *execute* is used to select an executable intention, one that corresponds to a directly executable action:

$$execute: \wp(Int) \rightarrow A$$

Intentions are often represented not only as sets but with some structure such as priorities. A well known implementations of the BDI architecture is the *procedural reasoning system* (PRS) in [35]. A large amount of work has been carried out to formalise the BDI model, a key paper being that by Rao [36],

which describes decision procedures for prepositional linear-time belief-desire-intention logics. [37] proposes a theoretical framework for teaming human and BDI intelligent agents for vehicles. It is highlighted that autonomous agent qualities are as much needed in supervised team-effort situations as in those of full autonomy.

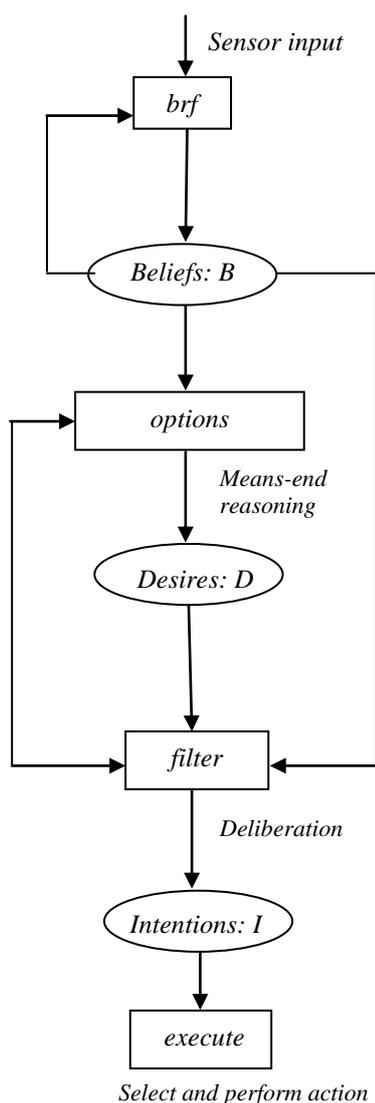


Fig. 4 Block diagram of belief-desire-intention (BDI) agent reasoning cycle [2].

3.1.5 Layered architectures

The natural idea of producing a model of the world in order to use that to plan ahead and then execute the plan, i.e. the sense-plan-act (SPA) cycle, is not without its limitations and problems. World modelling may be possible to simplify to such a degree that it may be performed suitably fast in realtime, but planning algorithms are difficult to run in realtime: by the time the plan is ready, the world may have changed, thus rendering its execution outdated.

From 1986 these difficulties of SPA were recognised: to achieve high speed and “intelligent” behaviour, the subsumption architecture was advocated [38],[10]. Behaviours based upon subsumption architecture were fast but did not store world model and memory, the scheme was instead based on the well emphasized principle of “the world is the model” and the “behaviours communicate through the world” via sensors and actuators. This architecture needed serious design effort, preplanning and proved to be fragile in face of sensor errors in practice. In response to this, more complex architectures appeared in a series of papers [39, 40], [35], [41, 42]. It was discovered that internal states are important but should be kept to a minimum. This

proved to be best achieved if abstractions were formed for sensing and actuation so that decision making did not make use of all the data in models. This led to *layered architectures*, where layers essentially represent abstraction levels. The term “reactive planning” has become “reactive execution” in the sense that the bottom layer was composed by automatic feedback-loop-based interaction with the environment. In a three layer architecture the top layer is the “*Deliberator*” or “*Planning player*” [43, 44], the middle layer is the “*Sequencer*” or “*Coordinator layer*” and the bottom layer is the “*Controller*” or “*Skills layer*” as illustrated in Fig. 4. A systematic overview of early layered architectures is presented in [45].

Apart from the layers of “deliberation”, “coordination” and “control”, the agent also needs the reverse process of “sensing, signal processing”, “abstraction” and “organisation of abstractions” that includes attention control for a purpose. This is represented in Fig. 4 with the parallel but opposite direction of information flow that includes sensing through signal processing, data fusion and abstraction organisation with attention control. The symbolically expressed, abstracted information about the world is processed by the *Deliberator* to make decisions on which goal to achieve and by what plan at a symbolic level.

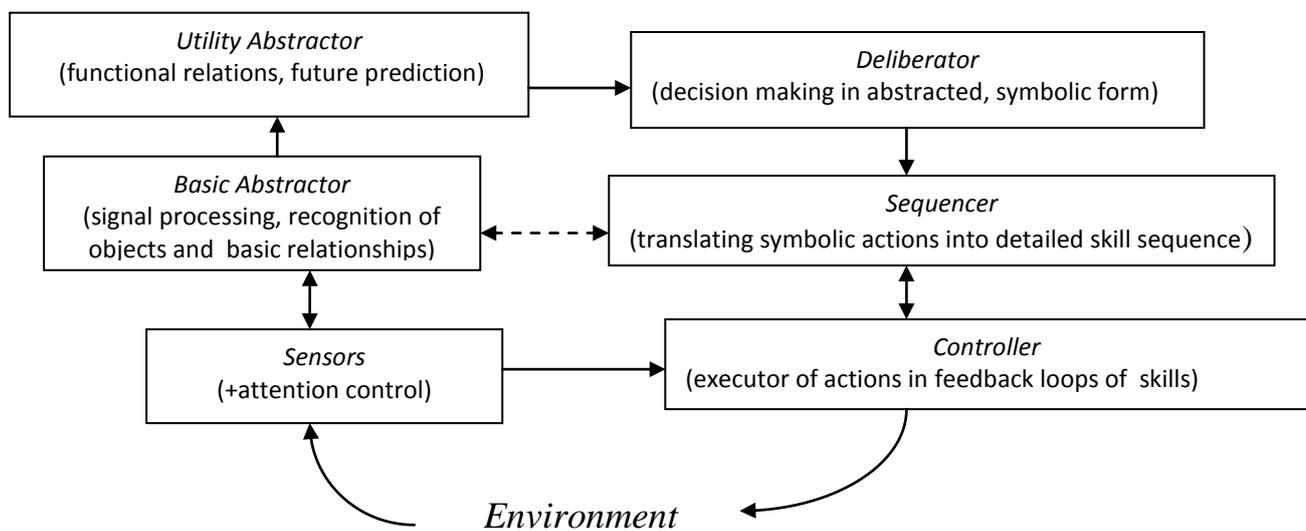


Fig. 5 An advanced layered architecture [45] that can have middle layers communicating: the *Sequencer* passes on to the *Abstractor* what it expects and the *Abstractor* passes an symbolic information to the *Sequencer* and thereby form a “symbolic control loop” at a higher level than the control loop of the Controller with sensor signals [2].

The *Sequencer* takes the symbolic plan and translates it into a sequence of skills to be exercised by the agent. The *Controller* executes the skills in a reactive manner by sensor-to-actuator feedback loops. An intermediate level symbolic control-loop is also possible to achieve by connecting the *Basic Abstractor* with Fig. 5. displays an advanced layered architecture that can have middle layers communicating. Layered architectures are the most versatile agent types and often include the layers or blocks of reactive, logic based and coordinated behaviours. Some of these are less formalised, and semantically clear, than the main agent types described in the previous three subsections. Important horizontal and vertically layered basic architectures are the “*TouringMachines*” as described in [19] and a well known example of vertical architectures is *INTERRUP*, as described in [46].

[47] presents an algorithm and its evaluation for collision avoidance and escape of autonomous vehicles operating in unstructured environments. This methodology mixes both reactive and deliberative components that provide the designers with an explicit means to design for robot certification. The traditional three layer architecture is extended to include a fourth ‘*scenario layer*’, where scripts describing specific responses are selected and parameterized in realtime. A local map is maintained using available sensor data, and adjacent objects are combined as they are observed. Objects have persistence and fade if not re-observed over time.

In common with behaviour based approaches, a reactive layer is maintained containing pre-defined immediate responses in extreme situations.

[48] advocates a *layered control architecture* for executing multi-vehicle team co-ordination algorithms with specific team behaviour. The control architecture consists of three layers of the team controller, vehicle supervisors and manoeuvre controllers.

3.1.6 Some general observations

Use of *neuro/fuzzy approaches* in most applications means that one of the above agent types relies on fuzzy logic instead of ordinary logic for decision making. Also *Markov decision processes* can be used in learning, perception, attention selection and decision making of agents that can include *reinforcement learning* algorithms. These can still be accommodated within the frameworks of classifications (1)-(6) as describe above. For their importance and popularity, the basics of MDPs are introduced in the next subsection.

There are fundamentally two kinds of *stability problems* in agent based control systems. One is that can occur while the agent executes a continuous feedback interaction with the environment (i.e. a skill execution) based on a programmed or learned feedback schema. The agent needs to detect an instability occurrence in realtime and its decision making must change the parameters of its feedback controller that can be for instance continuous or sampled in the sensing and control variables. A second type of instability can develop as a result of *cycles of decision making* by the agent. This is fundamentally a problem of the decision making mechanism of the agent and as such it needs to be avoided by careful design or by built-in “self-monitoring” of the agent actions.

3.2 Markov decision processes for attention and action selection learning

Markov decision processes (MDPs) can be used for focus selection in perception or can help an agent to learn what action to take under given circumstances in an environment [2]. A brief description of MDPs is as follows. Let S be a finite set of states of the environment and the agent. Let A be a finite set of actions that the agent can take. Let $P_a(s_2|s_1)$, $s_1, s_2 \in S$, denote the probability that state s_1 is followed by state s_2 if the agent takes action a . Let $R_a(s_1, s_2)$, $s_1, s_2, a \in A$ denote the immediate or expected reward of the agent if transition from s_1 to s_2 will actually happen as a result of taking action a .

The agent’s objective is to maximise some performance index that is a monotone function of rewards. A performance index can be evaluated for a policy $\pi: S \rightarrow A$ of the agent. For instance an often used discounted average performance index of a policy π is $\Pi(\pi) = \sum_{t=0}^{\infty} \alpha^t R_{\pi(s)}(s_t, s_{t+1})$ where $0 < \alpha < 1$ is a discounting factor. The agent can maximise its performance by some version of an iterative procedure of evaluating

$$\pi(s) = \arg \max_a \{ \sum_{q \in S} P_a(q|s) (R_a(s, q) + \alpha \Pi(q)) \}$$

where

$$\Pi(q) = \sum_{s \in S} P_{\pi(q)}(s|q) (R_{\pi(q)}(q, s) + \alpha \Pi(s)) .$$

This is a general approach to decision making that is less detailed than the agent frameworks described but can be powerfully used for skills and reinforcement learning of the agent. The literature of MDPs is vast. Here we briefly report on some interesting applications of MDPs that are relevant for autonomous vehicle controls.

[49] develops a hierarchical formulation of a *partially observable Markov decision processes* (POMDPs) for autonomous robot navigation that can be solved in real-time in dynamic environments. This method can be used for localization, planning and local obstacle avoidance. It decides each time step the actions the

robot should execute with consideration to motion and sensor uncertainty and enables fast learning. [50] presents a low level controller under communications and navigation uncertainty that has been applied to an autonomous robotic system using a WiFi-based POMDP. [51-53] present point-based approximate techniques for POMDPs to compute a policy based on a finite set of points collected in advance from the agent's belief space. The speed and efficiency of *point-based value iteration* and *randomized point-based value iteration* in POMDPs have been shown on robot games and in continuous navigation of mobile robots.

[54] describes how a robot can autonomously learn to execute navigation improvement plans, The problem is formalised as an MDP. The action selection function employs models of the robot's navigation improvement actions, which are autonomously acquired from experience using neural networks or regression tree learning algorithms. [55] uses dynamic Bayesian networks to represent stochastic actions in an MDP with a decision-tree representation of rewards. Versions of standard dynamic programming algorithms are developed that directly manipulate decision-tree representations of policies and value functions to reduce computational complexity via expected values. [56] proposes an abstraction technique for MDPs that allows approximately optimal solutions to be computed quickly. Abstractions are generated automatically using an intensional representation of the planning problem to determine the most relevant problem features. [57] presents a computational theory of developmental mental-architectures for artificial and natural systems, motivated by neuroscience, one of which is based on observation driven MDPs.

3.3 Agent oriented programming (AOP)

Table 3a lists some of the most popular agent oriented programming languages and development environments that engineers can use to develop autonomous systems. The first 6 can be interfaced to software that can handle sensors, actuators and signal processing. In addition, an engineer also needs hybrid system simulations and formal verification to thoroughly test her design. The Cognitive Agent Toolbox (CAT) is a purposeful integrated environment for engineers (rather than computer scientists). It integrates the capabilities of some other packages (MATLABTM/ Simulink/StateflowTM, MCMAS) and is also supported by *natural language programming* (sEnglish) that facilitates definition of abstractions for formal verification. The table highlights the fact that, to the best knowledge of the authors, CAT is the only integrated environment that provides facilities to study agent behaviour in the environment as a hybrid system that includes continuous world responses while also providing agent oriented programming features. Formal verification is particularly important for trustworthy industrial applications. Currently available robotics development software in Table 3b contrasts with Table 3a in that they do not use AOP for robot programming and most of them do not go beyond reactive/behavioural approaches to autonomy. Robotics programs focus instead on making low level realtime code development for real robots easier. Some of these, such as MS Robotics Studio, are low level programming environment that have the potential to be extended with AOP features.

From programming point of view agent oriented programming can also be considered as being at a level higher than object oriented programming. The similarity and difference between object oriented programming (OOP) and agent oriented programming (AOP) arises from that objects are generally passive and their methods are activated by external events. Objects encapsulate methods and functions of behaviour, but they do not encapsulate choice of action, i.e. "behaviour activation". Agents encapsulate behaviour and, as explained above, are capable of complex behaviour. All agent architecture types in subsections 3.3.1-3.3.5 can embrace the idea of AOP and provide an interactive programming framework for autonomous vehicle designers.

The key idea of AOP is to directly program agents in terms of mentalistic terms at high level, such as beliefs, desires and intentions and to enable them to communicate with each other in terms of some ontology language, see Shoham [58] about the fundamentals. For instance [59] proposes a programming method for

developing autonomous agents that behave intelligently in unpredictable environments, by extending their descriptions to realize large behavioural repertoires. An agent is described as a concurrent program which updates a finite set of data objects in realtime at regular intervals in an OOP language so that its description can be modular and reusable. The description of an agent specifies its property independent of hardware platforms.

Table 4a Some popular software packages for agent oriented programming (AOP) [2].

Software	Supports Agent Oriented Programming	Provides facility for hybrid system definitions	Supports embedded code generation	Provides facility for formal verification	Provides simulation of agent (s) in environment	Webpage
GOAL	+	-	+	-	-	http://mmi.tudelft.nl/~koen/goal.php
AgentSpeak Jason	+	-	+	-	-	www.jason.sf.net , [60]
PRS procedural reasoning system	+	-	+	-	-	www.ai.sri.com/~prs , [61]
3APL 3APL-M	+	-	+	-	-	http://www.cs.uu.nl/3apl/ , http://www.cs.uu.nl/3apl-m/ , [62]
Golog	+	-	+	-	-	http://www.cs.toronto.edu/cogrobot/main/systems/index.html
Jack	+	-	+	-	-	http://www.agent-software.com.au/products/jack/
CAT (MATLAB)	+ (Jason based)	+ MATLAB/ Stateflow	+	+ (MCMAS based)	+ (RTW, μ C/OS)	www.cognitive-agent-toolbox.com , [63]

[64] describes AOP based on a *logical theory of action*. The user provides a specification of the agents' basic actions (preconditions and effects) as well as relevant aspects of the environment, in an extended version of the situation calculus. Behaviours of the agents can be specified in terms of these actions in a programming language where one can refer to conditions in the environment. An interpreter automatically maintains the world model required to execute programs based on the specification. The theoretical framework of this programming approach allows agents to have incomplete knowledge of their environment. This theory also supports formal verification.

Execution monitoring for actions is also part of the AOP paradigm; [63] outlines a methodology borrowed from the already available fault detection and isolation (FDI) algorithms from the manufacturing industries.

[63, 65] describes the need for a next generation of system software architectures, where components are agents rather than objects “masquerading as agents”. The key qualities of future software systems need to be *autonomous, self-configuring, self-protecting, self-optimizing* together with the more advanced desirable features of “self-aware, environment-aware, self-monitoring and self-adjusting”. [63] introduces a comprehensive methodology of *agent oriented programming* that enables the implementation of an agent architecture tailored for a particular application in terms of speed and complexity. This programming environment also supports *formal verification* [66] and the use of natural language programming [67], [68] to aid the creation of agent abstractions and to assist a programmer team in the creation of a complex software system.

The agents produced in this system can read about new skills in English documents written by maintenance or research engineers after the commissioning of the vehicle system. Further information can be found in [69].

Table 4b. Some popular software robot programming languages without agent oriented programming [2].

Software	Supports Agent Oriented Programming	Provides facility for hybrid system definitions	Supports embedded code generation	Provides facility for formal verification	Provides simulation of agent(s) in environment	Webpage
KUKA	-	+	+	-	+	http://www.kuka-robotics.com/en/products/software/
MS Robotics Studio	-	+	+	-	+	http://www.microsoft.com/robotics/ http://msdn.microsoft.com/en-us/robotics/default
Actin	-	+	+	-	+	http://www.energid.com/products-actin.htm
CLARAty	-	+	+	-	+	http://claraty.jpl.nasa.gov/man/overview/index.php
CybelePro	-	+	+	-	+	http://products.i-a-i.com/wiki/index.php?title=Distributed_Control_Framework
iRSP	-	+	+	-	+	http://irsp.biz/irsp.html
Pyro	-	-	+	-	-	http://pyrorobotics.org/
RIK	-	-	+	-	-	http://www.inl.gov/adaptiverobotics/robotintelligencekernel/index.shtml

3.4 Multi-agent decision processes

A rich source of ideas and techniques originates for multi-agent decisions from teams in annual Robocup competitions. [70] provides a short introduction to Robocup principles and competition categories, followed by the clear complexity difference between decision making between chess playing and a football game. Typical multi-agent decision making problems, such as Robocup, have dynamic environment as opposed to the static environment of chess. State changes are realtime in football while discrete turn takings are used in chess, leaving more time to make a decisions. Sensor readings are purely symbolic in chess. Non-symbolic, continuous variables of positions and velocities are to be considered in football. Finally, decision making is clearly central in chess while it is distributed to individual agents in football.

In [70] the authors describe a multi-agent-decision process where each agent's action set is determined by a set of decision functions: (1) shooting evaluation (2) defensive evaluation. Then offensive and defensive decisions are taken by each individual agent. Joint defensive decisions result in formation by observation of team players and under limited communications. Finally there is a set of basic tactics available to the agents that give cause to *situation dependent decision making*: ball handling, pass, shoot, interception, movement of agents who do not control the ball. A combination of decision functions and the sensed situations defines each agent's realtime actions. More details of this approach can be found in [70].

[71] uses *situation based strategic positioning* (SBSP). To calculate SBSP an agent estimates the tactics and formation currently in use and its positioning and player type within that. This is adjusted by the ball position and velocity, the possible situation such as attack, defence, scoring opportunity, and also by the player type strategic characteristics. Strategic characteristics include potential for ball handling, admissible regions in the field, etc. There is a rich selection of *behaviour types* that the agent can select from, which depend on the tactics and strategic characteristics, and also on the agent's role in a formation, position of the

ball and the other players. A description can be found in [72]. A detailed analysis of multi-agent decision making is however beyond the scope of this paper, see references [73], [74].

A very original approach to programming is presented in [75] that advocates a formal grammar based method to self-assembly and self-organisation of robot teams with an overall system behaviour emerging from the concurrent system of member robot actions. [76] describes the design and implementation and programming of a sensor network system that can adapt to node failures and changes in node positions to support human actions in typical living and working environments such as buildings, offices, and homes.

[77] presents the results of simulations investigating a decentralised control algorithm able to maintain the coherence of a swarm of radio connected robots. Decision processes are solved while the radius of communication is considerably less than the global diameter of the swarm.

Programming of a network of agents can be based on theory presented in [22, 23]. [78] presents a game theoretical approach to multi-agent team coordination. In [48] a layered control architecture is programmed for executing multi-vehicle team co-ordination algorithms along with the specifications for team behaviour.

3.5 Levels of autonomy

The initial three *levels of autonomy* for autonomous vehicles can be described as follows[2].

Level 1 The vehicle is stabilised by feedback control and can autonomously track its self generated trajectory to given waypoints. There is a map and the vehicle is required to be able to locate itself on the map and follow a user specified track without human control.

Level 2 The navigation system of the vehicle is able to define intermediate waypoints by itself that permits the human supervisor to specify global targets without giving details. The vehicle is also capable of autonomous departure, arrival and collision avoidance.

Level 3 The human supervisor of the vehicle is allowed to define mission goals in terms of high abstraction levels and the vehicle has extensive mission-related knowledge and decision making capability onboard, while also having all the capabilities of levels 1-2 autonomy.

Level 3 autonomy is very broad and can be based on multi-layered and deliberative decision making architectures. Few of these are operational today. The first two autonomy levels have been demonstrated in numerous AAVs, AUVs, AGVs and ASC.

As the levels of autonomy increase, the dependence upon a human in the loop is reduced to that of a supervisory role; for highly mobile vehicles this clearly raises legal risks. Recently there have been civil aviation authorities that awarded *out-of-sight flight authorisation* [79] to AAVs. This included the development of a generic, safe and verifiable flight control architecture and algorithms for 3D path generation on the basis of a numerical terrain model and closed-loop strategy optimisation for mission planning and on-line re-planning. AUV certification is in its infancy and will probably develop fast in the near future. AGVs moving on tracks are already operating autonomously and reliably at many airports. Urban AGVs are however a great challenge as their low infrastructure solution can be AI complex.

Constrained path AGVs, such as autonomous buses, are being developed. Due to the possibility of unexpected events, these are likely to pose some challenges in the foreseeable future.

4 ABSTRACTIONS FOR ROBOTIC AGENTS

Abstractions of relationships in the sensed environment and abstractions of agent actions are discretisation of the continuous environment that are important for two major activities [2]:

- for the robot's decision making in terms of behaviour rules;
- for the design engineer of the robot to carry out formal verification in order to prove safe operation of the autonomous robot.

First abstraction processes will be considered for agent decision making and then verification tools will be reviewed that assume that abstractions have been formed. We will also comment on a tool available to carry out systematic a priori abstraction formation for robot activities using natural language programming.

Abstractions are needed by all agent architectures studied in this paper. Abstractions associate discrete symbols with *continuous phenomena*. On the other hand, decision making by agents is done in symbolic terms and decisions cannot be made without *discrete abstractions* of some sort. Logic based agents need abstractions to formulate symbolic representations of the changing environment and of the agent's situation in it. Behavioural and reactive architectures need abstractions to check whether some behaviour is to be committed. Layered architectures have built-in abstraction layers for higher level processing of information on their own actions and changes in the world. Deliberative agent architectures also need abstractions to update their beliefs, check whether they achieved some goal and also to symbolise actions to reason about them. In this section we report of research where sub-symbolic or symbolic abstractions are developed by agents themselves, or they are developed and *a priori programmed* by engineers.

4.1 Review of engineering abstractions

[80] addresses the question of how a mobile robot can *autonomously determine task-achieving sequences* of actions without using pre-supplied symbolic knowledge. Agent acquired perception-action-perception triplets are used to create: (i) a sub-symbolic representation of perceptual space, using the self-organising feature map (SOM) using neural networks, and (ii) a representation of perception-action-perception triplet associations. Three sub-symbolic action-planning mechanisms are presented to produce plans through unexplored regions of the perceptual space. One of these, based on the use of radial basis functions (RBFs), is capable of generalising over the perceptual space of the robot and pursuing plans through unexplored areas of the perceptual space of the robot. The question remains however, how this approach can be used to create shared understanding with humans and ensure certifiably safe behaviour of a robot for instance in a household environment. There is a certain *technological gap* between this approach and approaches taking the stance of verifiable agent behaviours using pre-programmed abstractions of the hybrid system of the robot-environment system.

[81, 82], provide abstractions of sensing and actions for AAV movements in natural environments. The hybrid model is mapped by abstractions to a *Kripke model of discretised worlds* that can be described by temporal logic formulae. This abstraction process can be both used for reasoning and verification of the AAV system in real environments.

An abstraction procedure of *robot task execution dynamics* is presented in [83] where a modelling approach is used to obtain a non-linear polynomial model of the robot's task in a real robot-environment-task system. Earlier, along the same line of research, the importance of *multi-resolution map-building*, that is clearly an abstraction process, had been analysed in [84] for autonomous mobile robots.

In [85] *a priori defined discrete abstractions* by triangulations are used to map out solutions for continuous motion planning and control problems. Discrete planning algorithms are linked to automatic generation of feedback control laws for robots with under-actuation constraints and control bounds. Kinematic robots with

velocity bounds and under-actuated unicycles, with forward and turning speed bounds, are covered. [86] presents an algorithm that translates the ‘abstract’ temporal logic specification of agent movements into a *hybrid automaton* where in each discrete mode the controller specifications for the continuous dynamics are imposed. [87] uses *hierarchical abstractions* for swarm robots in a 2D environment that is built on two levels of continuous and discrete abstractions. Discrete abstractions permit the use of linear temporal logic formulae to express control goals. The temporal logic formulae can express requirements of location visitations in a 2D area and where obstacles and agents are located, so that collision avoidance can be part of a goal formula. It is shown that there is an automated procedure that decides whether there are *control law solutions for each agent* in the swarm to satisfy a temporal logic goal.

[88] defines a *manoeuvre automaton* (MA) to abstract vehicle movements in terms of movement primitives. Motion plans are defined as concatenation of motion primitives such as *manoeuvre primitives* and *trim primitives* providing steady state movements. MA controllability is defined and constructive procedures are given to define motion plans. It is proven that under controllability, there exists a solution to a cost optimal control problem and the optimal motion plan has finite length in abstracted form of the MA language.

[89] reviews top-down approaches to *symbolic control of vehicles* in which logic tools are used on abstract models of vehicles and bottom up approaches to provide means by which such abstractions are implemented and are effective. As the authors admit the “two ends do not quite tie as yet”, and much work remains to be done in both directions to obtain generally applicable methods. The approach of *feedback encoding* [90] is used for symbolic control of nonlinear systems. The idea is to efficiently generate an abstracted control instruction sequence that is a plan for a sequence of continuous time actions of the autonomous vehicles achieving a desired movement in the continuous state space [90].

The corresponding perceptual abstractions of movements in state space, as obtained from sensor measurements, can be accomplished by *abstracted nonlinear observers* [63]. In [91] a symbolic description of a system is used to both design the layout of a recurrent neural network for control and also to write down the analytical expression of the rules for its training. This facilitates the definition of a *general procedure for control* of a complex dynamic systems using cost functions sampled along trajectories.

[92] shows that every incrementally globally asymptotically stable *nonlinear control system* is approximately equivalent (bisimilar) to a *symbolic model*. The approximation error is a design parameter in the construction of the symbolic model and can be rendered as small as desired. If the state space of the control system is bounded, then the symbolic model is finite.

Creating abstractions for agents can also be considered as *knowledge engineering for agents*. It is only for the simplest of agents that a single layer of abstractions from sensing to abstractions of perception and actions are sufficient. Multi-layers of abstractions constitute a knowledge base that represents what an agent can know and handle symbolically. Layers of abstractions can be built up by collecting similar relationships in one layer to form the next higher level layer. Highest level abstractions make concise communications and rational inference of an agent possible. Abstract communications and knowledge representation is only applicable if the agent is able to produce acceptable interpretations of higher level abstractions in any given situation. Ad hoc methods to define abstractions in terms of subroutines that check whether a proposition is true or wrong have been around since the start for robotics. However a systematic approach to creating levels of abstraction is through *natural language programming* (NLP) [93] that links human concepts with that of the robot. NLP is based on two major components:

- An *ontology* where all classes and properties can be derived back to basic types of a high level numerical language (*MATLAB, ADA, Python*, etc.) for signal processing of sensor and modelling data.
- A set of *sentences* in a natural language with the requirement that the meaning of each sentence is either defined by a sequence of other sentences or by code written in a high level language.

[94] illustrates the concepts used in sliding mode control of an autonomous spacecraft. [95] presents a complex system of knowledge representations, reasoning and planning tools that can provide some

interoperability between embedded agents to achieve high-level mission goals described by the operator. Ergonomic *knowledge engineering* solutions for agents can be found in [63, 67, 96-98].

[99] describes streams based *realtime abstraction processes* for logic based inference of autonomous systems. [99] develops some of the ideas of [100], [100] further by the use of *database query* technology and NLP techniques in sEnglish. [101] presents a complex system of knowledge representations, reasoning and planning tools with the desire to provide some interoperability between embedded agents to achieve high-level mission goals described by the operator.

The journal *special issue* [102] contains several papers where abstractions play a key role in decision making of vehicle agents. An important, role of abstraction processes is to provide a discretised model of the agent interacting in a real world environment [99]. The discretised model consists of logic propositions about the state of the world and the agent's actions. [103] proposes a fuzzy qualitative version of robot kinematics with the goal of bridging the gap between symbolic or qualitative functions and numerical sensing and control tasks.

[104, 105] presents a decision making strategy for autonomous multi-platform systems, wherein a number of platforms perform phased missions in order to achieve an overall mission objective.

4.2 Formal verification and certification

Formal verification tools use temporal logic statements (formed from basic propositions) to formalise transitions between discrete states of the environment that are triggered by actions, internal states and logic based reasoning of agents.

Autonomous vehicles can cause accidents with material damage and human injury or death, and as such can be considered as safety-critical. These systems must be *certified according to applicable standards* as adequately safe before they can be deployed. [106] reviews dangerous scenarios of autonomous vehicles and identifies the safety concerns that they raise. Safety-critical systems procured and operated by the UK Ministry of Defence must now be certified against the requirements given by Def Stan 00-56. [107] reviews some possible formal verification techniques to support *legal certification* and also mentions some abstraction and model checking approaches. [108] is concerned with the derivation of safety requirements after hazard identification and analysis using formal policy derivation methods. Table 5 presents some of the verification software available.

5 APPLIED AGENT ARCHITECTURES

The application of agent technology on a vehicle platform is not domain specific. Whilst a vehicle platform may be developed to perform within any medium, be it underwater, a surface craft, air vehicle or even a space vehicle, the agent technology is not concerned with this fact. Agent vehicle action, taken as a skill endowed to an agent system, that is executed by locally situated hardware systems (not part of any high level reasoning) is of great academic interest. For all vehicle types, innumerate papers exist on formation, position and attitude control, control reconfiguration, path planning, collision avoidance, as well as guidance and navigation issues. Taken by themselves such agent skills, as published, are generally shown as reactive agents: though these specialised and domain dependant skills may be interwoven into agent architectures to enable the development of a highly capable system. This section will consider the implemented agent architectures on this cross section of vehicle platforms.

5.1 Layered architectures

Functionally separate layered architectures exist for all vehicle platforms and are the predominant form of agent system applied to unmanned ground vehicles. [109] describes the sensing, planning, navigation, and actuation systems for the ‘Little Ben’ autonomous ground vehicle. The layered software architecture is organised hierarchically into a series of modules connected via interprocess communication messages. [110] provides a detailed description of a modified US Marine medium tactical vehicle replacement. The control architecture employs a layered design, where various software modules act as services and provide specific functions to the overall system. [111] describes Skynet, an autonomous vehicle with a Chevrolet Tahoe at its base, employing a multilayer perception and planning/control solution. [112] describes the architecture and implementation the Talos autonomous passenger vehicle, designed to navigate a road network using locally perceived whilst keeping to traffic rules. The employed solution resembles a layered control architecture, where the “navigator” is responsible for planning the high-level behaviour of the vehicle, the motion planner performs path generation and the controller is responsible for executing the motion. Layered control architectures for an AUV are detailed within [113, 114]. In these papers the control architecture is structured in three layers: the higher level planner/sequencer that acts as the deliberative layer in the overall architecture, the control execution layer and the functional reactive layer. This reactive layer takes care of the real-time issues related to the interactions with the environment. The middle control execution layer interacts between the upper and lower layers, supervising the accomplishment of the tasks. The three layer architecture employed for the SAUVIM underwater vehicle, consisting of the application layer, real-time layer and device layer is described within [115]. Here the application layer performs task management functions and to enables application specific purposes such as the user interface and supervisory control algorithms. The real-time layer comprises of a system configurator, a real-time operating system and some sub-task modules. The device layer connects directly to hardware and sends actuator command data and performs sensor data acquisition. The described control architecture claims to adopt the advantages of both traditional hierarchical and subsumption architectures. [116] proposes a three layer architecture for complex missions. A system consisting of a three-level software architecture (Strategic, Tactical and Execution levels) is described in [117]. These levels separate the control requirements into modularized functions, enclosing logically intense discrete state transitioning using asynchronously generated signals for control of the mission and real time synchronised controllers that stabilize the vehicle motion to callable commands.

5.2 Hierarchical architectures

Hierarchical organisation architectures have been investigated for application to underwater, ground and aerial agent vehicles. [29] presents experimental comparison of hierarchical and subsumption architectures for high-level mission control of AUVs. It was observed that both approaches results in behaviourally equivalent controllers. A hierarchical, hybrid, model-based architecture for mission control of a generic survey AUV is presented within [118, 119]. The control architecture is organized hierarchically, containing modules that are each modelled as a hybrid system. The lowest level of the hierarchy is formed by the underwater vehicle as the plant, and the vehicle controllers. These together serve as the plant for the higher level mission controller. The mission controller consists of a collection of high-level hybrid automata and is itself hierarchically decomposed into the behaviour controllers, operation controllers and the mission coordinators. [48] presents a hierarchical, layered control architecture for executing team-coordination algorithms within a multi-vehicle scenario. A three level hierarchy of team controller, vehicle supervisor and manoeuvre controller is implemented. [120] presents a hierarchical flight control system for unmanned aerial vehicles that executes high-level mission objectives by progressively decomposing them into machine-level commands. [121] develops a hierarchical cooperative guidance architecture for a multi-

missile system, that in the case of salvo attacks, ensures that the missiles achieve simultaneous target strike; distributed and reactive decision making is a characteristic of such coordinated control schemes.

5.3 Behavioural and reactive architectures

Behavioural and reactive approaches are prevalent throughout multi-agent robotic applications and have found implementation within a variety of cooperative and single robot vehicle platforms. [122] describes a comparative evaluation of four behaviour based control architectures for AUVs, including schema-based, subsumption, process description language and action selection dynamics. The general construct of a behavioural based system is that of composing elemental behaviours such as 'go to' and 'avoid'. The global behaviour is determined by the specific coordination method of the behaviour based control architecture under experimentation. Competitive methods such as the subsumption and action selection dynamics have only one behaviour active at a time. In terms of robustness, tuning time and modularity, competitive methods exhibit better performances. The disadvantage of the competitive methods is that command fusion is not allowed, and in the case of a situation of goal-seeking and obstacle avoidance the generated trajectory is non-optimal [122]. A space based behavioural application is considered as a solution for large scale satellite swarms applied to in-orbit assembly of large space structures within [123-125]. Here the behavioural elements are used to construct a three-dimensional vector with the orientation denoting the direction to be followed by the space vehicle, while the magnitude is expressing the strength of the response against other behaviour's commands. [126] describes a similar behavioural control architecture for the control of an AUV used to track a cable.

More complex behavioural and reactive approaches may be observed applied to autonomous ground and air vehicles, primarily concerning coordination within a multi-vehicle group. [127] investigates leader-follower formations of non-holonomic mobile ground robots wherein the AGV's control inputs are forced to satisfy suitable constraints restricting the set of leader possible paths and admissible positions of the follower, resulting in a purely reactive architecture. [128] presents a cooperative and decentralized 2D path-planning algorithm for a group of autonomous ground vehicles that guarantees collision free trajectories and may operate in real-time. Decisions on agent motion primitives are based on a set of behaviour rules defined; conflicts between agents are resolved by a cost-based negotiation process. [129] considers formation control for a group of unicycle-type mobile vehicles, which are capable of inter-robot communication, using a reactive agent architecture: the current situation of a vehicle determines its subsequent motion. [130] investigates cooperative control for multiple AAVs in the presence of known stationary obstacles and unknown enemy assets using a navigation function to plan the motion of the AAV team in a centralized fashion. The limited sensing zone of the AAVs is conditioned for by implementing an 'analytical switch' wherein the standard navigation function approach is extended to a multiple navigation strategy. In this system decision making is both reactive and behaviour rule based.

5.4 BDI architectures

The belief-desire-intention implementations, which cling to anthropomorphism applied to a computational entity, have been investigated for both underwater and space based applications. [131] proposes an AUV fuzzy neural BDI model, to facilitate the application of typical BDI agent models to the underwater realm. The authors highlight that a typical BDI agent model is not efficiently computable and the strict logic expression is not easily applicable to the uncertain AUV domain. The developed fuzzy neural network model consists of five layers: input (beliefs and desires), fuzzification, commitment, fuzzy intention, and defuzzification layer. Intentions are formed from beliefs and desires by the fuzzy commitment rules and neural network. Numerous agent methodologies for autonomous space vehicles have been presented within

simulated environments. [132, 133] present a BDI agent framework programmed using JACK, which is extended to permit numerical computation through a link to MATLAB. [134] presents satellite swarm planning, via communication and negotiation, while tasked with observation in an Earth orbit. Additional applications of a bespoke BDI agent language linked to a MATLAB environment and used for control, fault diagnostics and control reconfiguration of a spacecraft is presented within [135].

5.5 Hybrid architectures

Hybrid architectures evolve out of a desire to combine the advantageous traits of various systems architectures. Such architectures have received a lot of attention and this is evident within [136], wherein a comprehensive survey of twenty two control architectures for underwater vehicles shows that the hybrid control architecture is currently the most popular, merging the advantages of the deliberative and behavioural architectures while minimising their limitations. In some papers hybrid systems are presented where deliberative elements augment the fast reaction capabilities of behavioural elements that are driven directly from sensor inputs, with mission planning capabilities; such systems may be functionally similar to layered architectures. [136] presents a hybrid control architecture that, similar to [113, 114], is structured in three layers: the highest level is responsible for mission planning and re-planning; the intermediate layer activates the low-level behaviours and to passes parameters to them; while the lowest level layer is the reactive layer and contains the physical sensor and actuator interfaces. This similarity between control architectures is echoed within [137] where it is stated that “many control architectures recently proposed converge to a similar structure”. Of course, in the instance of hybrid systems, the similarity with another system is dependent upon where the hybridisation exists: it is argued here that hybrid systems should be referred to as those wherein a specific layer of the control architecture implements multiple disparate methods, rather than an architecture that may be broken down into interacting systems that are functionally different. Such hybrid layers are exemplified within the perception and planning systems of the Odin autonomous robotic vehicle where a hybrid deliberative/reactive architecture selects the appropriate behaviour based upon the current situation [138]. Additionally a hybrid coordination methodology between the disparate nature of competition and cooperation, trying to benefit from the advantages of both, is presented for an AUV within [139].

REFERENCES

1. Meystel, A.M. and J.S. Albus, *Intelligent Systems: Architecture, Design, and Control*, 2002: Wiley Interscience. **716**.
2. Veres, S.M., L. Molnar, N.K. Lincoln, and C.P. Morice, *Autonomous vehicle control systems - a review of decision making*. Proc. IMechE, Part I: J. Systems and Control Engineering, 2010. **225**(3): p. 155-195.
3. Ieropoulos, I., C. Melhuish, J. Greenman, and I. Horsfield, *EcoBot-II: An artificial agent with a natural metabolism*. Journal Advanced Robotic Systems 2005. **2**(4): p. 295-300.
4. Melhuish, C. and M. Kubo, *Collective Energy Distribution: Maintaining the Energy balance in Distributed Autonomous Robots*, in *7th International Symposium on Distributed Autonomous Robotic Systems*, . 2004: Toulouse, France. p. 261-270.
5. Hurlebaus, S. and L. Gaul, *Smart structure dynamics (a review)*. Mechanical Systems and Signal Processing 2006. **20**(2): p. 255-281
6. Tempesti, G., D. Mange, and A. Stauffer, *Self-Replicating and Self-Repairing Multicellular Automata*. Artificial Life, 1998. **4**(3): p. 259-282.
7. Feng, Z., Q. Wang, and K. Shida, *A review of self-validating sensor technology*. Sensor Review, 2007. **27**(1): p. 48-56.

8. Weiss, G., *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence.*, 1999, Cambridge: The MIT Press.
9. Wooldridge, M., *An Introduction to MultiAgent Systems*, 2002, Chichester: John Wiley & Sons.
10. Brooks, R.A., *Elephants don't play chess*. *Robotics and Autonomous Systems*, 1990. **6**(1&2): p. 3-15.
11. Veres, S.M. *Principles, architectures and trends in autonomous control*. in *Autonomous Agents in Control, 2005. The IEE Seminar on (Ref. No. 2005/10986)*. 2005, p. 1-9.
12. Wooldridge, M. and N. Jennings, *Intelligent Agents: Theory and Practice* Knowledge Engineering Review, 1995. **10**(2): p. 115-152.
13. Agre, P.E. and D. Chapman. *Pengi: An Implementation of a Theory of Activity*. in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 1987.
14. Agre, P.E. and D. Chapman, *What Are Plans for?* *Robotics and Autonomous Systems*, 1990. **6**: p. 17-34.
15. Coste-Maniere, E. and R. Simmons. *Architecture, the backbone of robotic systems*. in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*. 2000, p. 67-72 vol.1.
16. Langley, P., J.E. Laird, and S. Rogers, *Cognitive architectures: Research issues and challenges*. *Cognitive Systems Research*, 2009. **10**(2): p. 141-160.
17. Veres, S., *Mission Capable Autonomous Control Systems in the Oceans, in the Air and in Space, in Brain-Inspired Information Technology, Brain-IT 2007, SCI 266*. 2009, Springer Vrlg, Vol. SCI 266, p. 1-10.
18. Lespérance, Y., H. Levesque, F. Lin, D. Marcu, R. Reiter, and R. Scherl, *Foundations of a logical approach to agent programming*, in *Intelligent Agents II Agent Theories, Architectures, and Languages, Lecture Notes in Computer Science*. 1996, Springer, Vol. 1037, p. 331-346.
19. De Giacomo, G., Y. Lespérance, and H.J. Levesque, *ConGolog, a concurrent programming language based on the situation calculus*. *Artificial Intelligence*, 2000. **121**(1-2): p. 109-169.
20. Fisher, M., *A survey of concurrent MetateM — The language and its applications*, in *Proceedings of the First International Conference on Temporal Logic (ICTL)*, D.M. Gabbay and H.J. Ohlbach, Editors. 1994: Bonn, Germany. p. 480-505.
21. Eberbach, E., *Approximate reasoning in the algebra of bounded rational agents*. *International Journal of Approximate Reasoning*, 2008. **49**(2): p. 316-330.
22. Maes, P., *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, 1991: MIT Press. 200.
23. Maes, P., *The agent network architecture (ANA)*. *SIGART Bulletin*, 1991. **2**(4): p. 115-120.
24. Rosenschein, S.J. and L.P. Kaelbling, *Situated view of representation and control*, in *Computational theories of interaction and agency*, P. Agre and S.J. Rosenschein, Editors. 1996, MIT Press: Cambridge, Mass, p. 515-540.
25. Rosenblatt, J., S. Williams, and H. Durrant-Whyte, *A behavior-based architecture for autonomous underwater exploration*. *Information Sciences*, 2002. **145**(1-2): p. 69-87.
26. Ridao, P., J. Batlle, and M. Carreras, *O2CA2, a new object oriented control architecture for autonomy: the reactive layer*. *Control Engineering Practice*, 2002. **10**(8): p. 857-873.
27. Bryson, J.J., *Intelligence by design : principles of modularity and coordination for engineering complex adaptive agents*, in *Massachusetts Institute of Technology. Dept. of Electrical Engineering and Computer Science*. 2001, Massachusetts Institute of Technology: Boston.
28. Arkin, R.C., *Behavior-based robotics*. *Intelligent robots and autonomous agents*, 1998, Cambridge, Mass.: MIT Press. 490.
29. Byrnes, R.B., D.L. MacPherson, S.H. Kwak, R.B. McGhee, and M.L. Nelson. *An Experimental Comparison of Hierarchical and Subsumption Software Architectures for Control of an Autonomous Underwater Vehicle*. in *Symposium on Autonomous Underwater Vehicle Technology*. 1992, p. 135-141.

30. Flanagan, C., D. Toal, and M. Leyden, *Subsumption and fuzzy-logic, experiments in behavior-based control of mobile robots*. International Journal of Smart Engineering System Design, 2003. **5**(3): p. 161-175.
31. Leyden, M., D. Toal, and C. Flanagan. *An autonomous mobile robot built to develop and test behaviour based control strategies*. in *The 7th Mechatronics Forum International Conference*. 2000, Atlanta: Elsevier Science Ltd.
32. Das, S.K. and A.A. Reyes, *An approach to integrating HLA federations and genetic algorithms to support automatic design evaluation for multi-agent systems*. Simulation Practice and Theory, 2002. **9**(3-5): p. 167-192.
33. Levesque, H.J., R. Reiter, Y. Lesperance, L. Fangzhen, and R.B. Scherl, *Golog: a logic programming language for dynamic domains*. The Journal of Logic Programming, 1997. **31**: p. 59-83.
34. Ferrein, A. and G. Lakemeyer, *Logic-based robot control in highly dynamic domains*. Robotics and Autonomous Systems, 2008. **56**(11): p. 980-991.
35. Georgeff, M.P. and A.L. Lansky. *Reactive reasoning and planning*. in *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*. 1987, Seattle, WA, p. 677-682.
36. Rao, A., *Decision procedures for prepositional linear-time belief-desire-intention logics*, in *Intelligent Agents II Agent Theories, Architectures, and Languages, Lecture Notes in Computer Science*. 1996, Springer, Vol. 1037, p. 33-48.
37. Urlings, P., C. Sioutis, J. Tweedale, N. Ichalkaranje, and L. Jain, *A future framework for interfacing BDI agents in a real-time teaming environment*. Journal of Network and Computer Applications, 2005. **29**(2-3): p. 105-123.
38. Brooks, R., *A robust layered control system for a mobile robot*. Robotics and Automation, IEEE Journal of, 1986. **2**(1): p. 14-23.
39. Firby, R.J., *An Investigation Into Reactive Planning in Complex Domains*, in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 1987.
40. Firby, R.J., *Adaptive Execution in Complex Dynamic Domains*, in *Tech. Report YALEU/CSD/RR #672*. 1989, Yale University.
41. Gat, E., *Reliable Goal-directed Reactive Control for Real-world Autonomous Mobile Robots*. 1991, Virginia Polytechnic Institute and State University: Blacksburg, Virginia.
42. Gat, E., *On the Role of Stored Internal State in the Control of Autonomous Mobile Robots*. AI Magazine, 1993. **14**(1).
43. Gat, E. *ESL: a language for supporting robust plan execution in embedded autonomous agents*. in *Aerospace Conference, 1997. Proceedings., IEEE*. 1997, p. 319-324 vol.1.
44. Gat, E., *On Three-Layer Architectures*, in *Artificial intelligence and mobile robots : case studies of successful robot systems*, R.P.B. David Kortenkamp, and Robin Murphy., Editor. 1998, MIT Press: Cambridge, Mass., p. 195-210.
45. Müller, J., M. Pischel, and M. Thiel, *Modeling reactive behaviour in vertically layered agent architectures*, in *Intelligent Agents*. 1995, p. 261-276.
46. Bovio, E., D. Cecchi, and F. Baralli, *Autonomous underwater vehicles for scientific and naval operations*. Annual Reviews in Control, 2006. **30**(2): p. 117-130.
47. Evans, J., P. Patrón, B. Smith, and D. Lane, *Design and evaluation of a reactive and deliberative collision avoidance and escape architecture for autonomous robots*. Autonomous Robots, 2008. **24**(3): p. 247-266.
48. Sousa, J.B.d., K.H. Johansson, J. Silva, and A. Speranzon, *A verified hierarchical control architecture for co-ordinated multi-vehicle operations*. International Journal of Adaptive Control and Signal Processing, 2007. **21**(2-3): p. 159-188.
49. Foka, A. and P. Trahanias, *Real-time hierarchical POMDPs for autonomous robot navigation*. Robotics and Autonomous Systems, 2007. **55**(7): p. 561-571.

50. Sotelo, M.A., M. Ocaña, L.M. Bergasa, R. Flores, M. Marrón, and M.A. García, *Low level controller for a POMDP based on WiFi observations*. Robotics and Autonomous Systems, 2007. **55**(2): p. 132-145.
51. Pineau, J., G. Gordon, and S. Thrun, *Point-based value iteration: An anytime algorithm for POMDPs*, in *Proc. Int. Joint Conf. on Artificial Intelligence*. 2003: Acapulco, Mexico.
52. Spaan, M.T.J. and N. Vlassis, *Perseus: Randomized Point-based Value Iteration for POMDPs*. Journal of Artificial Intelligence Research, 2005. **24**: p. 195-220.
53. Thrun, S., W. Burgard, and D. Fox, *Probabilistic Robotics*, 2005: MIT Press.
54. Belker, T., M. Beetz, and A.B. Cremers, *Learning action models for the improved execution of navigation plans*. Robotics and Autonomous Systems, 2002. **38**(3-4): p. 137-148.
55. Boutilier, C., R. Dearden, and M. Goldszmidt, *Stochastic dynamic programming with factored representations*. Artificial Intelligence, 2000. **121**(1-2): p. 49-107.
56. Dearden, R. and C. Boutilier, *Abstraction and approximate decision-theoretic planning*. Artificial Intelligence, 1997. **89**(1-2): p. 219-283.
57. Weng, J., *On developmental mental architectures*. Neurocomputing, 2007. **70**(13-15): p. 2303-2323.
58. Shoham, Y., *Agent-oriented programming*. Artificial Intelligence, 1993. **60**(1): p. 51-92.
59. Oka, T., J. Tashiro, and K. Takase, *Object-oriented BeNet programming for data-focused bottom-up design of autonomous agents*. Robotics and Autonomous Systems, 1999. **28**: p. 127-139.
60. Bordini, R.H., J.F. Hübner, and M. Wooldridge, *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley Series in Agent Technology, ed. M. Wooldridge, 2007: John Wiley & Sons, Ltd.
61. Georgeff, M. and A. Lansky, *Procedural Knowledge*. Proceedings of the IEEE (Special Issue on Knowledge Representation), 1986. **74**: p. 1383--1398.
62. Dastani, M. *3APL Platform User Guide*. in *University of Utrecht*. Mehdi Dastani, Utrecht, p. 1-26.
63. Veres, S.M., L. Molnar, and N.K. Lincoln, *The Cognitive Agents Toolbox (CAT) for programming autonomous vehicles*, in *Southampton University e-Print Archives*, www.eprints.soton.ac.uk. 2009: Southampton. p. 1-30.
64. Lespérance, Y., H. Levesque, F. Lin, D. Marcu, R. Reiter, and R. Scherl, *Foundations of a logical approach to agent programming*, in *Intelligent Agents II Agent Theories, Architectures, and Languages*. 1996, p. 331-346.
65. Sterritt, R., C.A. Rouff, M.G. Hinchey, J.L. Rash, and W. Truszkowski, *Next generation system and software architectures: Challenges from future NASA exploration missions*. Science of Computer Programming, 2006. **61**(1): p. 48-57.
66. Molnar, L. and S.M. Veres. *Verification of Autonomous Underwater Vehicles Using Formal Logic*. in *European Control Conference 2009*. 2009, Budapest: European Control Council, p. 1-8.
67. Veres, S.M., *Natural Language Programming of Agents and Robotic Devices*, 2008, London: Sysbrain. pp 184.
68. Veres, S.M., *Authoring Tool for sEnglish Documents and Reader Agent*. 2008, SysBrain Ltd, www.system-english.com London.
69. Veres, S.M. and L. Molnar, *Documents for Intelligent Agents in English*, in *IASTED Conference on Artificial Intelligence and Applications*. 2010, IASTED: Innsbruck, Austria. p. 10.
70. Wang, X., C. Lu, and C. Gill, *FCS/nORB: A feedback control real-time scheduling service for embedded ORB middleware*. Microprocessors and Microsystems, 2008. **32**(8): p. 413-424.
71. Reis, L.P. and N. Lau, *FC Portugal Team Description: RoboCup 2000 Simulation League Champion*, in *RoboCup 2000*, P. Stone, T. Balch, and G. Kraetzschmar, Editors. 2002, Springer-Verlag: Berlin, Heidelberg, Vol. LNAI 2019, p. 29-40.
72. Meyer, J., R. Adolph, D. Stephan, A. Daniel, M. Seekamp, V. Weinert, and U. Visser, *Decision-Making and Tactical Behavior with Potential Fields*, in *RoboCup 2002*, G.A. Kaminka, P.U. Lima, and R. Rojas, Editors. 2003: Berlin Heidelberg, Vol. LNAI 2752, p. 304–311.
73. Kaminka, G.A., P.U. Lima, and R. Rojas, eds. *Robocup 2002*. Vol. LNAI 2752. 2003, Springer-Verlag: Berlin, Heidelberg.

74. Stone, P., T. Balch, and G. Kraetzschmar, eds. *Robocup 2000*. Vol. LNAI 2019. 2001, Springer-Verlag: Berlin, Heidelberg.
75. Klavins, E., *Programmable Self-Assembly - control of concurrent systems from bottom up* IEEE Control Systems Magazine, 2007(August): p. 43-56.
76. Kohno, M., M. Matsunaga, and Y. Anzai, *An adaptive sensor network system for complex environments*. Robotics and Autonomous Systems, 1999. **28**(2-3): p. 115-125.
77. Nembrini, J., A. Winfield, and C. Melhuish, *Minimalist coherent swarming of wireless networked autonomous robots*, in *SAB 2002, Proc. 7th International Conference on Simulation of Adaptive Behaviour* Edinburgh, UK.
78. Semsar-Kazerooni, E. and K. Khorasani, *Multi-agent team cooperation: A game theory approach*. Automatica, 2009. **45**(10): p. 2205-2213.
79. Fabiani, P., V. Fuertes, A. Piquereau, R. Mampey, and F. Teichteil-Königsbuch, *Autonomous flight and navigation of VTOL UAVs: from autonomy demonstrations to out-of-sight flights*. Aerospace Science and Technology, 2007. **11**(2-3): p. 183-193.
80. Pisokas, J. and U. Nehmzow, *Performance comparison of three subsymbolic action planners for mobile robots*. Robotics and Autonomous Systems, 2005. **51**(1): p. 55-67.
81. Jeyaraman, S., A. Tsourdos, R. Åbikowski, and B. White, *Kripke modelling approaches of a multiple robots system with minimalist communication: A formal approach of choice*. International Journal of Systems Science, 2006. **37**(6): p. 339 - 349.
82. Tsourdos, A., G. Sirigineedi, Rafał Zbikowski, and B.A. White, *Modelling and Verification of Multiple UAV Mission Using SMV*, in *Workshop on Formal Methods for Aerospace (FMA)*, M. Bujorianu and M. Fisher, Editors. 2010, EPTCS 20. p. 22-33.
83. Iglesias, R., U. Nehmzow, and S.A. Billings, *Model identification and model analysis in robot training*. Robot. Auton. Syst., 2008. **56**(12): p. 1061-1067.
84. Lemon, O. and U. Nehmzow, *The scientific status of mobile robotics: Multi-resolution mapbuilding as a case study*. Robotics and Autonomous Systems, 1998. **24**: p. 5-15.
85. Belta, C., V. Isler, and G.J. Pappas, *Discrete abstractions for robot motion planning and control in polygonal environments*. Robotics, IEEE Transactions on, 2005. **21**(5): p. 864-874.
86. Fainekos, G.E., S.G. Loizou, and G.J. Pappas. *Translating Temporal Logic to Controller Specifications*. in *Decision and Control, 2006 45th IEEE Conference on*. 2006, p. 899-904.
87. Kloetzer, M. and C. Belta, *Temporal Logic Planning and Control of Robotic Swarms by Hierarchical Abstractions*. Robotics, IEEE Transactions on, 2007. **23**(2): p. 320-330.
88. Frazzoli, E., M.A. Dahleh, and E. Feron, *Maneuver-based motion planning for nonlinear systems with symmetries*. Robotics, IEEE Transactions on, 2005. **21**(6): p. 1077-1091.
89. Belta, C., A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G.J. Pappas, *Symbolic planning and control of robot motion [Grand Challenges of Robotics]*. Robotics & Automation Magazine, IEEE, 2007. **14**(1): p. 61-70.
90. Bicchi, A., A. Marigo, and B. Piccoli, *Feedback encoding for efficient symbolic control of dynamical systems*. Automatic Control, IEEE Transactions on, 2006. **51**(6): p. 987-1002.
91. Apolloni, B., A. Piccolboni, and E. Sozio, *A hybrid symbolic subsymbolic controller for complex dynamic systems*. Neurocomputing, 2001. **37**(1-4): p. 127-163.
92. Pola, G., A. Girard, and P. Tabuada, *Approximately bisimilar symbolic models for nonlinear control systems*. Automatica, 2008. **44**(10): p. 2508-2516.
93. Veres, S.M., *Natural Language Programming of Agents and Robotic Devices*, 2008, London: SysBrain. 184.
94. Veres, S.M. and N.K. Lincoln, *Sliding Mode Control for Agents and Humans*, in *TAROS'08, Towards Autonomous Robotic Systems 2008*: Edinburgh.
95. Veres, S.M. and L. Molnar. *Documents for intelligent agents in English*. in *AIA'10, IASTED Conference on Artificial Intelligence Applications 2010*, Innsbruck: IASTED.

96. Veres, S.M., *Theoretical foundations of natural language programming and publishing for intelligent agents and robots*, in *TAROS 2010, Towards Autonomous Robotic Systems*. 2010: Plymouth, UK. p. 1-10.
97. Veres, S.M. and N.K. Lincoln. *Sliding Mode Control for Agents and Humans*. in *TAROS'08, Towards Autonomous Robotic Systems*. 2008, Edinburgh.
98. Veres, S.M. and L. Molnar. *Documents for Intelligent Agents in English*. in *IASTED Conference on Artificial Intelligence and Applications*. 2010, IASTED: Innsbruck, Austria, p. 10.
99. Dennis, L., M. Fisher, A. Lizitsa, N.K. Lincoln, and S.M. Veres, *Satellite Control Using Rational Agent Programming*, in *IEEE Intelligent Systems Magazine*. 2010, IEEE: US. p. 92 - 97
100. Heintz, F., J. Kvarnstr, and P. Doherty, *Bridging the sense-reasoning gap: DyKnow - Stream-based middleware for knowledge processing*. *Advanced Engineering Informatics*. **24**(1): p. 14-26.
101. Patrón, P., D.M. Lane, and Y.R. Petillot. *Interoperability of agent capabilities for autonomous knowledge acquisition and decision making in unmanned platforms*. in *OCEANS 2009-EUROPE, 2009. OCEANS '09*. 2009, Bremen, Germany, p. 1-8.
102. Veres, S.M., *Autonomous and adaptive control of vehicles in formation - Editorial to special issue*. *Int. J. Adapt. Control Signal Process* 2007. **21**(2-3): p. 93–94.
103. Liu, H., D.J. Brown, and G.M. Coghill, *Fuzzy Qualitative Robot Kinematics*. *IEEE Transactions on Fuzzy Systems*, 2008. **16**(3): p. 851-862
104. Prescott, D.R., R. Remenyte-Prescott, and J.D. Andrews, *A System Reliability Approach to Decision Making in Autonomous Multi-platform Systems Operating Phased Missions'*, in *RAMS 2008, Proceedings of Reliability, Availability and Maintainability Conference*. 2008: Las Vegas, USA.
105. Dickerson, C.E.a.M., D.N, *Architecture and Principles of Systems Engineering*. *Complex and Enterprise Systems Engineering*, 2009, New York: CRC / Auerbach Publications.
106. Alexander, R.D., M. Hall-May, and T.P. Kelly, *Certification of Autonomous Systems under UK Military Safety Standards* in *SEAS DTC - Systems Engineering for Autonomous Systems Defence Technology Centre* 2006, University of York: York.
107. Despotou, G., R. Alexander, and T. Kelly, *Addressing Challenges of Hazard Analysis in Systems of Systems*, in *Draft paper, Department of Computer Science*. 2007, University of York: York, UK.
108. Alexander, R., T. Kelly, and N. Herbert, *Deriving Safety Requirements for Autonomous Systems*, in *4th SEAS DTC Technical Conference*. 2009, UK MoD: Edinburgh.
109. Bohren, J., T. Foote, J. Keller, A. Kushleyev, D. Lee, A. Stewart, P. Vernaza, J. Derenick, J. Spletzer, and B. Satterfield, *Little Ben: The Ben Franklin Racing Team's Entry in the 2007 DARPA Urban Challenge*, in *The DARPA Urban Challenge, Springer Tracts in Advanced Robotics Series*. 2009, Springer: Berlin / Heidelberg, Vol. 56/2009, p. 231-255.
110. Chen, Y.-L., V. Sundareswaran, C. Anderson, A. Broggi, P. Grisleri, P. Porta, P. Zani, and J. Beck, *TerraMax: Team Oshkosh Urban Robot*, in *The DARPA Urban Challenge, Springer Tracts in Advanced Robotics Series*. 2009, Springer: Berlin / Heidelberg, Vol. 56/2009, p. 595-622.
111. Miller, I., M. Campbell, D. Huttenlocher, A. Nathan, F.-R. Kline, P. Moran, N. Zych, B. Schimpf, S. Lupashin, E. Garcia, J. Catlin, M. Kurdziel, and H. Fujishima, *Team Cornell's Skynet: Robust Perception and Planning in an Urban Environment*, in *The DARPA Urban Challenge, Springer Tracts in Advanced Robotics Series*. 2009, Springer: Berlin / Heidelberg, Vol. 56/2009, p. 257-304.
112. Leonard, J., J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, O. Koch, Y. Kuwata, D. Moore, E. Olson, S. Peters, J. Teo, R. Truax, M. Walter, D. Barrett, A. Epstein, K. Maheloni, K. Moyer, T. Jones, R. Buckley, M. Antone, R. Galejs, S. Krishnamurthy, and J. Williams, *A Perception-Driven Autonomous Urban Vehicle*, in *The DARPA Urban Challenge, Springer Tracts in Advanced Robotics Series*. 2009, Springer: Berlin / Heidelberg, Vol. 56/2009, p. 163-230.
113. Molnar, L., E. Omerdic, and D. Toal, *Guidance, navigation and control system for the Tethra unmanned underwater vehicle*. *International Journal of Control*, 2007. **80**(7): p. 1050 - 1076.

114. Molnar, L., *A Hybrid Control Architecture Development for the Guidance, Navigation and Control of the Tethra Prototype Submersible Vehicle*, in *Department of Electronics and Computer Engineering*. 2006, University of Limerick: Limerick.
115. Kim, T.W. and J. Yuh, *Development of a real-time control architecture for a semi-autonomous underwater vehicle for intervention missions*. *Control Engineering Practice*, 2004. **12**(12): p. 1521-1530.
116. Healey, A.J., A.M. Pascoal, and F.L. Pereira. *Autonomous Underwater Vehicles: An Application of Intelligent Control Technology*. in *American Control Conference*. 1995, Seattle, Washinton, p. 2943-2949.
117. Healey, A.J., D.B. Marco, and R.B. McGhee. *Autonomous Underwater Vehicle Control Coordination Using A Tri-Level Hybrid Software Architecture*. in *Proc. of the IEEE Int. Conf. on Robotics and Automation*. 1996, p. 2149-2159.
118. Tangirala, S., R. Kumar, S. Bhattacharyya, M. O'Connor, and L.E. Holloway, *Hybrid-model based hierarchical mission control architecture for autonomous underwater vehicles*, in *American Control Conference, 2005. Proceedings of the 2005*. 2005: Portland, OR. p. 668-673.
119. O'Connor, M., S. Tangirala, R. Kumar, S. Bhattacharyya, M. Sznaier, and L.E. Holloway, *A Bottom-Up Approach to Verification of Hybrid Model-Based Hierarchical Controllers with application to Underwater Vehicles*, in *Proceedings of the 2006 American Control Conference*. 2006: Minneapolis, Minnesota, USA.
120. Kim, H.J. and D.H. Shim, *A flight control system for aerial robots: algorithms and experiments*. *Control Engineering Practice*, 2003. **11**(12): p. 1389-1400.
121. Shiyu, Z. and Z. Rui, *Cooperative Guidance for Multimissile Salvo Attack*. *Chinese Journal of Aeronautics*, 2008. **21**(6): p. 533-539.
122. Carreras, M., J. Battle, P. Ridao, and G.N. Roberts. *An overview of behaviour based methods for AUV control*. in *5th IFAC Conference on Manoeuvring and Control of Marine Crafts*. 2000, Aalborg, Denmark: Elsevier.
123. Izzo, D. and L. Pettazzi. *Self-assembly of large structures in space using intersatellite Coulomb forces*. in *In 57th International Astronautical Congress*. 2006, Valencia, Spain.
124. Izzo, D. and L. Pettazzi, *Autonomous and Distributed Motion Planning for Satellite Swarm*. *Journal of Guidance Control and Dynamics*, 2007. **30**(2): p. 449-459.
125. Izzo, D., L. Pettazzi, and M. Ayre. *Mission Concept for Autonomous on Orbit Assembly of a Large Reflector in Space*. in *In 56th International Astronautical Congress*. 2005, Fukuoka, Japan.
126. Ortiz, A., J. Antich, and G. Oliver, *A PFM-based control architecture for a visually guided underwater cable tracker to achieve navigation in troublesome scenarios*. *Journal of maritime research: JMR*, 2005. **2**(1): p. 18.
127. Consolini, L., F. Morbidi, D. Prattichizzo, and M. Tosques, *Leader-follower formation control of nonholonomic mobile robots with input constraints*. *Automatica*, 2008. **44**(5): p. 1343-1349.
128. Purwin, O., R. D'Andrea, and J.-W. Lee, *Theory and implementation of path planning by negotiation for decentralized agents*. *Robotics and Autonomous Systems*, 2008. **56**(5): p. 422-436.
129. Do, K.D. and J. Pan, *Nonlinear formation control of unicycle-type mobile robots*. *Robotics and Autonomous Systems*, 2007. **55**(3): p. 191-204.
130. Chen, J., D.M. Dawson, M. Salah, and T. Burg, *Cooperative control of multiple vehicles with limited sensing*. *International Journal of Adaptive Control and Signal Processing*, 2007. **21**(2-3): p. 115-131.
131. Hai-bo, L., G. Guo-chang, S. Jing, and F. Yan, *AUV fuzzy neural BDI*. *Journal of Marine Science and Application*, 2005. **4**(3): p. 37-41.
132. Thanapalan, K.K.T. and S.M. Veres. *Agent Based Controller for Satellite Formation Flying*. in *Intelligent Sensors, Sensor Networks and Information Processing Conference, 2005. Proceedings of the 2005 International Conference on*. 2005, p. 385-389.
133. Veres, S.M. and J. Luo. *A class of BDI agent architectures for autonomous control*. in *Decision and Control, 2004. CDC. 43rd IEEE Conference on*. 2004, Paradise Islands, Bahamas, p. 4746-4751.

134. Bonnet, G. and C. Tessier, *Collaboration among a satellite swarm*, in *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. 2007, ACM: Honolulu, Hawaii. p. 1-8.
135. Dennis, L.A., M. Fisher, N.K. Lincoln, A. Lisitsa, and S.M. Veres, *Agent Based Approaches to Engineering Autonomous Space Software*, in *Proceedings FM-09 Workshop on Formal Methods for Aerospace*, M.L. Bujorianu and M. Fisher, Editors. 2009: Eindhoven, The Netherlands. p. 63-67.
136. Ridao, P., J. Batlle, J. Amat, and G.N. Roberts, *Recent trends in control architectures for autonomous underwater vehicles.*, in *International Journal of Systems Science*. 1999, Taylor & Francis Ltd. p. 1033-1056.
137. Ridao, P., J. Yuh, J. Batlle, and K. Sugihara. *On AUV control architecture*. in *2000 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2000: Taylor & Francis Ltd, p. 1033-1056.
138. Reinholtz, C., D. Hong, A. Wicks, A. Bacha, C. Bauman, R. Faruque, M. Fleming, C. Terwelp, T. Alberi, D. Anderson, S. Cacciola, P. Currier, A. Dalton, J. Farmer, J. Hurdus, S. Kimmel, P. King, A. Taylor, D. Van Covern, and M. Webster, *Odin: Team VictorTango's Entry in the DARPA Urban Challenge*, in *The DARPA Urban Challenge, Springer Tracts in Advanced Robotics Series*. 2009, Springer: Berlin / Heidelberg, Vol. 56/2009, p. 125-162.
139. Carreras, M., J. Batlle, and P. Ridao. *Hybrid coordination of reinforcement learning-based behaviors for AUV control*. in *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*. 2001, p. 1410-1415 vol.3.