Autonomous vehicle control systems – a review of decision making

S M Veres*, L Molnar, N K Lincoln, and **C P Morice**
School of Engineering Sciences, University of Southampton, Highfield, SO17 1BJ, UK

The manuscript was received on 17 May 2010 and was accepted after revision for publication on 11 August 2010.

DOI: 10.1177/2041304110394727

Abstract: A systematic review is provided on artificial agent methodologies applicable to control engineering of autonomous vehicles and robots. The paper focuses on some fundamentals that make a machine autonomous: decision making that involves modelling the environment and forming data abstractions for symbolic processing and logic-based reasoning. Most relevant capabilities such as navigation, autonomous path planning, path following control, and communications, that directly affect decision making, are treated as basic skills of agents. Although many autonomous vehicles have been engineered in the past without using the agent-oriented approach, most decision making onboard of vehicles is similar to or can be classified as some kind of agent architecture, even if in a naïve form. First the ANSI standard of intelligent systems is recalled then a summary of 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. The review identifies some missing links between computer science results on discrete agents and engineering results of continuous world sensing, actuation, and path planning. In this context design tools for 'abstractions programming' are identified as needed to fill in the gap between logic-based reasoning and sensing. Finally, research is reviewed on autonomous vehicles in water, on the ground, in the air, and in space with comments on their methods of decision making. One of the main conclusions of this review is that standardization of decision making through agent architectures is desirable for the future of intelligent vehicle developments and their legal certification.

Keywords: autonomous vehicles, artificial intelligence, intelligent agents, control systems

1 INTRODUCTION

Reliability and trustworthiness of autonomous systems need thorough testing or formal verification in case of safety-critical systems. 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. On the other hand any decision-making mechanism for switching among feedback control mechanisms, to control an autonomous vehicle, is invariably some kind of, perhaps naïve, 'agent' implementation. This paper provides a brief review of the agent methodologies for engineers that computer scientists can offer and also informs about some important missing gaps between engineering and computer science research efforts. The theory that computer scientist created for agents does not cover some fundamental issues for engineers, for instance how agents can handle variable sets of discrete abstractions from a continuous environment and also how formal verification of robot behaviour can be carried out. These are challenges that engineers face in the near future and this paper provides an introduction to the basics of agent-based control systems to assist that effort.

A keyword search in journals for autonomous control of vehicles results in many hundreds of journal papers between 1999 and 2009. Many of the engineering result of autonomous vehicles provide only low levels of autonomy, restricting the autonomous vehicle design problem only to navigation/tracking problems and pre-programmed reactive

**Corresponding author: School of Engineering Sciences, University of Southampton, Highfield, SO17 1BJ, UK.*
email: s.m.veres@soton.ac.uk

behaviours. A lot of research effort is devoted to the sub-problems of tracking control or navigation technology, without consideration of mission capable agent architectures that need sophisticated decision making. The number of papers using agent architectures for autonomous vehicles is a small fraction of the papers devoted to technical issues of autonomous vehicles. Hence the goal of this paper is to review higher levels of decision-making methodologies for autonomous vehicles and thereby encourage their use in practice. The current authors attempt a systematic approach to highlight achievements and challenges on agent architectures that determine intelligent decision making. The objective of this paper is to raise awareness of research issues related to decision making without completeness to research details.

A couple of decades ago 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 ultimately to fight with circumstances or with other beings. Until only recently it was 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 a hostile environment. Although we are still far from being able to produce a system as able as a fly or a monkey, it is now becoming increasingly clear that our prospects to engineer intelligent machines are not only good, but is already in progress. 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 review aims to help the development of advanced autonomous vehicles at higher levels of intelligence in their decision making with regards to how they achieve goals and how they act in face of sudden developments in their environment.

The paper is organized in a pyramid structure from top to base: starting from general concepts to publications on engineered autonomous vehicles (Fig. 1).

Most vehicle developments reviewed in sections 6–8 can be classified or recognized as being similar to a mainstream agent architecture described in section 3. Recent developments in software tools are mentioned for engineers to make agent programming easy. Section 4 identifies that an important

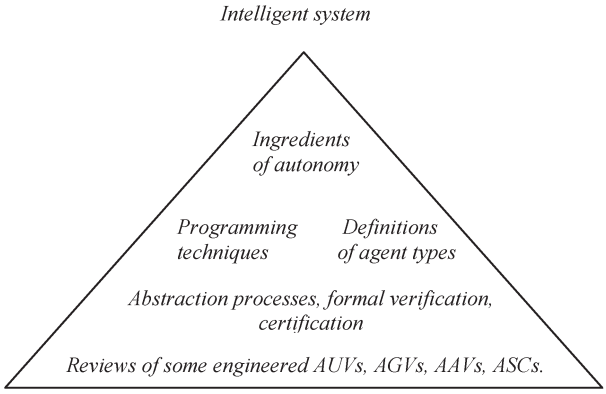

Fig. 1 Build up of this review in terms of areas considered

missing link between engineering and agents is the abstraction processes of detecting and recognizing change in the environment. Events, actions, objects, environment responses, and environment properties are abstractions we humans take for granted. This is not so for agents and recognition of important events and objects in the environment is a bottleneck for progress in the development of verifiable autonomous vehicles and robots. A real-time capability layer for creating abstractions from the continuous world to the discrete logic of agents is currently as yet underdeveloped.

There is consistent build up of the relevant concepts in this paper. The conceptual and logical groupings are:

(a) fundamentals of artificial intelligence of vehicles;
(b) reference architectures of intelligence;
(c) essential components of autonomy;
(d) programming: object-oriented and agent-oriented programming;
(e) formal definitions of the main types of agents for decision making;
(f) abstraction processes for agent decision making, formal verifiability, and certification;
(g) review of engineering developments of autonomous vehicles in the air, under water, on ground, and in space.

The standard agent-based approaches aim to enable autonomous vehicles to determine and manage their health and adapt their behaviour accordingly. Ideally, the autonomous vehicle needs to acquire the fault detection and decision-making traits of the human pilot or become even more capable in terms of precise planning and timely execution. Real-time vehicle system diagnostic and prognostics are capabilities that agents can integrate beside their ability

to achieve mission goals. Ultimately, an agent framework provides intelligence in contingency management in military and civilian applications alike.

The next section examines some of the basic concepts of intelligence, the ANSI reference architecture, autonomous vehicles, and robots. Section 3 is a theoretical overview of agent architectures. It is here where 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. The whole of section 4 is devoted to abstractions and the processes of creating abstractions. Discrete abstractions of continuous systems are important for the creation of formally verifiable agent architectures that is the basis of trustworthy autonomous vehicles. The remaining sections 5–8 report on recent engineering efforts of developing autonomous vehicles such as autonomous underwater vehicles (AUVs), autonomous ground vehicles (AGVs), autonomous aerial vehicles (AAVs), and autonomous spacecraft (ASC). The conclusions section summarizes the lessons of this review.

1.1 Acronyms used

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

2 ARTIFICIAL INTELLIGENCE OF AUTONOMOUS VEHICLES

A key question is why one would raise the issue of 'intelligence' in connection with autonomous vehicles. Primarily 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 all of 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 vehicle 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 vehicles that reduces their need for higher levels of intelligence.

'Intelligence' is one of the hardest concepts to grasp and formalize. For the purpose of this brief review 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.

Some practical autonomous systems in operation today fall short of such fundamental requirements owing 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

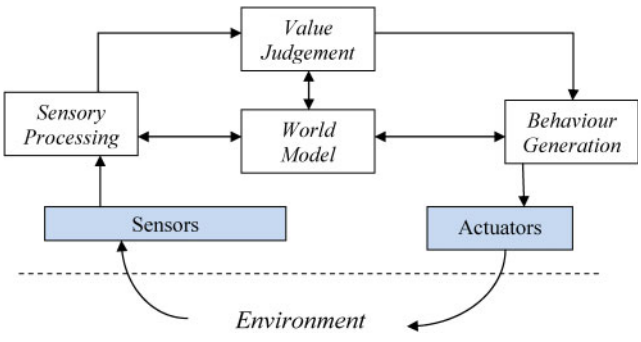

Fig. 2 Reference architecture (ANSI) of intelligent systems [**1**] in the most generic form

(BG) of signalling, relaying images, 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 collaboratively to serve the ultimate goal of the system that is expressed in quality of performance, quality of output, energy efficiency, and reliability.

For engineers with a control systems background Table 1 points out parallel concepts in control systems for engineers with a control systems background.

2.1 Conceptual classifications

The class of 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. This paper takes the view that 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. A number of publications will be quoted from the AMR area for their results can be applied to autonomous vehicles.

2.2 Autonomous vehicle 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.

Although autonomous software is an indispensable component of an autonomous vehicle, there are

Table 1 Concepts of the ANSI reference architecture for intelligent systems as compared with parallel concepts in control engineering

ANSI intelligent systems concept	Control engineering concept	Brief description of differences
Sensory Processing	Filtering, data fusion	Sensory processing can be more complex than filtering in that it can include abstraction processes such as recognition of environmental objects and events, Sensor processing can also involve learning of appearances and learning to recognize things in the environment in an unsupervised manner.
Behaviour Generation	Control algorithm	Behaviour generation can be more complex than running a simple feedback/feedforward control algorithm. It can also include complex composition of switching and real-time adaptation of controllers that can be controlled symbolically.
World model	Modelling and estimation	World model can be more complex than dynamical modelling and parameter estimation. World model can also include static geometric models, visual models, memory organization, abstract models of the present and past, etc.
Value judgement	Control criterion	Value judgement can be much more than evaluating a fixed performance function. Goal-oriented behaviour of agents means that the agent selects its own performance functions which can be constrained by behaviour rules and can require to strike a compromise between partially conflicting goals.

Table 2 Summary of what is needed to build a functioning autonomous vehicle. There are many technological papers contributing to 'autonomous' vehicles while discussing only topics 1–4

	Is the system component specific to this class of vehicles?	
Area of research/system component	Remotely operated vehicles (ROV)	Semi-autonomous and autonomous vehicles AV
1. Efficient energy source	No	No
2. Structural hardware	No	No
3. Computing hardware	No	No
4. Sensors and actuators	No	No
5. Autonomous software	No	Yes

hardware components that support autonomy in their respective areas. These include

(a) energy harvesting (http://www.energyharvestingjournal.com) [**2**, **3**];
(b) self-repairing or self-reproducing structures[**4**];
(c) self-reconfiguring computing hardware [**5**];
(d) self-diagnostic sensors and actuators [**6**].

It is clear, however, that the software component is an indispensible element of autonomy. The rest of the components are useful for remotely operated vehicles and are not indispensible for behaviour autonomy.

3 AUTONOMOUS SOFTWARE 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 reviewed owing to their general use today. Finally agent-oriented programming, multi-agent systems and levels of autonomy are discussed.

3.1 Object-oriented programming and hybrid systems modelling

Software that can handle sensory objects, action plans and memory to make decisions has initially been programmed in object-oriented programming (OOP). Yasuda [**7**] describes OOP for distributed control environments in industrial multi-robot applications [**8**]. Workpieces, robots, and other industrial devices such as sensors are defined as 'software objects' with class inheritance hierarchy. Task-level programming and automated robot program gen-

eration can be supported with three-dimensional (3D) graphic simulation.

Ng and Luk [**9**] develop OOP further and present a multi-paradigm language I+ that has been designed with decision making in mind. It is an integration of the three major programming paradigms: object-oriented, logic, and functional. I+ has an object-oriented framework in which the notions of classes, objects, methods, inheritance, and message passing are supported. Methods may be specified as clauses or functions, thus the two declarative paradigms are incorporated at the method level of the object-oriented paradigm. In addition, two levels of parallelism may be exploited in I+ programming. Therefore I+ is a multi-paradigm language for object-oriented declarative programming as well as parallel programming. Ridao *et al.* [**10**] presents an object-oriented control architecture for autonomy that is organized in three layers (deliberative, control execution, and reactive) merging the deliberation mechanisms of planning with reactive behaviours in charge of controlling the robot in real-time. An AUV has been programmed using this approach to navigate through a sequence of way-points while avoiding contact with obstacles. A general design methodology of OOP for data-focused bottom-up design of autonomous agents is presented in reference [**11**].

In terms of software the OOP approach has historical links with hybrid system modelling software. The formal theory of hybrid systems (HS) can be used to represent decision making of agents in a continuous environment. Table 3 contains a list of hybrid system software packages that can be used to study decision making in a continuous environment in simulations. The same packages can also be used to program embedded systems by transferring the decision-making parts of the simulations into embedded software in applications.

Note that neither of the languages/software in Table 3 is explicitly supporting agent-oriented programming (see section 3.5) in terms of defining the reasoning cycle of agents based on goals, behaviour constraints, agent knowledge, or abstraction layers and skills. The software packages in Table 3 are hybrid system programs on which agent-oriented programming can potentially be built on.

3.2 Agents for autonomous vehicle control

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

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.

Table 3 Some popular hybrid system (HS) analysis, simulation, and controller development tools, most of them programmed under the OOP paradigm

Software environment	Short description, purpose	Reference and website
Charon	HS modelling language, supports hierarchy and concurrency, has simulator and interfaces to Java	www.cis.upenn.edu/mobies/charon [**12–14**]
Modelica/Dymola	OO HS modelling language for multi-domain physics, has simulator, has object libraries in various domains	[**13**, **14**] http://www.modelica.org/ http://www.3ds.com/products/catia/portfolio/dymola
HyTech	Modelling and verification of hybrid automata, has symbolic model checker	[**13**, **14**] http://embedded.eecs.berkeley.edu/research/hytech/
HyVisual	Visual modelling (Ptolemy II) and simulation of HS, supports hierarchies	[**13**, **14**] http://Ptolemy.berkeley.edu/hyvisual/
Scicos/Syndex	Modelling and simulation of HS, has toolbox, real-time code generation, provides formal verification tools	[**13**, **14**] http://www.scicos.org/ http://www.syndex.org/
Shift	Has its own programming language for modelling of dynamic networks of http://path.berkeley.edu/shift/ hybrid automata, has extension to real-time control and C-code generator	[**12**, **14**], http://path.berkeley.edu/shift/
Simulink/Stateflow	HS analysis, simulation, has libraries and domain specific block-sets, can compile C-code for embedded applications via the use of embedded MATLAB™, and Real Time Workshop™	[**12–14**] http://www.mathworks.com/
HSIF	Hybrid Systems Interchange Format	[**12**, **13**, **15–19**]

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 summarized in reference [**20**]: '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 optimize 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.'

3.3 Formal definitions of agents

A formal description of agents [**21**–**24**] 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

$$action : S^* \rightarrow A$$

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

$$env : S \times A \rightarrow \wp(S)$$

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

$$h : s_0 - {}^{a}{}_1 \rightarrow s_1 - {}^{a}{}_2 \rightarrow s_2 - {}^{a}{}_3 \rightarrow s_3 - {}^{a}{}_4 \rightarrow s_4 \dots$$

hist(*agent*,*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 *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

$$see : S \rightarrow P,\ act : P^* \rightarrow A$$

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. 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:

(a) reactive agents;
(b) behavioural agents;
(c) logic-based agents;
(d) situation calculus;
(e) layered architectures;
(f) 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 previous approximate classification of agents (a)–(f). 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.

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 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 references [**25**] and [**26**]. The tutorial paper [**27**] calls the architecture 'the backbone of a robotic system'. Reference [**28**] presents 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 reference [**29**]. The rest of this section introduces 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.

3.3.1 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 Δ, Δ_1, Δ_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.

The notation $\rho(D) \in \wp(L)$ is used for a set of deduced formulae 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 $\Pi \subseteq \mathrm{L}^* \times \mathrm{I}$ (where $\mathrm{I} = \{1, 2, 3, \ldots\}$ 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-update*: $\rho(D) \times G \rightarrow G$ function to be performed during each reasoning cycle to develop sub-goals and cancel unachievable goals.

A number of pure logical approaches have been developed and applied to agents since the early 1980s and throughout the 1990s; for instance well known systems are CONGOLOG [**30**], [**31**], or METATEM [**32**]. A more practical approach in reference [**25**] uses various performance measures, including utilities, costs, and fitness, and probability theory for decision making of rational agents. Eberbach [**33**] 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.

3.3.2 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 [**22**].

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 $a \in \mathrm{A}$ is an action. One says that a behaviour (c, a) *fires* in state $s \in \mathrm{S}$ iff $see(s) \in c$. Let $\mathrm{R_b} = \{ (c, a) \mid c \subseteq P, a \in A \}$ be the set of all behaviours. The agent's *behaviour rules* will be defined as some subset $\mathrm{B} \subseteq \mathrm{R_b}$. A binary *inhibition relation* $\lhd \subseteq \mathrm{B} \times \mathrm{B}$ is defined as a transitive, irreflexive and anti-symmetric relation. The inhibition relation $\lhd$ 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 $fired(p) = \{ (c, a) \mid (c, a) \in \mathrm{B} \ \& \ p \in c \}$. For each $(c, a) \in fired(p)$ it is found out whether there is a $(c', a') \in fired$ such that $(c', a') \lhd (c, a)$ 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 prioritize.

Computational simplicity is a great strength of the subsumption architecture: the overall time complexity is not greater than $\mathrm{O}(n^2)$ where n is the number of behaviours or percepts, whichever is greater. This allows strictly bounded and small decision time in real-time 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 relationship 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 [**22**]. A similar approach is the *agent network architecture* [**34**, **35**]. The related work on *situated automata* [**36**] is interesting in that it provides a method to compile agents, specified in a logical framework, into computationally efficient and very simple machines.

Rosenblatt *et al.* [**37**] present 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, Ridao *et al.* [**10**] use voting mechanisms. With regards to higher levels of autonomous behaviours, Bryson [**38**] 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. Bryson [**38**] presents behaviour-oriented design of agents for engineering agents with complex mission capabilities. Arkin [**39**] provides a systematic descriptions and an overview of decision-making methods in behaviour-based robotics. Byrnes *et al.* [**40**] provides experimental comparison of hierarchical and subsumption software architectures for control of an autonomous underwater vehicle. Flanagan *et al.* [**41**] describe 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 reference [**41**]. Leyden *et al.* [**42**] describe 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.

3.3.3 *The 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 owing 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 $do(move(2, 3), \mathrm{S}_0))$ 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

$$do : A \times \mathrm{S} \rightarrow \mathrm{S}.$$

A binary predicate symbol $\sqsubset \subseteq \mathrm{S}\times\mathrm{S}$ 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 $Poss \subseteq A\times\mathrm{S}$. $Poss(a, s)$ 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 quantization 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

$$Poss(A(x_1, \ldots, x_n), s) \Leftrightarrow P_A(x_1, \ldots, x_n, s)$$

where A is an n-ary action function symbol and $P_A(x_1, \ldots, x_n, s)$ is a formula that is uniform in s and whose free variables are among $x_1,\ldots,x_n$, 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 for instance

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

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

$$F(x_1, \ldots, x_n, do(a, s)) \Leftrightarrow \Phi_F(x_1, \ldots, x_n, a, s)$$

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

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

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

A 'basic action theory' D is the union of sets of axioms: $D = \Sigma \cup D_{\mathrm{ss}} \cup D_{ap} \cup D_{\mathrm{una}} \cup D_{\mathrm{so}}$ 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 $D_{\mathrm{una}} \cup D_{\mathrm{so}}$ is satisfiable. Further details of the properties of the situation calculus can be found in reference [**43**].

GOLOG [**44**] is a robot action programming language based on the situational calculus [**45**] 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 real-time 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.

3.3.4 Beliefs–desire–intention (BDI) architectures

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 [**23**]. Practical reasoning involves two important phases: (a) what goals a human want to achieve and (b) how they are going to be achieved. 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 [**21**, **46**].

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

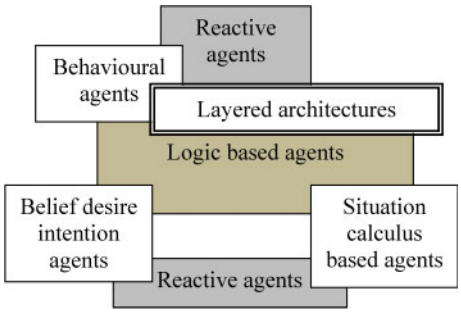

Fig. 3 A map of decision methods for autonomous agents. Most approaches to decision making in current autonomous vehicles can be placed on this map

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

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

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

The main function of 'options' is means-end reasoning, and this must be consistent with beliefs and current intentions as well as 'opportunistic' to recognize 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 implementation of the BDI architecture is the procedural reasoning system (PRS) in reference [**46**]. A large amount of work has been carried out to formalize the BDI model, a key paper being that by Rao [**47**], which describes decision procedures for prepositional linear-time BDI logics. Urlings *et al.* [**48**] propose 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.

3.3.5 *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 recognized: to achieve high speed and 'intelligent' behaviour, the subsumption architecture was advocated [**22**, **49**]. 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 the face of sensor errors in practice. In response to this, more complex architectures appeared in a series of papers [**46**, **50–53**]. 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 lead 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' [**54**, **55**], 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 reference [**56**].

Apart from the layers of 'deliberation', 'coordination', and 'control', the agent also needs the reverse process of 'sensing, signal processing', 'abstraction', and 'organization of abstractions' that includes attention control for a purpose. This is represented

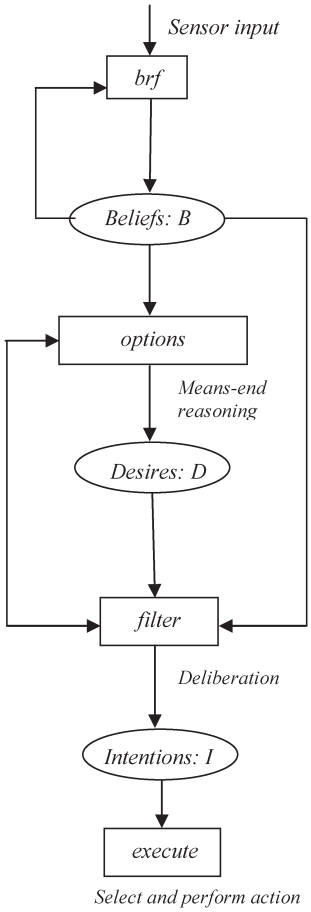

Fig. 4 Block diagram of belief–desire–intention (BDI) agent reasoning cycle

in Fig. 4 with the parallel but opposite direction of information flow that includes sensing through signal processing, data fusion and abstraction organization 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. 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 the Sequencer.

Figure 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 formalized, 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 reference [**31**] and a well-known example of vertical architectures is INTERRUP, as described in reference [**57**].

Evans *et al.* [**58**] present 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

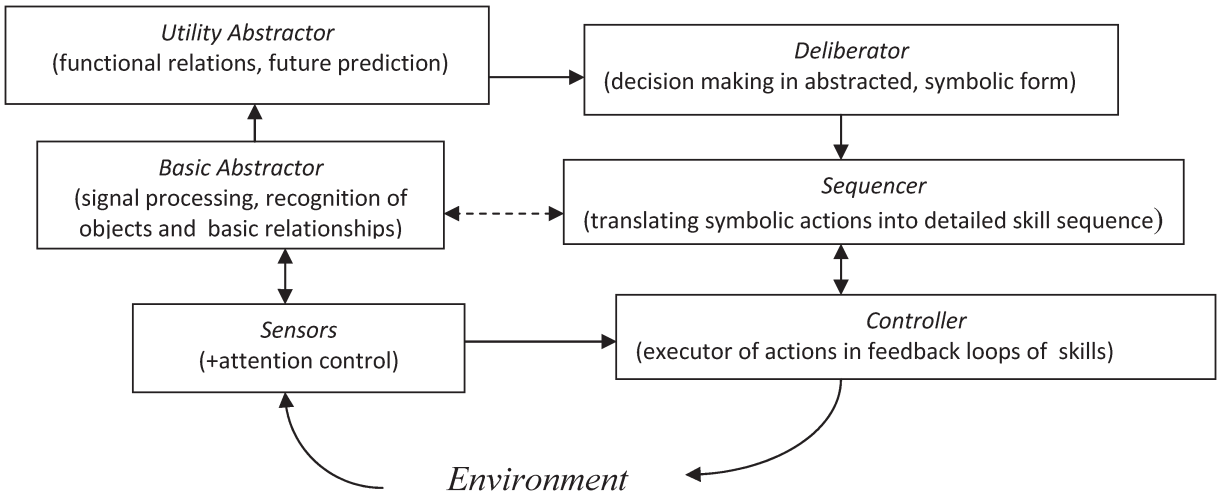

Fig. 5 An advanced layered architecture [**56**] that can have middle layers communicating: the Sequencer passes on to the Abstractor what it expects and the Abstractor passes an symbolic information to the *Sequencer* and thereby form a 'symbolic control loop' at a higher level than the control loop of the *Controller* with sensor signals

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.

Sousa *et al.* [**59**] advocate a layered control architecture for executing multi-vehicle team coordination algorithms with specific team behaviour. The control architecture consists of three layers of the team controller, vehicle supervisors and manoeuvre controllers.

3.3.6 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.

3.4 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. 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 maximize 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 $\Pi(\pi) = \Sigma_1^{\infty}\ \alpha^t\ R_{\pi(s)}(s_t, s_{t+1})$ where $0 < \alpha < 1$ is a discounting factor. The agent can maximize its performance by some version of an iterative procedure of evaluating

$$\pi(s) = \arg_a \max \left\{ \sum\nolimits_{q \in S} P_a(q|s)(R_a(s,q) + \alpha \Pi(q)) \right\}$$

where

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

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 a brief report is given on some interesting applications of MDPs that are relevant for autonomous vehicle controls.

Foka and Trahanias [**60**] develop 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. Sotelo *et al.* [**61**] present a low-level controller under communications and navigation uncertainty that has been applied to an autonomous robotic system using a WiFi-based POMDP. References [**62–64**] 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.

Belker *et al.* [**65**] describe how a robot can autonomously learn to execute navigation improvement plans. The problem is formalized 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. Boutilier *et al.* [**66**] use 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. Dearden and Boutilier [**67**] propose 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. Weng [**68**] 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.

3.5 Agent-oriented programming (AOP)

Table 4(a) lists some of the most popular agent-oriented programming languages and development environments that engineers can use to develop autonomous systems. The first six 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 their 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 (MATLAB^{TM}/Simulink/Stateflow^{TM}, 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 4(b) contrasts with Table 4(a) 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 real-time code development for real robots easier. Some of these, such as MS Robotics Studio, are low-level programming environments that have the potential to be extended with AOP features.

From the 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 objects that 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

Table 4a Some popular software packages for agent oriented programming (AOP)

Software	Supports Agent Oriented Programming	Provides facility for hybrid system definitions	Supports emb edded 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, [**69**]
PRS: proced. reasoning system	+	—	+	—	—	www.ai.sri.com/~prs, [**70**]
3APL 3APL-M	+	—	+	—	—	http://www.cs.uu.nl/3apl/, http://www.cs.uu.nl/3apl-m/, [**71**]
Golog	+	—	+	—	—	http://www.cs.toronto.edu/cogrobo/main/systems/index.html
Jack	+	—	+	—	—	http://www.agent-software.com.au/products/jack/
CAT (MATLAB)	+ (Jason based)	+ MATLAB /Stateflow	+	+ (MCMAS based)	+ (RTW, μC/OS)	www.cognitive-agent-toolbox.com, [**72**]

Table 4b Some popular software robot programming languages without agent oriented programming

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.kukarobotics.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

interactive programming framework for autonomous vehicle designers.

The key idea of AOP is directly to 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 [**73**] about the fundamentals. For instance Oka *et al.* [**11**] propose 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.

Lespérance *et al.* [**30**] describe 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; Veres *et al.* [**72**] outline a methodology borrowed from the already available fault detection and isolation (FDI) algorithms from the manufacturing industries.

Veres *et al.* [**72**] and Sterritt *et al.* [**74**] describe 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'. Veres *et al.* [**72**] introduce 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 [**75**] and the use of natural language programming [**76**, **77**] 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 reference [**78**].

As the world is complex and pre-programming of responses for agent to all situations is too demanding, one important aspect of vehicle agents is learning. Learning always introduces some uncertainty but the most capable systems today are strongly learning based. This is the challenge of creating agents with the ensured performance of learned abilities. Uncertainty about the behaviour of a vehicle agent can be dangerous in safety-critical applications. If, however, learning is not extended to behaviour logic then

there is an opportunity to reconcile this, as in reference [**79**], by creating autonomous vehicles that learn skills of navigation, perception of the environment and manoeuvre-control. Table 5 presents some of the most popular verification tools that can be used in connection with AOP.

3.6 Multi-agent decision processes

A rich source of ideas and techniques originates for multi-agent decisions from teams in annual Robocup competitions. Reference [**80**] 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 decision. 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 reference [**80**] the authors describe a multi-agent-decision process where each agent's action set is determined by a set of decision functions: (a) shooting evaluation (b) 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 real-time actions. More details of this approach can be found in reference [**80**].

Reis and Lau [**81**] use 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 reference [**82**]. A detailed analysis of multi-agent decision making is however beyond the scope of this paper, see references [**83**] and [**84**].

A very original approach to programming is presented in reference [**85**] that advocates a formal grammar-based method to self-assembly and self-organization of robot teams with an overall system behaviour emerging from the concurrent system of member robot actions. Kohno *et al.* [**86**] describe 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.

Nembrini *et al.* [**87**] present the results of simulations investigating a decentralized 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 references [**34**] and [**35**]. Semsar-Kazerooni and Khorasani [**88**] present a game theoretical approach to multi-agent team coordination. In reference [**59**] a layered control architecture is programmed for executing multi-vehicle team co-ordination algorithms along with the specifications for team behaviour.

3.7 Levels of autonomy

The initial three levels of autonomy for autonomous vehicles can be described as follows.

Level 1. The vehicle is stabilized 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.

Table 5 Some popular formal verification tools by model checking

Name	Description	Webpage and references
Uppaal	Systems that Uppaal is capable to model and verify are systems represented as networks of timed automata. The software consists of a description language, simulator and a model checker. Uppaal supports verification of simple liveness and reachability properties of timed automata.	http://www.uppaal.com/ [**14**, **89–91**]
SMV	SMV is a tool for checking finite state systems against specifications in the temporal logic CTL (computation tree logic). The input language of SMV can be used to describe finite state systems ranging from the completely synchronous to completely asynchronous, from the detailed to the abstract, through various refinement stages. The logic CTL allows specifying in a concise syntax a rich class of temporal properties, such as safety, liveness, fairness, and deadlock freedom.	http://www.cs.cmu.edu/~modelcheck/smv.html [**92–96**]
NuSMV	NuSMV is built on SMV concepts, allowing modular system architecture and some additional functionality features such as linear temporal logic (LTL) specifications, and multiple interfaces for interaction with the system. NuSMV is considered as a variant of SMV, focusing on compositionality. NuSMV allows LTL model checking using a tableau construction.	http://nusmv.irst.itc.it/ [**97–100**]
MCMAS	MCMAS is a model checker for multi-agent systems, intended for the automatic verification of formulae with temporal and epistemic properties in deontic interpreted systems. MCMAS uses an extended variant of an interpreted system's formalism. Each agent from a system of agents can be modelled by means of a set of local states, a set of actions, a protocol and an evolution function. In addition to the agent's representation, a set of initial states, the evaluation relation, agent groups, fairness statements and requirement formulae completes the description of a system.	http://www-lai.doc.ic.ac.uk/mcmas/ [**101–104**]
SPIN	SPIN is considered a well-known LTL model checker. Transition systems are modelled using the Promela input language, and LTL formulae are checked using the algorithm described in reference [**105**]. SPIN verification models are aiming to prove the correctness of process interactions, attempting to abstract away from internal sequential computations. Process interactions can be specified in SPIN with rendezvous primitives, with asynchronous message passing through buffered channels, through access to shared variables, or with any combination of these. Channel systems that are familiar for describing communication protocols form the basis of SPIN's Promela input language.	http://spinroot.com/spin/whatispin.html [**93**, **100**, **105–108**]
VIS	VIS is a system for formal verification, simulation and synthesis of finite-state hardware systems. VIS is able to synthesize finite state systems and/or verify properties of such systems, which have been specified hierarchically as a collection of interacting finite state machines. VIS supports fair CTL model checking, language emptiness checking, combinational and sequential equivalence checking, cycle-based simulation, and hierarchical synthesis.	http://vlsi.colorado.edu/~vis/ [**109**]
HyTech	HyTech verifies linear hybrid systems against specifications given as temporal logic expressions. The modelled hybrid systems are specified as collections of automata with discrete and continuous components. Temporal requirements are verified by symbolic model checking.	http://embedded.eecs.berkeley.edu/research/hytech/ [**110**, **111**]
Kronos	In Kronos components of real-time systems are modelled by timed automata and the correctness requirements are expressed in the real-time temporal logic TCTL. TCTL is an extension of the temporal logic CTL that allows quantitative temporal reasoning over dense time.	http://www-verimag.imag.fr/DIST-TOOLS/TEMPO/kronos/ [**112–115**]
Taxys	Taxys is a tool that has been resulted by extending Kronos with a compiler for the synchronous language Esterel [**116**, **117**].	http://www-verimag.imag.fr/~yovine/projects/taxys/index.html [**118–120**]

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 authorization [**121**] 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 optimization 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. Owing to the possibility of unexpected events, these are likely to pose some challenges in the foreseeable future.

4 ABSTRACTIONS FOR VEHICLE AGENTS

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

(a) for the robot's decision making in terms of behaviour rules;
(b) 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 reviewed for agent decision making and then verification tools will be reviewed that assume that abstractions have been formed. Comments will also be made 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 abstractions 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 symbolize 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.

Pisokas and Nehmzow [**122**] address 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: (a) a sub-symbolic representation of perceptual space, using the self-organizing feature map (SOM) using neural networks, and (b) 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 generalizing 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 preprogrammed abstractions of the hybrid system of the robot-environment system.

References [**123**] and [**124**] provide abstractions of sensing and actions for AAV movements in natural environments. The hybrid model is mapped by abstractions to a Kripke model of discretized 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 reference [**125**] 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 reference [**83**] for autonomous mobile robots.

In reference [**127**] 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. Fainekos *et al.* [**128**] present 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. Kloetzer and Belta [**129**] use hierarchical abstractions for swarm robots in a two-dimensional (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.

Frazzoli *et al.* [**130**] define 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.

Belta *et al.* [**131**] review 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 [**132**] is used for symbolic control of non-linear 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 [**132**].

The corresponding perceptual abstractions of movements in state space, as obtained from sensor measurements, can be accomplished by abstracted non-linear observers [**72**]. In reference [**133**] a symbolic description of a system is used to both design the lay-out 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.

Pola *et al.* [**134**] show that every incrementally globally asymptotically stable non-linear 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) [**76**] that links human concepts with that of the robot. NLP is based on two major components:

(a) an ontology where all classes and properties can be derived back to basic types of a high level numerical language (MATLAB, ADA, Phyton, etc.) for signal processing of sensor and modelling data;
(b) 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.

Veres and Lincoln [**135**] illustrate the concepts used in sliding mode control of an autonomous spacecraft. Veres and Molnar [**78**] present 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 reference [**72**], [**76**], [**78**], [**135**], and [**136**].

Dennis *et al.* [**137**] describe streams based real-time abstraction processes for logic-based inference of autonomous systems. Veres *et al.* [**72**] develop some of the ideas of Heintz [**138**] further by the use of database query technology and NLP techniques in sEnglish. Patrón *et al.* [**139**] present 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 [**140**] 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 discretized model of the agent interacting in a real-world environment [**137**]. The discretized model consists of logic propositions about the state of the world and the agent's actions. Liu *et al.* [**141**] propose 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.

References [**142**] and [**143**] present 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.

4.1 Formal verification and certification

Formal verification tools use temporal logic statements (formed from basic propositions) to formalize 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. Reference [**144**] presents a review of 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. Despotou *et al.* [**145**] review some possible formal verification techniques to support legal certification and also mention some abstraction and model checking approaches. Alexander *et al.* [**146**] are 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 AUTONOMOUS UNDERWATER VEHICLES

Various descriptions and reviews are available on autonomous mobile robot control architectures [**27**, **39**, **42**, **147**–**151**] and their implementations on autonomous underwater vehicles [**151**–**157**] demonstrate that the field is an area of active research with important contributions and progress, but where further challenges still need to be addressed. A review of key areas in current underwater robotic technologies is presented also in reference [**158**]. The author focuses on the capabilities, recent development on the various subsystems of the autonomous underwater vehicles. The present section describes pertinent points and examples of control architectures used on various types of AUVs.

References [**159**] and [**160**] report on a layered control architecture and states that execution of real-world missions require regulation of competing behaviours by a top-level finite state machine that divides a mission into phases and permits activation of only of those behaviours which are relevant to a given phase. Their application specific software is divided into major categories such as sensor processing, software for performing closed-loop control and software for planning the vehicle's mission. They also point out the following:

(a) a reconfigurable layered control architecture allows the AUV to perform a variety of missions;
(b) a masking technique allows fusion of behaviours and avoids conflicting behaviours;
(c) algorithmic development avoids trapping situations resulting from behavioural conflict-trapping and non-ideal sensor performance.

Byrnes *et al.* [**40**] present experimental comparison of hierarchical and subsumption architectures in simulations for high-level mission control of AUVs. The experiments involved a traversal of a number of way-points, using simple line-of-sight guidance. Simulations were not carried out in real time owing to the limitations of a graphic workstation, aiming to achieve a maximum frame rate in the graphic system. After a series of experiments, Byrnes *et al.* [**40**] conclude that both approaches results in essentially identical behaviour of the system, i.e. the two approaches lead to behaviourally equivalent controllers. Hence, the choice of the method for this type of mission control can be determined in a

pragmatic way, or as dictated by the experience of the programmer.

In reference [**152**] three-layer architecture for complex missions is proposed. In a review on control architectures, the same three-level structure is identified with examples provided, as in classical approaches (organization, coordination, and execution levels); in systems of reactive planning (hierarchic architecture consisting of a temporal planner, coordination level, and functional level) and in hybrid approaches (strategic, tactical, and execution level). A hybrid system consisting of three-level software architecture (strategic, tactical, and execution levels) is described in reference [**161**]. The three levels separate the control requirements into easily modularized functions, enclosing logically intense discrete state transitioning using asynchronously generated signals for control of the mission and real time synchronized controllers that stabilize the vehicle motion to callable commands [**161**].

Valavanis *et al.* [**153**] identify four AUV control architectures: the hierarchical architecture, the heterarchical architecture, the subsumption architecture, and the hybrid architecture. A survey of eleven AUV control architectures is provided, with examples of the hierarchical and hybrid control architectures. Examples for the subsumption architecture include the enhanced version of state-configured layered control architecture [**160**]; as illustration for the heterarchical architecture, a variant example of mixed hierarchical and heterarchical architecture is given. Valavanis *et al.* [**153**] propose a two-level hybrid control architecture for implementation on AUVs. A state diagram residing at the supervisory control level determines the sequences of AUV tasks/operations throughout the various phases of the mission, providing a state-configured functionality for the overall system. The functional control at the bottom level is composed of functionally independent modules arranged in a heterarchical, parallel structure. The functional level modules directly controls the vehicle's actuators, sensors, and hardware components residing directly on the vehicle [**153**]. A conceptual level design is also presented for embedded hardware control architecture.

References [**162**] and [**163**] describe a modular, distributed and networked control architecture adopted for system implementation of the Autosub AUV. The distributed control network of Autosub is implemented using the Echelon Lonworks control network operating standard. The Autosub subsystem nodes are distributed throughout the vehicle and carry out tasks such as guidance/mission control, control of position, depth and forward speed, navigation, actuator control, battery/power system monitoring, and communication. The planning layer/sequencer functionality resides on the mission control node. The role of the mission control node is to issue a timely sequence of pre-loaded control commands to the three main vehicle control nodes, i.e. speed, position, and depth control nodes, allowing the vehicle to follow a 3D course through the sea. These commands are specified as an ASCII text mission script and are downloaded to the vehicle by the operator prior to the start of the mission. Missions are specified in a text script format using event triggers, waypoint defined tracks, and either depth or altitude demands. A mission element might trigger a 'got position' event and specify a new track consisting of two waypoints in latitude and longitude. This system may be regarded as the bottom two levels on the standard three-layer control architecture described in reference [**55**].

Reference [**154**] provides a comprehensive survey of 22 control architectures for underwater vehicles. The three main methodologies consist of the deliberative, behavioural and hybrid architectures. In addition, a tool-kit control architecture category is also included, specifying the case when no apparent control architecture is present. The hybrid control architecture is currently the most popular, as it merges the advantages of the deliberative and behavioural architectures while minimizing their limitations. Deliberative elements offer the hybrid system predictable functions and mission planning capabilities, while the behavioural elements exhibit timely reaction to sensor input using behaviour-based approaches in a changing and dynamic environment. Ridao *et al.* [**154**] conclude with a set of properties that an effective AUV control architecture should possess. A hybrid control architecture structured in three layers is proposed for implementation on the Garbi underwater robot. The higher-level deliberative layer is responsible for mission planning and re-planning on demand by lower-level layers or by an operator; the intermediate control execution level's task is to activate the low-level behaviours and to pass parameters to them; while the functional reactive layer contains the physical sensor and actuator interfaces [**154**].

Ridao *et al.* [**151**] reiterate the same classification of the AUV control architectures into three categories of deliberative, reactive, and hybrid architectures. The authors of reference [**151**] reinforce the

advantage of using the hybrid architecture for AUV control, stating that 'many control architectures recently proposed convergence to a similar structure that addresses the use of reusable and modularized software packages such as task modules and behaviours linked together for both predictability and reactivity'. A description of the ITOCA (intelligent task-oriented control architecture), hybrid control architecture developed for the SAUVIM semi-autonomous underwater vehicle is also provided in reference [**151**]. The hybrid control architecture consists of three levels, i.e. the planning, control and execution layers.

Carreras *et al.* [**156**] describe a comparative evaluation of four behaviour-based control architectures, i.e. schema-based, subsumption, process description language, and action selection dynamics. Behaviours implemented in these architectures include the 'go to', 'obstacle avoidance', and 'avoid trapping' behaviours. The emergent behaviour is determined by the specific coordination method of the behaviour-based control architectures 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 (in terms of smoothness and trajectory length) [**156**].

Carreras *et al.* [**164**] propose a hybrid coordination methodology between competition and cooperation, trying to benefit from the advantages of both. Competitive coordination in navigation tasks confers robustness to the systems with the drawback of non-optimal trajectory performance. Cooperative methods offer the advantage of optimal, i.e. smooth, short and safe, trajectory performance. A behaviour response consists of a vector of normalized robot control action, i.e. robot velocity for a particular degree of freedom, and as well an associated activation level. Hierarchical layering and complete coordination process for the behaviours is accomplished, using the concept of hierarchical hybrid coordination nodes. Depending on the activation level of the behaviours and the hierarchy between them, a hierarchical hybrid coordination node can behave competitively or can determine responses combined in a cooperative manner.

Rosenblatt *et al.* [**37**] describe a distributed architecture for mobile navigation (DAMN) that consists of a group of distributed task-achieving modules or behaviours and communicate with a centralized command arbiter. Various mission subtasks are accomplished by means of these independent behaviours and arbiters, using decoupled controllers. A behaviour encapsulates the perception, planning, and task execution capabilities necessary to achieve one specific aspect of robot control. Each behaviour uses domain and procedural knowledge to vote for desirable actions and against objectionable ones. The arbiter's role is to combine the behaviour's votes to generate controller's commands. The distributed independent and asynchronous behaviours provide real-time responsiveness to the environment, while the centralized command arbitration framework is capable of producing coherent behaviour.

References [**165**] and [**166**] describe a general-purpose guidance and control system for unmanned underwater vehicles. A three-level architecture is proposed to uncouple the guidance determined execution of user-defined motion task-functions, from linear and angular speed control, and mapping of the required control actions onto the actuation system. The addressed hierarchical system divides into the major blocks of Lyapunov-based design guidance module, conventional ROV autopilots for vehicle heading and depth control along with a set of uncoupled speed controllers, and an actuator management system that maps the required control action onto individual actuator settings of a vehicle with over-actuated configuration.

Toal [**149**] states that, in terms of coordination methods used for behaviour output, pragmatism can point to a mix of competitive and co-operative approaches between the behaviours of one robot. In some instances cooperation of behaviours is a preferred solution, e.g. using fuzzy logic to combine behaviours such as 'Obstacle avoidance' and 'Goal seeking' in the context of navigation. Using command fusion, the fuzzy logic-based navigation system implemented on a land-based mobile robot, has offered considerable advantages over the standard subsumption architecture [**41**], [**167**].

References [**150**] and [**168**] detail a control architecture for the guidance, navigation, and control of an AUV. The control architecture is structured in three layers: the higher level planner/sequencer, the control execution layer, and the functional reactive layer. The way-point guidance algorithm is implemented as a higher level planner/sequencer; acting as the deliberative layer in the overall control architecture, transforming the mission into a set of

tasks. The reactive layer takes care of the real-time issues related to the interactions with the environment. It calls a set of low-level controllers with auto-tuning capabilities, followed by a control allocator. The control execution layer interacts between the upper and lower layers, supervising the accomplishment of the tasks.

Reference [**157**] describes a three-layer architecture employed for the SAUVIM underwater vehicle, consisting of the application layer, real-time layer, and device layer. The application layer consists of application software, an application integrator, and sub-task modules. The role of the application layer is to perform task management functions, and to carry out application specific purposes such as the user interface, interpreter for task description language and supervisory control algorithms. The real-time layer comprises of a system configurator, a real-time operating system, and some parts of sub-task modules. The device layer connects directly to hardware and sends actuator command data and performs sensor data acquisition. The addition of a sensor data bus enhances the direct communication between low-, mid-, and high-level layers, resulting in a modified hierarchical architecture. The described control architecture claims to adopt the advantages of both traditional hierarchical and subsumption architecture.

Reference [**169**] proposes an AUV fuzzy neural BDI (belief–desire–intention) model, to facilitate the application of typical BDI agent models. The authors point out that a typical BDI agent model is not efficiently computable and the strict logic expression is not easily applicable to the AUV domain with uncertainties. The presented fuzzy neural network model consists of five layers: input (beliefs and desires), fuzzification, commitment, fuzzy intention, and defuzzification layer. The fuzzy commitment rules and neural network are combined to form intentions from beliefs and desires. The model is demonstrated by solving a pursuit-evasion game, providing satisfactory simulation results.

Ortiz *et al.* [**170**] describe a behavioural control architecture-based on motor schemas with 3D potential fields for the control of an underwater cable tracker. Primitive behaviours are used to implement the reactive control layer of the vehicle. Each behaviour's response is coded as a 3D vector with the orientation denoting the direction to be followed by the vehicle, while the magnitude is expressing the strength of the response against other behaviour's commands. A hybrid coordination mechanism determines the final control system response, combining recommendations from multiple active behaviours in order to obtain a single control action, representing their consensus.

References [**171**] and [**172**] present a hierarchical, hybrid, model-based architecture for mission control of a generic survey AUV. The control architecture is organized hierarchically, containing various modules. Each module is modelled as a hybrid system, while the overall control architecture is modelled as a set of interacting hybrid systems. 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. The mission controller is hierarchically decomposed into the behaviour controllers, operation controllers, and at highest level being the mission coordinators. A mission consists of a coordinated sequence of operations, each of which is a sequence of behaviours, which are in turn sequences of vehicle commands. It is claimed that this control architecture supports automated code generation, real-time operations, and formal verification.

Zheping and Hou [**173**] propose an architecture for the modelling, control, and coordination of multiple AUV systems. The concept of a hybrid intelligent control agent is used, as an intelligent agent wrapped around an embedded hybrid control system. The hybrid control system consists of the hybrid control primitives and the plant. The hybrid control primitives controls the hybrid system modes' dynamics and the mode transitions. The plant represents a nominal model of the process to be controlled. A knowledge-based deliberative planning form is integrated with a set of verified hybrid control primitives and coordination logic, providing full-authority coordinated control of systems by agents.

Sousa *et al.* [**59**] present a hierarchical, layered control architecture for executing multi-vehicle team-coordination algorithms. Execution control is organized as a three-level hierarchy of team controller, vehicle supervisor, and manoeuvre controller.

References [**174**] and [**175**] present a goal-oriented architecture with embedded automated planning and adaptive execution. The T-REX (Teleo-Reactive EXecutive) system is built around the hybrid architecture's sense – deliberate – act paradigm, where sensing, planning, and execution are interleaved. The implementation of the goal-oriented architecture is built on a framework integrating deliberation and reaction, called Deliberative Reactor, based on the Europa-2 [**176**] constraint-based temporal planning library.

Evans *et al.* [**58**] describe an architecture using reactive and deliberative components for the collision avoidance and escape of an AUV. To the traditional three-layer architecture consisting of the sensing, planning and reactive layer, there is a fourth layer added, the scenario layer, that contains fast-reacting deliberative scripts. Using local maps generated by the sensor layer, scripts from the scenario layer are selected and parameterized on the fly to produce an appropriate avoidance behaviour.

Patrón *et al.* [**139**] describe an architecture that facilitates long-term autonomy and on-board decision-making capability for underwater platforms. The developed framework addresses the representation of knowledge and its distribution between embedded agents at all levels of representation. Through a knowledge-based framework and the goal-based mission planning, interoperability of embedded intelligent agents for distributed service discovery and embedded decision making is provided [**139**]. A library of ontologies is developed to implement the knowledge framework required for situation awareness. The goal-based mission planning is concerned with the generation of a mission plan that autonomously covers the mission requirements using the available platform capabilities.

Aguiary *et al.* [**177**] describe a multi-vehicle mission control system. The architecture implementation is based on the development of single and multiple vehicle primitives for coordinated motion control. Aguiary *et al.* [**177**] propose a general architecture for cooperative autonomous marine vehicle control in the presence of time-varying communication topologies and communication losses.

Fiorelli *et al.* [**178**] describe cooperative control of autonomous underwater glider fleets. The cooperative control method for multiple vehicles reported in reference [**178**] enables adaptable formation control and missions such as gradient climbing in a sampled environment. The Slocum autonomous gliders [**179**, **180**] were used with much success in the Autonomous Ocean Sampling Network (AOSN) II project, as well as in the development of the Rutgers University (RU) Coastal Ocean Observation Lab's (COOL) Mid-Atlantic Regional Coastal Ocean Observing System (MARCOOS). The Scarlet Knight robot (modified Slocum autonomous underwater gliding vehicle) achieved a remarkable success, crossing the Atlantic Ocean while collecting oceanographic data.

References [**181**] to [**183**] present control architecture decisions and navigation for the HUGIN AUV that is organized around three processors of control, navigation and payload tasks. The three processes are in effect similar to a multi-agent system organized under HUGIN OS that is a modularized plug and play system for reliability and tolerance against operator errors.

6 AUTONOMOUS GROUND VEHICLES

Autonomous ground vehicles are best known from DARPA Grand Challenges 2004, 2005 (http://www.darpa.mil/grandchallenge/index.asp) and the DARPA Urban Challenge 2007, that have popularized the idea of AGVs and the need for research effort in this area. The goal has been to follow a route and safely arrive to a distant location without hitting obstacles and causing damage on the way. Samples of technical reports [**184**–**186**] indicated that most research effort went into designing reliable sensing systems and rule-based execution mechanism. At DARPA most of the decisions were based on current situations reported by sensors and how much of the route has been completed, hence reactive behaviour based or situation calculus-based approaches were used.

The following section briefly describes from the perspective of the employed control architecture, the autonomous ground vehicles that have participated in the DARPA Urban Challenge [**187**].

Reference [**188**] describes the sensing, planning, navigation, and actuation systems for Little Ben autonomous ground vehicle (modified Toyota Prius hybrid vehicle). Omni-directional and long-range sensing navigation is provided by an array of GPS/INS, LIDAR and vision sensor systems. Important criteria for the design of the vehicle are to meet the desired detection distance, processing time objectives, as well as to achieve a reaction time that ensures safe operation of driving manoeuvres at a mandated upper speed limit of 30 mph. The combination of hardware sensing systems with associated reactive software modules allows meeting the performance demands required of such an autonomous vehicle travelling in an uncertain urban environment. The layered software architecture is organized hierarchically into a series of modules connected via interprocess communication messages. The planning algorithms consist of a high-level mission planner that uses information from an appropriate provided route network definition file and mission data file to select routes, while the lower-level planner uses the latest dynamic map information to optimize a feasible trajectory to the next waypoint.

Chen *et al.* [**189**] provides a detailed description of a modified US Marine MTVR (medium tactical vehicle replacement) along with the overall system architec-

ture that includes the sensors and sensor processing system, the mission planning system, and the autonomous behavioural controls that are implemented on a TerraMax vehicle. The vehicle is equipped with camera-based vision system, LIDAR, and GPS/INS systems. The control architecture employs a layered design, where various software modules act as services and provide specific functions to the overall system. Two different types of services are identified: i.e. the autonomous services, whose modules provide functionalities for autonomous vehicle behaviours; and the system services, whose modules support the reliable operations of the vehicle and the mission. The service functionality software modules resemble an agent-based control architecture design. The autonomous services include the autonomous vehicle manager (AVM), and the autonomous vehicle driver system (AVD). The AVM manages the high-level autonomous operation of the vehicle. It is primarily responsible for performing route planning, trajectory planning, and behaviour management. The AVD provides vehicle-level autonomy, such as waypoint following and lateral, longitudinal, and stability control by accepting messages from the AVM and commanding the lower level control-by-wire actuations.

Reference [**190**] describes a LIDAR-based navigation approach to be used in situations without any prior knowledge about the terrain and without GPS. In addition to the solely LIDAR-based navigation system, at the DARPA Urban Challenge this approach was combined with a GPS-based path follower. The approach was used on two vehicles: a VW Passat of DARPA Urban Challenge finalist team AnnieWAY, and on a VW Touareg MuCAR-3 (Munich Cognitive Autonomous Robot, third generation). These autonomous vehicles are enabled with motion planning capability within an unknown environment, relying on an approach of using a set of virtual antennae called 'tentacles' probing an ego-centred occupancy grid for drivability. Similar to an insect's antennae, they fan out with different curvatures discretizing the basic driving options of the vehicle. The paper describes how this approach can be used for exploration of unknown environments and how this can be extended to combined GPS path following and obstacle avoidance that allows safe road following in case of GPS inaccuracies.

Kammel *et al.* [**191**] describe the hardware components, software architecture and algorithms developed for the AnnieWAY automobile using a VW Passat Variant as base. Environmental perception relies on range and reflectivity measurements, provided by a laser-based range and intensity sensor. Range measurements are used to provide 3D scene geometry. Measuring reflectivity allows for robust lane marker detection. The navigation system also features an advanced INS system with GPS receivers for position and accurate heading measurements. Odometry is provided by wheel encoders. Among the core components of the software architecture are mentioned the perception of the environment, a situation interpreter for selection of the appropriate behaviour, a path planning component and an interface to the vehicle control. Mission and manoeuvre planning is conducted using a concurrent hierarchical state machine. The major challenge imposed by the competition is collision-free driving in traffic in compliance with traffic rules. In order to accomplish this the vehicle is required to be capable of analysing the situation, choosing the appropriate behaviour and executing it in a controlled way. In reference [**191**] these problems are addressed by a planning module organized in three layers, carrying out functionalities of mission planning, manoeuvre planning, and collision avoidance.

Leonard *et al.* [**192**] describe the architecture and implementation of on Talos (Land Rover LR3), an autonomous passenger vehicle designed to navigate using locally perceived information in preference to potentially inaccurate map data to navigate a road network while keeping to traffic rules. The overall system architecture includes subsystems of road paint detector; lane tracker; obstacle detector to identify stationary and moving obstacles; low-lying hazard detector; fast approaching vehicle detector; positioning module estimating the vehicle position in two reference frames; navigator that tracks the mission state and develops a high-level plan; drivability map providing an efficient interface to perceptual data; motion planner that identifies and optimizes feasible vehicle trajectories; and the controller unit executing low-level motion control. The vehicle architecture is claimed to address the requirements of perceiving and navigating a road network with segments defined by sparse waypoints. The requirements of carrying out complex manoeuvres in an urban environment are generated with a unified planner. A key feature of the planner consists in using closed-loop simulation in a Rapidly-exploring Randomized Trees (RRT) algorithm that can randomly explore the space while efficiently generating smooth trajectories in a dynamic and uncertain environment. The employed solution resembles a layered control architecture, where (a) the 'navigator' is responsible for planning the high-level behaviour of the vehicle, such as determining the

shortest route to mission data file (MDF) checkpoints, dealing with intersection precedence, merging, crossing and blockage situations, etc.; (b) the motion planner receives, as guidance goal, a point from the route network definition file (RNDF) and uses the RRT algorithm for path generation, then outputs a motion plan consisting of a path and speed command; and (c) the controller is responsible for taking the motion plan and generating control signals for tracking the desired motion plan.

Reference [**193**] addresses the hardware and software design of the autonomous platform XAV-250 (Ford F-250). An overview is given of the vision, radar and LIDAR-based perceptual sensing suite, its fusion with a military grade inertial navigation package and map-based control and planning architectures. The autonomous platform has been used for the investigation and testing of various control architectures. Initially the control architecture employs a situational-dependent behaviour-based solution. The arbiter combines the outputs from the current set of contextual behaviours to produce a resultant steering and speed response. Although this framework proved effective, arbitration delays have caused unwanted results. As an alternative to this approach, a map-based sensor fusion strategy that was neither situational dependent nor arbitrated has been pursued. The layered software architecture reported in reference [**154**] has also been successfully adapted to and used on the XAV-250 autonomous ground vehicle.

Miller *et al.* [**194**] describe Skynet, an autonomous vehicle with a Chevrolet Tahoe at its base, employing a multilayer perception and planning/control solution. The vehicle is equipped with a GPS, IMU and vehicle odometry sensor for measurements in the vehicle's own reference frame. These measurements are fused in the pose estimator to obtain position, velocity and attitude estimates in an Earth-fixed coordinate frame. The vehicle is also equipped with cameras, radar and LIDAR systems. Skynet subsystems include the actuation and power distribution designed in-house, a tightly coupled attitude and position estimator, novel obstacle detection and tracking system, a system for augmenting position estimates with vision-based detection algorithms, a path planner based on physical vehicle constraints and a non-linear optimization routine, and a state-based reasoning agent for obeying traffic laws. The planning system uses a probabilistic perception of the environment to plan mission paths within the context of a rule-based road network. The planner is split into three primary layers, i.e. the top-level behavioural layer, the middle-level tactical layer, and the low-level operational layer. The behavioural layer combines offline mission information with sensed vehicle and environment information to decide which high-level behavioural state should be executed given Skynet's current context. The tactical layer plans a contextually appropriate set of actions to perform, given Skynet's current pose and the states of other nearby agents. The operational layer translates these abstract actions into actuator commands, taking into account road constraints and nearby obstacles.

Reference [**195**] describes a practical approach to engineering a robot to effectively navigate in an urban environment. The paper proposes a range of relatively simple sensors, actuators and processors to be used on the Knight Rider robot (1996 Subaru Outback Legacy), to achieve the tasks of robot vision, intelligence and planning. The software architecture is partitioned into six areas: laser data processing, vision data processing, sensor fusion, intelligence, planning, and control. An artificial intelligence (AI) module is responsible for the high-level planning and tactical-level decision making for the Knight Rider. The AI module builds on key aspects and principles of existing research in driver modelling approaches [**196**, **197**]. Such existing driver models are usually structured into a three level (layered) hierarchy [**198**], dividing the driving task into strategic, tactical, and operational levels. The strategic level is concerned with high-level goals such as navigation. The strategic level maps these goals into a series of sub goals, which remain unchanged unless affected by external factors. The tactical level is responsible for generating sequential tasks to implement a given subgoal. The operational level is responsible for low-level guidance of the robot.

Reference [**199**] presents the architecture of Junior, a robotic vehicle (modified 2006 VW Passat) capable of navigating urban environments autonomously. The vehicle's inertial navigation sensor suite consists of GPS receivers, wheel odometry sensor and INS. For obstacle and moving vehicle detection roof-, front- and rear-mounted LIDAR systems, as well as a front-mounted radar assembly are used.

The software is organized into five groups of modules: sensor interfaces, perception and navigation modules, the drive-by-wire interface, and a global services module. The navigation modules are responsible to determine the behaviour of the vehicle. The navigation group consists of a number of motion planners and a hierarchical finite state machine that is responsible to invoke the different

robot behaviours and to prevent deadlocks. The motion planners provide capabilities of global path planning, common road navigation according to a route network definition file (RNDF), free-form navigation for parking lots, and abilities that are required for situations when the vehicle is at road crossings. *A finite state machine (FSM) determines the robot's behaviour by switching* between different driving states.

Rauskolb *et al.* [**200**] describe the Caroline (2006 VW Passat) autonomous mobile robot. The system architecture comprises eight main modules: sensor data acquisition; sensor data fusion; image processing; digital map; artificial intelligence (AI) system; vehicle path planning and low-level control; supervisory watchdog and online-diagnosis; telemetry and data storage for offline analysis. Caroline's design approach uses a multi-sensor fusion of LIDAR, radar, and laser scanners for sensing approaching traffic and stationary obstacles. Four cameras are used for detection and tracking of lane markings. Terrain drivability information is provided by a stereo vision system and an additional camera combined with laser scanners.

The modelled AI system is based on a modified distributed architecture for mobile navigation (DAMN) architecture. The AI module receives a drivability grid resulted from the fusion of data provided by the stereo vision system, laser scanners and additional cameras. The AI system generates driving decisions based theDAMN as proposed by Rosenblatt [**201**]. The implemented DAMN architecture consists of a number of operating behaviours, i.e. 'follow way points', 'stay in lane', 'avoid obstacles', 'stay on roadway', and 'stay in zone', each behaviour holding a vote. The resultant behaviour is determined by arbiter systems using a vote-based mechanism. The decisions made by the AI system are provided to the path planner, which calculates optimal vehicle trajectories with respect to its dynamics in real time. A supervisory watchdog system monitors the vehicle's hardware and software modules in order to ensure safety and robustness.

Reinholt *et al.* [**202**] describe the base vehicle and controls development of the perception-, and planning systems of the Odin autonomous robotic vehicle (a Ford Escape forming the basis). Perception uses laser scanners, GPS, and a priori knowledge to identify obstacles, cars, and roads. Planning, decision making relies on a hybrid deliberative/reactive architecture to analyse the situation, select the appropriate behaviour, and plan a safe path. At the coarsest level of decision planning, the route planner component uses prior information from the route network definition file (RNDF) and mission data file (MDF), to determine the road segments to travel by generating a series of waypoints. The driving behaviours module of the behaviour-based architecture is responsible: (a) for producing behaviour profile commands that defines short-term goals for the motion planning; (b) to request a new set of directions from the route planner in the event of a roadblock; and (c) to control and indicate the intent of the vehicle. An action selection mechanism based on a behaviour-based paradigm is used to determine the emergent driving behaviour. In order to address the situational awareness problem, a system of hierarchical finite state machines is used, allowing the driving behaviours to distinguish between intersection, parking lot, and normal road scenarios.

Urmson *et al.* [**203**] describe Boss (with Chevy Tahoe as basis), an autonomous vehicle that uses on-board sensors (GPS, lasers, radars, and cameras) to track other vehicles, detect static obstacles, and localize itself relative to a road model. A three-layer planning system combines mission-, behavioural layer, and motion planning to drive in urban environments. The mission planning layer considers which street to take to achieve a mission goal. Data provided in the road network definition file are used to create a graph that encodes the connectivity of the environment. A behavioural layer determines when to change lanes, advantage at intersections and performs error recovery manoeuvres. The motion planning layer selects actions to avoid obstacles while making progress towards local goals.

The remainder of this section presents some of the more recent representative examples of autonomous ground vehicle developments, without completeness as the number of papers is large in this area. Fregene *et al.* [**204**] develop a systems- and control-oriented intelligent agent framework called the 'hybrid intelligent control agent' and describes its composition into multi-agent systems. It is essentially developed around a hybrid control system core so that knowledge-based planning and coordination can be integrated with verified hybrid control primitives to achieve the coordinated control of multiple multi-mode dynamical systems. The scheme is applied to the control of a team of unmanned ground vehicles (UGVs) engaged in an outdoor terrain mapping task. Although the framework could potentially allow decision making via deliberative agents, the example provided is similar to reactive: hybrid automaton-based decision making with a priori designed coordinated control transitions between operational

modes ('scan', 'avoid object', 'lock on', 'acquire image', etc.) for autonomous navigation during mapping.

Girault [**205**] addresses the problem of hybrid control of autonomous vehicles driving on automated highways. Vehicles are autonomous, so they do not communicate with each other or with the infrastructure. Two problems are dealt with: a vehicle driving in a single-lane highway must never collide with its leading vehicle; and a vehicle entering the highway at a designated entry junction must be able to merge from the merging lane to the main lane, again without any collision. To solve these problems, each vehicle is equipped with a hybrid controller, consisting of several continuous control laws embedded inside a finite state automaton. The automaton specifies when a given vehicle must enter the highway, merge into the main lane, yield to other vehicles, exit from the highway, and so on. Decision making is based on the hybrid automaton approach (i.e. guard conditions defining transitions between a finite set of operational modes) through which it is proven that if all the vehicles are equipped with this hybrid controller, then no collision can ever occur, and all vehicles either merge successfully or are forced to drop out when they reach the end of their merging lane.

Posadas *et al.* [**206**] deal with the communications framework necessary to design and implement cooperative control of autonomous vehicles. A modular and portable architecture is proposed that allows the development of robot control systems. Its main components are mobile software agents that interact through a distributed blackboard communications framework. Decision making is based on a multi-level and distributed agent architecture using mixed set of reactive and deliberative agents.

Reference [**207**] is a review of automated guided vehicles (AGVs) that are used for the internal and external transport of materials. Traditionally, AGVs were mostly used at manufacturing plants. Currently, AGVs are also used for repeating transportation tasks in other areas, such as warehouses, container terminals, and external (underground) transportation systems. Vis [**207**] discusses literature related to decision and control issues of AGV systems at manufacturing, distribution, trans shipment, and transportation systems. Also a new classification for dispatching rules, a guideline for selecting a suitable scheduling system and a decision framework for design and control of an AGV systems are proposed.

Consolini *et al.* [**208**] investigate leader–follower formations of non-holonomic mobile robots with a new formation control strategy in that the AGV's control inputs are forced to satisfy suitable constraints that restrict the set of leader possible paths and admissible positions of the follower with respect to the leader. The follower position is not rigidly fixed with respect to the leader but varies in proper circle arcs centred in the leader reference frame. This archirecture is purely reactive.

Reference [**209**] presents a cooperative decentralized 2D path-planning algorithm for a group of autonomous vehicles that provides guaranteed collision free trajectories in real-time Agents move on reserved areas which are guaranteed not to intersect. A handshaking procedure is used to guarantee up-to-date information states for the agents. Decisions of agents on executing motion primitives are based on a set of behaviour rules defined over communications conditions, priorities and computational geometry of reserved trajectory regions. Conflicts between agents are resolved by a cost-based negotiation process.

Fuzzy logic decision making is described in reference [**210**] for cooperation of multiple automated vehicles with passengers comfort. This approach develops a cooperative scheme based on a decentralized planning algorithm which considers the vehicles in an initial open chain configuration.

In reference [**211**] a multi-agent architecture for mobile robot control is presented using distributed decision making and compared with other classical architectures (control-loop, layered, and task trees) according to various measures: coordinability (the degree to which a system can coordinate by respecting interdependencies between agent actions, meeting global constraints, and optimizing its operations), predictability, reliability-tolerance, and adaptability.

Ono *et al.* [**212**] propose the use of a multi-agent architecture in order to build easily expandable and portable mobile robotic control programs. Decision making is based on reactive behaviour definitions and fuzzy logic. Different developers can implement agents to perform a specific task in different programming languages, and integrate them into the architecture, provided there is a common communication ontology.

In reference [**213**] the control of a mobile robot is based on a multi-agent system performing cooperative control. The combination of the different controllers is achieved by using fuzzy logic with the individual agents using reactive fuzzy behaviour rules. The benefits of agent technologies and collaborative control are combined to form a single 'robot agent' via a multi-agent system. Fast real-time

execution of a similar system has also been demonstrated in the sensor network area in reference [**214**].

Reference [**215**] investigates formation control of a group of unicycle-type mobile vehicles with inter-robot communication. Each robot has only position and orientation available for feedback control. For each vehicle an observer is used to estimate the unmeasured velocities that drives a feedback controller. Agent operations are essentially reactive agent architectures in that the current situation of a vehicle determines its movement.

Decision processes are also part of navigation systems where ambiguity needs to be resolved. Reference [**216**] presents a method for adaptive mobile robot navigation and mapping. Ono *et al.* [**212**] present a method of corridor navigation using the multi-agent system. Morice and Veres [**217**] describe the application of geometric bounding techniques to 'range only' navigation of an underwater vehicle based on measurements of range from a supporting ship.

7 AUTONOMOUS UNMANNED AERIAL VEHICLES

Remotely controller unmanned, also called 'uninhabited', flying vehicles have been widely used in the air forces, agriculture, rescue, surveillance, and for leisure. Levels of autonomy and certification, as described in section 3.6 is the most developed for AAVs. Autonomous and semi-autonomous AAVs have been built and flow using various simple and advanced decision-making mechanisms such as hierarchical [**218**], reactive and behaviours based [**219–222**], and hierarchical [**223**]. A lot of AAV related effort has been devoted to path planning [**224**, **225**], to fault diagnosis [**226**], and to navigation [**227–231**] to aid correct decision making by the autonomous vehicles. There are a large number of papers on these topics and this section only mentions a few representative samples of publications in this area.

Kim and Shim [**218**] present a hierarchical flight control system for unmanned aerial vehicles that executes high-level mission objectives, by progressively substantiating them into machine-level commands. The information from various sensors is passed to higher layers for reactive decision making. Decision making is based on multilayered agent architecture. Saffarian and Fahimi [**219**] address the problem of helicopter group formations with application of non-linear model predictive control. Although control signal computations are complex non-linear, decision making for autonomous formation flight is essentially reactive.

Shiyu and Rui [**223**] discuss guidance problems of multiple missiles and propose a cooperative guidance scheme, where coordination algorithms and local guidance laws are combined together. The scheme builds up a hierarchical cooperative guidance architecture that in the case of salvo attacks achieves that the missiles hit the target simultaneously. Distributed reactive decision making is a characteristic of such coordinated control schemes.

In reference [**220**] a reactive mechanism is presented for navigation of AAVs that is based on perception-action bionics. This method has been inspired by the olfactory bulb neural activity observed in rabbits under external stimuli.

Zemalache and Maaref [**221**] describe an on-line self-tunable fuzzy inference system of a mini AAV with on-line optimization of a zero-order Takagi–Sugeno fuzzy system by a back propagation-like algorithm. The fuzzy inference system is not used for decision making but to obtain control signals for stability and manoeuvres. Reference [**222**] presents a tracking controller for a small helicopter to follow a given trajectory. A tracking controller based on optimal control theory is synthesized that also handles control constraints. In these approaches decision making is fundamentally reactive despite that fuzzy logic or optimal control is involved in control signal generation.

Kim *et al.* [**224**] propose two real-time path planning schemes based on limited information for autonomous AAVs in a hostile environment. Al-Jarrah and Hasan [**225**] present an algorithm to create paths in 3D for a AAV and the organization of waypoints to achieve a specific goal. Such algorithms *can aid the decision-making* process of a AAVs but they do not determine whether the AAV's decision-making architecture is reactive, layered, behaviour rules based or deliberative. Fault detection can also aid the decision making of a AAV. Reference [**226**] demonstrates an actuator and sensor-based FDI system for small autonomous helicopters. Fault detection is accomplished by evaluating any significant change in the behaviour of the vehicle with respect to the fault-free behaviour, which is estimated using observers. Once the sign of a fault is detected, the vehicle may decide to execute emergency landing before the fault deteriorates.

Duan *et al.* [**232**] present methods for coordinated trajectory re-planning in dynamic and uncertain environments for multi AAVs using the analogy of ant colony optimization. Reference [**121**] looks at search and rescue as an important application area for autonomous VTOL vehicles as a case study. Liu *et al.* [**233**] present mission planning and execution

methods for power line inspection AAVs. Chen *et al.* [**234**] investigate a navigation function-based cooperative control for multiple AAVs in the presence of known stationary obstacles and unknown enemy assets (EAs). The motion of AAVs are planned in a centralized fashion. The standard navigation function approach is extended to a multiple navigation strategy with an analytical switch among different cases due to the limited sensing zone of the AAVs. Decision making is reactive and behaviour rules based in these vehicles.

There are generic methods of fundamental importance in vehicle navigation for decision making; for instance Metni *et al.* [**235**] introduce attitude and gyro bias estimation methods for vertical take off and landing (VTOL) AAVs. Reference [**227**] presents an optical flow-based mechanisms of AAV navigation that is based on a biological model of a motion detector for the purpose of long-range goal-directed navigation in 3D environment. A miniature bio-inspired optical flow sensor is presented in reference [**228**] that is based on low temperature co-fired ceramics technology. Aubépart and Franceschini [**229**] develop a functional model for an elementary motion detector (EMD) and present a Field Programmable Gate Array (FPGA) implementation of an EMD array, designed for estimating the optic flow in parts of the visual field of an MAV. FPGA technology has proved particularly suitable for receiving inputs from a multitude of photoreceptors. Reference [**230**] presents an optical flow navigation sensor. Bachmann *et al.* [**231**] describe the design of a small robot capable of morphing between aerial and terrestrial locomotion. Xie *et al.* [**236**] report on vision-based navigation and tracking of a small airship.

8 AUTONOMOUS SPACE VEHICLES

Autonomous vehicles operating in the space environment are becoming pivotal in the achievement of complex exploratory missions where human presence, or indeed tele-presence, is not possible. Autonomous vehicles operating in the space environment encompass a broad range of vehicle types and echo those observed within a terrestrial environment: autonomous ground vehicles are candidates for planetary surface exploration where soil samples are desired. Autonomous uninhabited air vehicles are candidates for surface mapping and proximate exploration of planets over greater distances than that is possible by a planetary rover. Autonomous underwater vehicles are required for missions involving exploration under the surface ice of frozen moons such as Europa [**237**, **238**]. While these applications are extensions to those observed within our terrestrial environment, autonomous vehicles in space also capture very different scenarios involving small bodies such as comets, where interest is in nucleus sample return and impacting events, exploration and categorization of hazardous asteroid belt environments and complex astrophysics experiments with exquisitely demanding real-time constraints [**74**, **239**, **240**]. All of these scenarios involve an inherent need for autonomy owing to mission-critical communication delays experienced with deep space operations, or indeed communication loss due to planetary occlusion. An early overview of the future possibilities for using agents in space missions is presented in reference [**241**].

Despite being seen as an enabling technology for future space missions, the incorporation of autonomy within any space system is resisted by designers due to lacking an established background. Obviously without ever implementing new technologies, experience and trust in such systems can never be gained: to combat this both ESA and NASA have started planning and executing technology proving missions. Perhaps the most documented and relevant of these missions is the Remote Agent (RA) experiment by NASA: an autonomous spacecraft agent used to operate the Deep Space 1 spacecraft [**242**]. The Automated Rendezvous Vehicle, constructed and operated by ESA to service the international space station, while being highly advanced, is automated rather than autonomous: upon orbit insertion a pre-programmed set of operations are carried out, with humans being able to initiate an abort sequence if deemed necessary. Muscettola *et al.* [**243**] details the design of Remote Agent, which fuses *both reactive and logical agent frameworks* to provide for both robust plan execution and the ability to develop plans to achieve goals. In contrast to the operation of the ATV, RA is not requested to execute a set of commands but is designed such that a set of goals are to be achieved. From the specified goals, RA forms a plan to accomplish the goals and then executes the plan, while concurrently maintaining feedback on the current plan execution and the status of flight hardware. Such goal-oriented programming fits within the BDI agent model methodology, itself a logical framework and similar in design to that of PRS. Reference [**244**] presents an extension of the architecture implemented within RA, through the introduction of machine learning, and was intended to be a technology proving mission for a completely autonomous fleet of cooperating

space vehicles named TechSat-21. Programmed in a C++ implementation of ObjectAgent, the TechSat-21 fleet was destined to be a hierarchical physical agent system, with a single spacecraft coordinating all fleet activity. Sadly, the mission was cancelled.

Numerous agent methodologies for autonomous space vehicles have been presented within simulated environments. References [**245**] and [**246**] present a BDI agent framework programmed using JACK, which is extended to permit numerical computation through a link to MATLAB. Bonnet and Tessier [**247**] present satellite swarm planning via communication and negotiation while tasked with observation in an Earth orbit. Simulated within JAVA using the JADE platform, agent negotiation was observed to prevent redundant observation instances: of interest here would be how well the method extends to a more challenging environment such as asteroid belt exploration.

Behavioural approaches within space vehicle systems have been investigated greatly by ESA, wherein behavioural primitives are combined to result in a global and emergent response by a large-scale satellite swarm [**248**–**250**]. Applied to the concept of in-orbit self assembly, reactive behaviours have been show to exhibit the desired response with minimal computational overhead. While computationally attractive, such methods do not lend themselves to verification of system behaviour: undesired and unexpected behaviours may emerge. Reference [**251**] presents behavioural coordination of satellite clusters through stigmergy within a digital environment, which permits group coordination with lower communication requirements than that required by negotiation. Unlike satellite swarms investigated within reference [**249**], which is absent of any communication and is strictly reactive, stigmergistic action permits a behavioural response via indirect communication.

Autonomous spacecraft action, taken as a skill availed to an agent system that is executed by locally situated hardware systems and not part of any high level reasoning, is of great academic interest. Innumerate papers exist on formation flight, position and attitude control, control reconfiguration, path planning, collision avoidance, as well as guidance and navigation. Taken by themselves such agent skills, as published, are generally shown as reactive automata: although these skills may be interwoven into a suitable agent architecture to enable a highly capable system to be developed. Capability to reconfigure can also be treated as an agent skill; Pei *et al.* [**252**] present autonomous control reconfiguration of aerospace vehicle based on control effectiveness estimation. Ren [**253**] provides a solution to the distributed attitude alignment problem for a team of deep space formation flying spacecraft through local information exchange. Lincoln and Veres [**254**] examine autonomous formation flying of satellites where problems of position, velocity and attitude estimation, data fusion across the cluster, constrained actuators, fuel efficient manoeuvres, and robustness are all optimized by a 'skilled' centralized control agent. Calibration of control gains for translational force actuation is another skill of this agent that it may decide to execute.

9 CONCLUSIONS

This review of decision making has provided some lessons and highlighted future research areas for growth. In summary, the lessons and challenges are as follows.

1. There are almost as many architectures as there are vehicles built and there is no serious effort to share methodologies. As sections 5–8 demonstrate, some principles of operation in decision making and in agent architectures have been shared between some vehicles. At lower levels of abstractions software is not shared and portability and compatibility is lacking [**29**, **74**, **121**, **153**, **251**].
2. The ability of learning is difficult to reconcile with provable performance. Learning always introduces some uncertainty but the most capable systems today are strongly learning based. This is the paradox of 'certainty of learned abilities'. Conclusions from section 4 are that creating methodologies to reconcile formal verifiability with learning of capabilities is the second major challenge [**14**, **58**, **75**, **104**, **123**, **144**].
3. There is no serious effort to define a systematic approach to knowledge engineering for agents. This is partly attributed to the variety of agent architectures around and partly to underdeveloped and incompatible abstraction processes in different agents who need to communicate [**23**, **24**, **76**, **78**, **138**, **139**, **255**, **256**]. Sections 4–8 highlight the diversity and lack of a knowledge engineering approach to intelligent agent development.
4. Reliability of decision making and safety of vehicles strongly depends on high quality sensor systems and mechanism of self localization and mapping. There is a strong industrial demand for practically reliable certifiable real-time sensor and abstraction systems [**58**, **123**, **144**]. Analysis

in sections 3–4 has shown that verifiable logic-based decisions depend on reliable discrete abstractions of sensor signals fundamentally.

Apart from these problems there are hundreds of sub-problems in navigation and robust control of all kinds and perhaps also in new agent architectures themselves. The danger however is that proliferation of agent architecture types will not help industrial certification of autonomous vehicles for public safety. So the final conclusion is that industrial standardization of formally verifiable decision-making processes is desirable for any significant further progress in the area of autonomous vehicles.

ACKNOWLEDGEMENT

This work has been partially funded by EPSRC grants No. EP/F037570/1 and EP/E02386X/1.

REFERENCES

1 **Meystel, A. M.** and **Albus, J. S.** *Intelligent systems: architecture, design, and control*, 2002 (Wiley Interscience).

2 **Ieropoulos, I., Melhuish, C., Greenman, J.,** and **Horsfield, I.** EcoBot-II: An artificial agent with a natural metabolism. *Journal Advanced Robotic Systems*, 2005, **2**(4), 295–300.

3 **Melhuish, C.** and **Kubo, M.** Collective energy distribution: maintaining the energy balance in distributed autonomous robots. In Proceedings of 7th International Symposium on *Distributed autonomous robotic systems*, Toulouse, France, 2004, pp. 261–270.

4 **Hurlebaus, S.** and **Gaul, L.** Smart structure dynamics (a review). *Mech. Systems Signal Processing*, 2006, **20**(2), 255–281.

5 **Tempesti, G., Mange, D.,** and **Stauffer, A.** Self-replicating and self-repairing multicellular automata. *Artif. Life*, 1998, **4**(3), 259–282.

6 **Feng, Z., Wang, Q.,** and **Shida, K.** A review of self-validating sensor technology. *Sensor Rev.*, 2007, **27**(1), 48–56.

7 **Yasuda, G. i.** An object-oriented multitasking control environment for multirobot system programming and execution with 3D graphic simulation. *Int. J. Prod. Econ.*, 1999, **60–61**, 241–250.

8 **Fraser, R. J. C.**, *et al.* Implementing task level mission management for intelligent autonomous systems. *Engng Applic. Artific. Intell.*, 1991, **4**(4), 257–268.

9 **Ng, K. W.** and **Luk, C. K.** I+: A multiparadigm language for object-oriented declarative programming. *Computer Languages*, 1995, **21**(2), 81–100.

10 **Ridao, P., Batlle, J.,** and **Carreras, M.** O2CA2, a new object oriented control architecture for autonomy: the reactive layer. *Control Engng Practice*, 2002, **10**(8), 857–873.

11 **Oka, T., Tashiro, J.,** and **Takase, K.** Object-oriented BeNet programming for data-focused bottom-up design of autonomous agents. *Robotics Autonomous Systems*, 1999, **28**, 127–139.

12 **Balarin, F., Lavagno, L., Passerone, C., Sangiovanni-Vincentelli, A., Sgroi, M.,** and **Watanabe, Y.** Modeling and Designing Heterogeneous Systems, in *Concurrency and Hardware Design, Lecture Notes in Computer Science*. 2002, Vol. 2549, pp. 228–273.

13 **Pinto, A., Sangiovanni-Vincentelli, A., Carloni, P. L.,** and **Passerone, R.** Interchange formats for hybrid systems: Review and proposal. In *Hybrid Systems: Computation and Control, Lecture Notes in Computer Science*. 2005, Springer-Verlag, Vol. 3414, pp. 526–541.

14 **Silva, B. I., Stursberg, O., Krogh, B. H.,** and **Engell, S.** An assessment of the current status of algorithmic approaches to the verification of hybrid systems. In *Decision and Control, 2001*. Proceedings of the *40th IEEE Conference on*. 2001: Orlando, FL. pp. 2867–2874.

15 **Carloni, L., Benedetto, M. D. D., Pinto, A.,** and **Sangiovanni-Vincentelli, A.** *Modeling Techniques, Programming Languages, Design Toolsets and Interchange Formats for Hybrid Systems*. 2004, Project IST- 2001-38314 COLUMBUS, Design of Embedded Controllers for Safety Critical Systems, WPHS: Hybrid System Modeling.

16 **Lee, E. A.** and **Sangiovanni-Vincentelli, A.** A framework for comparing models of computation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1998, **17**(12), 1217–1229.

17 **Pinto, A., Carloni, L. P., Passerone, R.,** and **Sangiovanni-Vincentelli, A. L.** Interchange semantics for hybrid system models. In Proceedings of *5th International Conference on Mathematical Modeling*, 2006, Vienna, Austria. pp. 1–9.

18 **Carloni, L. P., Passerone, R., Pinto, A.,** and **Sangiovanni-Vincentelli, A. L.** Languages and tools for hybrid systems design. In *Foundations and Trends in Electronic Design Automation*, Vol. 1, No. 1/2 (2006), 1–193, Now Publishers.

19 **Pinto, A., Carloni, L., Passerone, R.,** and **Sangiovanni-Vincentelli, A.** Interchange format for hybrid systems: Abstract semantics, Lecture Notes in computer science. In Proceedings of *Hybrid Systems: Computation and Control*, 2006, Vol. 3927, pp. 491–506.

20 **Weiss, G.** *Multiagent systems: a modern approach to distributed artificial intelligence*, 1999 (The MIT Press, Cambridge).

21 **Wooldridge, M.** *An introduction to multiagent systems*, 2002 (John Wiley, Chichester).

22 **Brooks, R. A.** Elephants don't play chess. *Robotics Autonomous Systems*, 1990, **6**(1&2), 3–15.

23 **Veres, S. M.** *Principles, architectures and trends in autonomous control.* In Proceedings of *Autonomous Agents in Control, 2005 (Ref. No. 2005/10986)*, 2005, pp. 1–9.

24 **Wooldridge, M.** and **Jennings, N.** Intelligent Agents: Theory and Practice. *Knowledge Engng Rev.*, 1995, **10**(2), 115–152.

25 **Agre, P. E.** and **Chapman, D.** Pengi: An Implementation of a Theory of Activity. In Proceedings of the International Joint Conference on *Artificial intelligence (IJCAI)*, AAAI-87, Seattle, Kaufmann, 1987.

26 **Agre, P. E.** and **Chapman, D.** What are plans for? *Robotics Auton. Systems*, 1990, **6**, 17–34.

27 **Coste-Maniere, E.** and **Simmons, R.** Architecture, the backbone of robotic systems. In Proceedings of *Robotics and Automation, ICRA '00, 2000.* pp. 67–72. vol. 1, San Francisco, CA.

28 **Langley, P., Laird, J. E.,** and **Rogers, S.** Cognitive architectures: Research issues and challenges. *Cognitive Systems Res.*, 2009, **10**(2), 141–160.

29 **Veres, S.** Mission capable autonomous control systems in the oceans, in the air and in space. In Proceedings of *Brain-inspired information technology, Brain-IT 2007, SCI 266*, 2009, Springer Verlag, Vol. SCI 266, pp. 1–10.

30 **Lespérance, Y., Levesque, H., Lin, F., Marcu, D., Reiter, R.,** and **Scherl, R.** 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, pp. 331–346.

31 **De Giacomo, G., Lespérance, Y.,** and **Levesque, H. J.** ConGolog, a concurrent programming language based on the situation calculus. *Artif. Intell.*, 2000, **121**(1–2), 109–169.

32 **Fisher, M.** A survey of concurrent MetateM — The language and its applications. In Proceedings of the 1st International Conference on *Temporal Logic* (ICTL), (Eds D. M. Gabbay and H. J. Ohlbach), 1994, Bonn, Germany, pp. 480–505.

33 **Eberbach, E.** Approximate reasoning in the algebra of bounded rational agents. *Int. J. Approximate Reasoning*, 2008, **49**(2), 316–330.

34 **Maes, P.** *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, 1991, MIT Press, p. 200.

35 **Maes, P.** The agent network architecture (ANA). *SIGART Bulletin*, 1991, **2**(4), 115–120.

36 **Rosenschein, S. J.** and **Kaelbling, L. P.** Situated view of representation and control, in *Computational theories of interaction and agency* (Eds P. Agre and S. J. Rosenschein), 1996, MIT Press, Cambridge, Mass, pp. 515–540.

37 **Rosenblatt, J., Williams, S.,** and **Durrant-Whyte, H.** A behavior-based architecture for autonomous underwater exploration. *Inf. Sci.*, 2002, **145**(1–2), 69–87.

38 **Bryson, J. J.** *Intelligence by design : principles of modularity and coordination for engineering complex adaptive agents*, 2001 (Massachusetts Institute of Technology, Boston).

39 **Arkin, R. C.** Behavior-based robotics. *Intelligent robots and autonomous agents*, 1998, Cambridge, Mass.: MIT Press, p. 490.

40 **Byrnes, R. B., MacPherson, D. L., Kwak, S. H., McGhee, R. B.,** and **Nelson, M. L.** An experimental comparison of hierarchical and subsumption software architectures for control of an autonomous underwater vehicle. In Symposium on *Autonomous underwater vehicle technology*, 1992, pp. 135–141, Washington, DC.

41 **Flanagan, C., Toal, D.,** and **Leyden, M.** Subsumption and fuzzy-logic, experiments in behavior-based control of mobile robots. *Int. J. Smart Engng System Des.*, 2003, **5**(3), 161–175.

42 **Leyden, M., Toal, D.,** and **Flanagan, C.** An autonomous mobile robot built to develop and test behaviour based control strategies. In Proceedings of the 7th Mechatronics Forum International Conference, 2000 (Atlanta: Elsevier Science).

43 **Das, S. K.** and **Reyes, A. A.** 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), 167–192.

44 **Levesque, H. J.,** *et al.* Golog: a logic programming language for dynamic domains. *J. Logic Programming*, 1997, **31**, 59–83.

45 **Ferrein, A.** and **Lakemeyer, G.** Logic-based robot control in highly dynamic domains. *Robotics and Auton. Systems*, 2008, **56**(11), 980–991.

46 **Georgeff, M. P.** and **Lansky, A. L.** Reactive reasoning and planning. In *P*roceedings of the Sixth National Conference on *Artificial intelligence (AAAI-87)*, 1987, Seattle, WA.

47 **Rao, A.** Decision procedures for prepositional linear-time belief-desireintention logics. In *Intelligent Agents II Agent Theories, Architectures, and Languages, Lecture Notes in Computer Science*, 1996, Springer, Vol. 1037, pp. 33–48.

48 **Urlings, P.,** *et al.* A future framework for interfacing BDI agents in a real-time teaming environment. *J. Network Computer Applic.*, 2005, **29**(2–3), 105–123.

49 **Brooks, R.** A robust layered control system for a mobile robot. *IEEE J. Robotics Autom.*, 1986, **2**(1), 14–23.

50 **Firby, R. J.** An investigation into reactive planning in complex domains. In Proceedings of the International Joint Conference on *Artificial intelligence (IJCAI),* Seattle, WA, 1987, **1**, 202–206.

51 **Firby, R. J.** Adaptive execution in complex dynamic domains. In Technical report YALEU/CSD/RR #672, 1989, Yale University.

52 **Gat, E.** *Reliable goal-directed reactive control for real-world autonomous mobile robots*, 1991 (Virginia Polytechnic Institute and State University, Blacksburg, Virginia).

53 **Gat, E.** On the role of stored internal state in the control of autonomous mobile robots. *AI Magazine*, 1993, **14**(1), 64–73.

54 **Gat, E.** *ESL:* a language for supporting robust plan execution in embedded autonomous agents. In Proceedings of Aerospace Conference, 1997, IEEE.

55 **Gat, E.** *On Three-Layer Architectures*, in *Artificial intelligence and mobile robots: case studies of successful robot systems* (Eds R. P. B. David Kortenkamp and R. Murphy), 1998, pp. 195–210 (MIT Press, Cambridge, MA).

56 **Müller, J., Pischel, M.,** and **Thiel, M.** Modeling reactive behaviour in vertically layered agent architectures. In *Intelligent Agents*, 1995, Vol. 890, pp. 261–276.

57 **Bovio, E., Cecchi, D.,** and **Baralli, F.** Autonomous underwater vehicles for scientific and naval operations. *A. Rev. Control*, 2006, **30**(2), 117–130.

58 **Evans, J.,** *et al.* Design and evaluation of a reactive and deliberative collision avoidance and escape architecture for autonomous robots. *Auton. Robots*, 2008, **24**(3), 247–266.

59 **Sousa, J. B. d.,** *et al.* A verified hierarchical control architecture for co-ordinated multi-vehicle operations. *Int. J. Adaptive Control Signal Processing*, 2007, **21**(2–3), 159–188.

60 **Foka, A.** and **Trahanias, P.** Real-time hierarchical POMDPs for autonomous robot navigation. *Robotics Auton. Systems*, 2007, **55**(7), 561–571.

61 **Sotelo, M. A.,** *et al.* Low level controller for a POMDP based on WiFi observations. *Robotics and Auton. Systems*, 2007, **55**(2), 132–145.

62 **Pineau, J., Gordon, G.,** and **Thrun, S.** Point-based value iteration: An anytime algorithm for POMDPs. In Proceedings of International Joint Conference on *Artificial intelligence*, 2003, Acapulco, Mexico.

63 **Spaan, M. T. J.** and **Vlassis, N.** Perseus: randomized point-based value iteration for POMDPs. *J. Artif. Intell. Res.*, 2005, **24**, 195–220.

64 **Thrun, S., Burgard, W.,** and **Fox, D.** *Probabilistic robotics*, 2005 (MIT Press, Cambridge, MA).

65 **Belker, T., Beetz, M.,** and **Cremers, A. B.** Learning action models for the improved execution of navigation plans. *Robotics Auton. Systems*, 2002, **38**(3–4), 137–148.

66 **Boutilier, C., Dearden, R.,** and **Goldszmidt, M.** Stochastic dynamic programming with factored representations. *Artif. Intell.*, 2000, **121**(1–2), 49–107.

67 **Dearden, R.** and **Boutilier, C.** Abstraction and approximate decision-theoretic planning. *Artif. Intell.*, 1997, **89**(1–2), 219–283.

68 **Weng, J.** On developmental mental architectures. *Neurocomputing*, 2007, **70**(13–15), 2303–2323.

69 **Bordini, R. H., Hübner, J. F.,** and **Wooldridge, M.** Programming Multi-Agent Systems in AgentSpeak using Jason. *Wiley Series in Agent Technology* (Ed. M. Wooldridge), 2007 (John Wiley & Sons, Ltd).

70 **Georgeff, M.** and **Lansky, A.** Procedural Knowledge. *Proceedings of the IEEE (Special Issue on Knowledge Representation)*, 1986, **74**, 1383–1398.

71 **Dastani, M.** 3APL Platform User Guide. In University of Utrecht, Mehdi Dastani, Utrecht, pp. 1–26.

72 **Veres, S. M., Molnar, L.,** and **Lincoln, N. K.** The Cognitive Agents Toolbox (CAT) for programming autonomous vehicles, in Southampton University e-Print Archives, www.eprints.soton.ac.uk, 2009, Southampton, pp. 1–30.

73 **Shoham, Y.** Agent-oriented programming. *Artif. Intell.*, 1993, **60**(1), 51–92.

74 **Sterritt, R.,** *et al.* Next generation system and software architectures: Challenges from future NASA exploration missions. *Sci. Computer Programming*, 2006, **61**(1), 48–57.

75 **Molnar, L.** and **Veres, S. M.** Verification of autonomous underwater vehicles using formal logic. In Proceedings of European Control Conference, 2009, European Control Council, Budapest.

76 **Veres, S. M.** *Natural Language Programming of Agents and Robotic Devices*, 2008, London: Sysbrain, p. 184.

77 **Veres, S. M.** *Authoring Tool for sEnglish Documents and Reader Agent*, 2008 (SysBrain, London), available from: www.system-english.com London.

78 **Veres, S. M.** and **Molnar, L.** Documents for Intelligent Agents in English. In Proceedings of IASTED Conference on *Artificial intelligence and applications*, 2010, p. 10. IASTED, Innsbruck, Austria.

79 **Veres, S. M.** and **Veres, A. G.** Learning and adaptation of skills in autonomous physical agents. In *IFAC World Congress*, 2008 Seoul, Korea: International Federation of Automatic Control, pp. 15457–15462.

80 **Wang, X., Lu, C.,** and **Gill, C.** FCS/nORB: A feedback control real-time scheduling service for embedded ORB middleware. *Microprocessors and Microsystems*, 2008, **32**(8), 413–424.

81 **Reis, L. P.** and **Lau, N.** FC Portugal Team Description: RoboCup 2000 Simulation League Champion, in *RoboCup 2000* (Eds P. Stone, T. Balch, and G. Kraetzschmar), 2002, Springer-Verlag, Berlin, Heidelberg, Vol. LNAI 2019, pp. 29–40.

82 **Meyer, J., Adolph, R., Stephan, D., Daniel, A., Seekamp, M., Weinert, V.,** and **Visser, U.** Decision-making and tactical behavior with potential fields. In *RoboCup 2002* (Eds G. A. Kaminka, P. U. Lima, and R. Rojas), 2003, Berlin Heidelberg, Vol. LNAI 2752, pp. 304–311.

83 **Kaminka, G. A., Lima, P. U.,** and **Rojas, R.** (Eds) *Robocup 2002*. Vol. LNAI 2752, 2003 (Springer-Verlag, Berlin, Heidelberg).

84 **Stone, P., Balch, T.,** and **Kraetzschmar, G.** (Eds) *Robocup 2000*. Vol. LNAI 2019, 2001 (Springer-Verlag, Berlin, Heidelberg).

85 **Klavins, E.** Programmable self-assembly – control of concurrent systems from bottom up. *IEEE Control Systems Mag.*, 2007(August), 43–56.

86 **Kohno, M., Matsunaga, M.,** and **Anzai, Y.** An adaptive sensor network system for complex environments. *Robotics Auton. Systems*, 1999, **28**(2–3), 115–125.

87 **Nembrini, J., Winfield, A.,** and **Melhuish, C.** Minimalist coherent swarming of wireless networked autonomous robots. In Proceedings of 7th

International Conference on *Simulation of adaptive behaviour SAB 2002*, Edinburgh, UK.

88 **Semsar-Kazerooni, E.** and **Khorasani, K.** Multi-agent team cooperation: A game theory approach. *Automatica*, 2009, **45**(10), 2205–2213.

89 **Behrmann, G., Larsen, K. G., Moller, O., David, A., Pettersson, P.,** and **Wang, Y.** UPPAAL – present and future. In Proceedings of the 40th IEE Conference on *Decision and control*, 2001, pp. 2881–2886, vol. 3.

90 **Ejersbo Jensen, H., Guldstrand Larsen, K.,** and **Skou, A.** *Scaling up Uppaal, in Formal Techniques in Real-Time and Fault-Tolerant Systems*, 2000, pp. 641–678.

91 **Ke, X., Pettersson, P., Sierszecki, K.,** and **Angelov, C.** Verification of COMDES-II Systems Using UPPAAL with Model Transformation. 2007, University of Southern Denmark, MCI Mads Clausen Institute, Technical report.

92 **McMillan, K. L.** The SMV system. User Manual for SMV version 2.5.4, http://www.cs.cmu.edu/~modelcheck/smv.html, Carnegie Mellon University, 2000.

93 **Edmund, M., Clarke, J., Grumberg, O.,** and **Peled, D. A.** *Model Checking*, 1999 (The MIT Press, London, England).

94 **Clarke, E., Grumberg, O., Jha, S., Lu, Y.,** and **Veith, H.** Counterexample-Guided Abstraction Refinement. In *Computer Aided Verification, Lecture Notes in Computer Science*, 2000, Springer, Vol. 1855, pp. 154–169.

95 **Pecheur, C.** and **Simmons, R.** From Livingstone to SMV. In *Formal Approaches to Agent-Based Systems, Lecture Notes In Computer Science*, 2001, Vol. 1871, pp. 103–113.

96 **Qianchuan, Z.** and **Krogh, B. H.** Formal verification of statecharts using finite-state model checkers. *Control Systems Technology, IEEE Transactions on*, 2006, **14**(5), 943–950.

97 **Cimatti, A., Clarke, E., Giunchiglia, F.,** and **Roveri, M.** NUSMV: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer (STTT)*, 2000, **2**(4), 410–425.

98 **Huth, M.** and **Ryan, M.** *Logic in Computer Science, modelling and reasoning about systems*, 2004 (Cambridge University Press).

99 **Miller, S. P., Heimdahl, M. P. E.,** and **Tribble, A. C.** Proving the shalls. In Proceedings of the 12th International FME Symposium FM 2003, Pisa, Italy.

100 **Baier, C.** and **Katoen, J.-P.** *Principles of Model Checking*, 2008 (The MIT Press), p. 975.

101 **Raimondi, F.** and **Lomuscio, A.** Automatic verification of deontic interpreted systems by model checking via OBDDs. *Journal of Applied Logic. Special issue on Logic based agent verification*, 2005, **5**(2), 235–251.

102 **Raimondi, F.** *Model Checking Multi-Agent Systems, in Department of Computer Science*, 2006 (University College London, London) p. 142.

103 **Ezekiel, J.** and **Lomuscio, A.** An automated approach to verifying diagnosability In multi-agent systems. In Proceedings of *Software engineering and formal methods*, 2009, Hanoi, Vietnam.

104 **Molnar, L.** and **Veres, S. M.** System Verification of Autonomous Underwater Vehicles by Model Checking. In *OCEANS 2009-EUROPE, 2009. OCEANS '09*, 2009, Bremen, Germany, pp. 1–10.

105 **Gerth, R., Peled, D., Vardi, M. Y.,** and **Wolper, P.** Simple on-the-fly automatic verification of linear temporal logic, in Proceedings of the Fifteenth IFIP WG6.1 International Symposium on *Protocol specification, testing and verification XV*, 1996 (Chapman &Hall, Ltd).

106 **Holzmann, G. J.** The model checker SPIN. *Software Engineering, IEEE Transactions on*, 1997, **23**(5), 279–295.

107 **Holzmann, G. J.** *The SPIN Model Checker: Primer and Reference Manual*, 2003 (Addison-Wesley).

108 **Long, B., Dingel, J.,** and **Graham, T. C. N.** Experience applying the SPIN model checker to an industrial telecommunications system. In Proceedings of the International Conference on Software Engineering, 2008. ICSE, pp. 693–702.

109 **Brayton, R., Hachtel, G., Sangiovanni-Vincentelli, A., Somenzi, F., Aziz, A., Cheng, S., Edwards, S., Khatri, S., Kukimoto, Y., Pardo, A., Qadeer, S., Ranjan, R., Sarwary, S., Staple, T., Swamy, G.,** and **Villa, T.** VIS: A system for verification and synthesis. In *Computer Aided Verification*, 1996, pp. 428–432.

110 **Alur, R., Henzinger, T. A.,** and **Pei-Hsin, H.** Automatic symbolic verification of embedded systems. *Software Engineering, IEEE Transactions on*, 1996, **22**(3), 181–201.

111 **Lundvall, H., Bunus, P.,** and **Fritzson, P.** *Towards Automatic Generation Of Model Checkable Code From Modelica*, 2004 (Department of Computer and Information Science, Linköping University, Sweden).

112 **Yovine, S.** *Kronos: A verification tool for real-time systems, User's guide, VERIMAG, Centre Equation*, 2002 (Gieres, France).

113 **Daws, C., Olivero, A., Tripakis, S.,** and **Yovine, S.** The tool Kronos, in *Hybrid Systems III*, 1996, pp. 208–219.

114 **Yovine, S.** KRONOS: a verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 1997, **1**(1), 123–133.

115 **Bozga, M., Daws, C., Maler, O., Olivero, A., Tripakis, S.,** and **Yovine, S.** Kronos: A model-checking tool for real-time systems, in *Computer Aided Verification*, 1998, pp. 546–550.

116 **Tardieu, O.** A Deterministic Logical Semantics for Esterel. *Electron. Notes Theor. Comput. Sci.*, 2005, **128**(1), 103–122.

117 **Singh, S.** Design and Verification of CoreConnectTM IP Using Esterel. In *Correct Hardware*

Design and Verification Methods, 2003, pp. 283–288.

118 Closse, E., Poize, M., Pulou, J., Sifakis, J., Venter, P., Weil, D., and **Yovine, S.** TAXYS: A Tool for the Development and Verification of Real-Time Embedded Systems? *Computer Aided Verification*, 2001, pp. 391–395.

119 Tripakis, S. and **Yovine, S.** Timing Analysis and Code Generation of Vehicle Control Software using Taxys. *Electronic Notes in Theoretical Computer Science*, 2001, **55**(2), 277–286.

120 Bertin, V., Poize, M., and **Sifakis, J.** Towards Validated Real-Time Software. In Proceedings of the 12th Euromicro Conference on *Real time systems*, 2000, Stockholm, pp. 157–164.

121 Fabiani, P., *et al.* Autonomous flight and navigation of VTOL UAVs: from autonomy demonstrations to out-of-sight flights. *Aerospace Sci. Technol.*, 2007, **11**(2–3), 183–193.

122 Pisokas, J. and **Nehmzow, U.** Performance comparison of three subsymbolic action planners for mobile robots. *Robotics Auton. Systems*, 2005, **51**(1), 55–67.

123 Jeyaraman, S., *et al.* Kripke modelling approaches of a multiple robots system with minimalist communication: A formal approach of choice. *Int. J. Systems Sci.*, 2006, **37**(6), 339–349.

124 Tsourdos, A., *et al.* Modelling and verification of multiple UAV mission using SMV. In Workshop on *Formal methods for aerospace (FMA)*, (Eds M. Bujorianu and M. Fisher), 2010, pp. 22–33. EPTCS 20.

125 Iglesias, R., Nehmzow, U., and **Billings, S. A.** Model identification and model analysis in robot training. *Robotics Auton. Systems*, 2008, **56**(12), 1061–1067.

126 Lemon, O. and **Nehmzow, U.** The scientific status of mobile robotics: Multi-resolution mapbuilding as a case study. *Robotics Auton. Systems*, 1998, **24**, 5–15.

127 Belta, C., Isler, V., and **Pappas, G. J.** Discrete abstractions for robot motion planning and control in polygonal environments. *IEEE Trans. Robotics*, 2005, **21**(5), 864–874.

128 Fainekos, G. E., Loizou, S. G., and **Pappas, G. J.** Translating Temporal Logic to Controller Specifications. In Proceedings of the 45th IEEE Conference on *Decision and control*, 2006, pp. 899–904, San Diego, California.

129 Kloetzer, M. and **Belta, C.** Temporal Logic Planning and Control of Robotic Swarms by Hierarchical Abstractions. *IEEE Trans. Robotics*, 2007, **23**(2), 320–330.

130 Frazzoli, E., Dahleh, M. A., and **Feron, E.** Maneuver-based motion planning for nonlinear systems with symmetries. *IEEE Trans. Robotics*, 2005, **21**(6), 1077–1091.

131 Belta, C., Bicchi, A., Egerstedt, M., Frazzoli, E., Klavins, E., and **Pappas, G. J.** Symbolic planning and control of robot motion [Grand Challenges of Robotics]. *Robotics & Automation Magazine, IEEE*, 2007, **14**(1), 61–70.

132 Bicchi, A., Marigo, A., and **Piccoli, B.** Feedback encoding for efficient symbolic control of dynamical systems. *IEEE Trans. Autom. Control*, 2006, **51**(6), 987–1002.

133 Apolloni, B., Piccolboni, A., and **Sozio, E.** A hybrid symbolic subsymbolic controller for complex dynamic systems. *Neurocomputing*, 2001, **37**(1–4), 127–163.

134 Pola, G., Girard, A., and **Tabuada, P.** Approximately bisimilar symbolic models for nonlinear control systems. *Automatica*, 2008, **44**(10), 2508–2516.

135 Veres, S. M. and **Lincoln, N. K.** Sliding mode control for agents and humans. In Proceedings of *TAROS'08, Towards autonomous robotic systems*, 2008, Edinburgh.

136 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, pp. 1–10.

137 Dennis, L., Fisher, M., Lizitsa, A., Lincoln, N. K., and **Veres, S. M.** Satellite Control Using Rational Agent Programming, in *IEEE Intelligent Systems Magazine*, 2010, IEEE, US, **25**(3), 92–97.

138 Heintz, F., Kvarnstr, J., and **Doherty, P.** Bridging the sense-reasoning gap: DyKnow – stream-based middleware for knowledge processing. *Advd Engng Inf.*, **24**(1), 14–26.

139 Patrón, P., Lane, D. M., and **Petillot, Y. R.** Interoperability of agent capabilities for autonomous knowledge acquisition and decision making in unmanned platforms. In OCEANS 2009–EUROPE, 2009. OCEANS '09, 2009, Bremen, Germany.

140 Veres, S. M. Autonomous and adaptive control of vehicles in formation – editorial to special issue. *Int. J. Adapt. Control Signal Processing*, 2007, **21**(2–3), 93–94.

141 Liu, H., Brown, D. J., and **Coghill, G. M.** Fuzzy qualitative robot kinematics. *IEEE Trans. Fuzzy Systems*, 2008, **16**(3), 851–862.

142 Prescott, D. R., Remenyte-Prescott, R., and **Andrews, J. D.** A system reliability approach to decision making in autonomous multi-platform systems operating phased missions. In Proceedings of Conference on RAMS 2008, *Reliability, availability and maintainability*, 2008, Las Vegas, USA.

143 Dickerson, C. E. and **Mavris, D. N.** Architecture and principles of systems engineering, *Complex and enterprise systems engineering*, 2009 (CRC/Auerbach Publications, New York).

144 Alexander, R. D., Hall-May, M., and **Kelly, T. P.** 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).

145 Despotou, G., Alexander, R., and **Kelly, T.** Addressing challenges of hazard analysis in sys-

tems of systems. In Draft paper, Department of Computer Science, 2007 (University of York, York, UK).

146 **Alexander, R., Kelly, T.,** and **Herbert, N.** Deriving safety requirements for autonomous systems. In *4th SEAS DTC Technical Conference*, 2009, UK MoD, Edinburgh.

147 **Habib, M. K.** Robot Control Architecture and Autonomous Navigation Strategies. In *International Symposium on Robotics*, 2002, Stockholm, Sweden, pp. 17–22.

148 **Carreras, M.** *A proposal of a behavior based control architecture with reinforcement learning for an autonomous underwater robot*, PhD Thesis, Department of Electronics, Informatics and Automation. 2003, University of Girona.

149 **Toal, D.** *Sensor driven autonomous robots from experimental wheeled robots toward intervention AUVs*, PhD thesis, University of Limerick, Limerick, 2004.

150 **Molnar, L.** A hybrid control architecture development for the guidance, navigation and control of the tethra prototype submersible vehicle. PhD thesis, Department of Electronics and Computer Engineering, 2006, University of Limerick: Limerick.

151 **Ridao, P., Yuh, J., Batlle, J.,** and **Sugihara, K.** On AUV control architecture. In Proceedings of *Intelligent Robots and Systems*, 2000 (IROS 2000). 2000 IEEE/RSJ International Conference on. 2000, Takamatsu, Japan, pp. 855–860.

152 **Healey, A. J., Pascoal, A. M.,** and **Pereira, F. L.** Autonomous underwater vehicles: An application of intelligent control technology. In Proceedings of the American Control Conference, 1995, Seattle, Washinton, pp. 2943–2949.

153 **Valavanis, K. P.,** *et al.* Control architectures for autonomous underwater vehicles. *IEEE Control Systems Mag.*, 1997, **17**(6), 48–64.

154 **Ridao, P., Batlle, J., Amat, J.,** and **Roberts, G. N.** Recent trends in control architectures for autonomous underwater vehicles, *International Journal of Systems Science*, 1999, pp. 1033–1056.

155 **Mahyuddin, M. N.** and **Arshad, M. R.** Classes of control architectures for AUV: a brief survey. In Proceedings of the 2nd International Conference on *Underwater system technology*, Bali Indonesia, 2008.

156 **Carreras, M., Batlle, J., Ridao, P.,** and **Roberts, G. N.** An overview of behaviour based methods for AUV control. In Proceedings of the 5th IFAC Conference on *Manoeuvring and control of marine crafts*, 2000, Aalborg, Denmark (Elsevier).

157 **Kim, T. W.** and **Yuh, J.** Development of a real-time control architecture for a semi-autonomous underwater vehicle for intervention missions. *Control Engng Practice*, 2004, **12**(12), 1521–1530.

158 **Yuh, J.** Design and Control of autonomous underwater robots: a survey. *Auton. Robots*, 2000, **8**(1), 7–24.

159 **Bellingham, J. G., Beaton, R., Triantafyllou, M.,** and **Shupe, L.** An Autonomous Submersible Designed for Software Development. In Proceedings of *OCEANS* '89, Seattle, WA, vol. 3, pp. 799–803.

160 **Bellingham, J. G., Consi, T. R., Beaton, R. M.,** and **Hall, W.** Keeping layered control simple [autonomous underwater vehicles]. In Proceedings of *Autonomous Underwater Vehicle Technology*, 1990, pp. 3–8, Washington, DC.

161 **Healey, A. J., Marco, D. B.,** and **McGhee, R. B.** Autonomous underwater vehicle control coordination using a tri-level hybrid software architecture. In Proceedings of the IEEE International Conference on *Robotics and automation*, 1996, Minneapolis, MI, pp. 2149–2159.

162 **McPhail, S. D.** and **Pebody, M.** Navigation and control of an autonomous underwater vehicle using a distributed, networked, control architecture underwater technology. *Int. J. Soc. Underwater*, 1998, **23**(12), 19–30.

163 **Pebody, M.** Autonomous underwater vehicle collision avoidance for under-ice exploration, *Proc. IMechE, Part M: J. Engineering for the Maritime Environment*, 2008, **222**(2), 53–66.

164 **Carreras, M., Batlle, J.,** and **Ridao, P.** Hybrid coordination of reinforcement learning-based behaviors for AUV control. In Proceedings of 2001 IEEE/RSJ International Conference on *In telligent robots and systems*, 2001, pp. 1410–1415, vol. 3, Maui, HI.

165 **Caccia, M.** and **Veruggio, G.** Guidance and control of a reconfigurable unmanned underwater vehicle. *Control Engng Practice*, 2000, **8**(1), 21–37.

166 **Caccia, M., Bruzzone, G.,** and **Veruggio, G.** Bottom-following for remotely operated vehicles: algorithms and experiments. *Auton. Robots*, 2003, **14**(1), 17–32.

167 **Leyden, M., Toal, D.,** and **Flanagan, C.** A fuzzy logic based navigation system for a mobile robot. In Automatisierungssymposium, Wismar, 1999.

168 **Molnar, L., Omerdic, E.,** and **Toal, D.** Guidance, navigation and control system for the Tethra unmanned underwater vehicle. *Int. J. Control*, 2007, **80**(7), 1050–1076.

169 **Hai-bo, L., Guo-chang, G., Jing, S.,** and **Yan, F.** AUV fuzzy neural BDI. *Journal of Marine Science and Application*, 2005, **4**(3), 37–41.

170 **Ortiz, A., Antich, J.,** and **Oliver, G.** A PFM-based control architecture for a visually guided underwater cable tracker to achieve navigation in troublesome scenarios. *J. Maritime Res.: JMR*, 2005, **2**(1), 18.

171 **Tangirala, S., Kumar, R., Bhattacharyya, S., O'Connor, M.,** and **Holloway, L. E.** Hybridmodel based hierarchical mission control architecture for autonomous underwater vehicles. In Proceedings of American Control Conference, 2005, Portland, OR, pp. 668–673.

172 **O'Connor, M., Tangirala, S., Kumar, R., Bhattacharyya, S., Sznaier, M.,** and **Holloway, L. E.** 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.

173 **Zheping, Y.** and **Hou, S.** A coordinated method based on hybrid intelligent control agent for multi-AUVs Control. In Proceedings of IEEE International Conference on *Information acquisition*, 2006, Weihai, China, pp. 1179–1184.

174 **McGann, C., Py, F., Rajan, K., Ryan, J.,** and **Henthorn, R.** Adaptive control for autonomous underwater vehicles. In Proceedings of the 23rd national conference on *Artificial intelligence*, Volume 3, 2008, (AAAI Press, Chicago, Illinois).

175 **McGann, C., Py, F., Rajan, K., Thomas, H., Henthorn, R.,** and **McEwen, R.** A deliberative architecture for AUV control. In Proceedings of the IEEE International Conference on Robotics and Automation, 2008, ICRA Pasadena, CA, pp. 1049–1054.

176 **Frank, J.** and **Jonsson, A.** Constraint-based attribute and interval planning. *J. Constraints*, 2003, **8**(4), 339–364.

177 **Aguiary, A., Almeiday, J., Bayaty, M., Cardeiray, B., Cunhay, R., Hauslery, A., Mauryay, P., Oliveiray, A., Pascoaly, A., Pereira, A., Rufinoy, M., Sebastiaoy, L., Silvestrey, C.,** and **Vanniy, F.** Cooperative Autonomous Marine Vehicle motion control in the scope of the EU GREX Project: Theory and Practice. In *OCEANS 2009-EUROPE, 2009. OCEANS '09*, 2009, Bremen, Germany, pp. 1–10.

178 **Fiorelli, E., Naomi Ehrich, L., Pradeep, B., Derek, A. P., Ralf, B.,** and **David, M. F.** Multi-AUV control and adaptive sampling in Monterey Bay. *IEEE Journal of Oceanic Engineering*, 2006, **31**(4), 935–948.

179 **Paley, D. A., Zhang, F.,** and **Leonard, N. E.** Cooperative control for ocean sampling: The Glider Coordinated Control System. *IEEE Transactions on Control Systems Technology*, 2008, **16**(4), 735–744.

180 **Schofield, O., Kohut, J., Aragon, D., Creed, L., Graver, J., Haldeman, C., Kerfoot, J., Roarty, H., Jones, C., Webb, D.,** and **Glenn, S.** Slocum Gliders: Robust and ready: Research Articles. *J. Field Robot.*, 2007, **24**(6), 473–485.

181 **Hegrenæs, Ø., Berglund, E.,** and **Hallingstad, O.** Model-aided inertial navigation for underwater vehicles. In IEEE Conference on *Robotics and automation, ICRA-08*, Pasadena, CA, USA, 2008.

182 **Hegrenaes, O., Hallingstad, O.,** and **Jalving, B.** A comparison of mathematical models for the HUGIN 4500 AUV based on experimental data. In Proceedings of 15th International Symposium on *Unmanned untethered submersible technology*, UT'07. Tokyo, 2007.

183 **Hagen, P. E.** AUV/UUV mission planning and real time control with the HUGIN operator system. In Proceedings of *Oceans 2001*. Honolulu, HI, USA, 2001.

184 **Bebel, J. C., Howard, N.,** and **Patel, T.** An autonomous system used in the DARPA grand challenge. In Proceedings of 7th International IEEE Conference on *Intelligent transportation systems*. Washington DC, 2004, pp. 487–490.

185 **Beal, B. J.** Technical paper for BJB Engineering, Team D112, In *DARPA grand challenge reports*, BJB Engineering, Willoughby Hills, 2005, pp. 1–8.

186 **Kent, T. J.** DARPA grand challenge technical paper. In *DARPA grand challenges technical reports*, Las Vegas, 2004, pp. 1–10.

187 **Buehler, M., Iagnemma, K.,** and **Singh, S.** (Eds) *The DARPA Urban Challenge, Autonomous Vehicles in City Traffic. Springer Tracts in Advanced Robotics.* vol. 56, 2010, p. 626 (Springer, Berlin).

188 **Bohren, J., Foote, T., Keller, J., Kushleyev, A., Lee, D., Stewart, A., Vernaza, P., Derenick, J., Spletzer, J.,** and **Satterfield, B.** 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, pp. 231–255.

189 **Chen, Y.-L., Sundareswaran, V., Anderson, C., Broggi, A., Grisleri, P., Porta, P., Zani, P.,** and **Beck, J.** TerraMax: Team Oshkosh Urban Robot. In *The DARPA Urban Challenge, Springer Tracts in Advanced Robotics Series*, 2009, Springer, Berlin/Heidelberg, Vol. 56/2009, pp. 595–622.

190 **Hundelshausen, F., Himmelsbach, M., Hecker, F., Mueller, A.,** and **Wuensche, H.-J.** Driving with tentacles – integral structures for sensing and motion. In *The DARPA Urban Challenge, Springer Tracts in Advanced Robotics Series*, 2009, Springer, Berlin/Heidelberg, Vol. 56/2009, pp. 393–440.

191 **Kammel, S., Ziegler, J., Pitzer, B., Werling, M., Gindele, T., Jagzent, D., Schöder, J., Thuy, M., Goebl, M., von Hundelshausen, F., Pink, O., Frese, C.,** and **Stiller, C.** Team AnnieWAY's autonomous system for the DARPA Urban Challenge 2007. In *The DARPA Urban Challenge, Springer Tracts in Advanced Robotics Series*, 2009, Springer, Berlin/Heidelberg, Vol. 56/2009, pp. 359–391.

192 **Leonard, J., How, J., Teller, S., Berger, M., Campbell, S., Fiore, G., Fletcher, L., Frazzoli, E., Huang, A., Karaman, S., Koch, O., Kuwata, Y., Moore, D., Olson, E., Peters, S., Teo, J., Truax, R., Walter, M., Barrett, D., Epstein, A., Maheloni, K., Moyer, K., Jones, T., Buckley, R., Antone, M., Galejs, R., Krishnamurthy, S.,** and **Williams, J.** A perception-driven autonomous urban vehicle. In *The DARPA Urban Challenge, Springer Tracts in Advanced Robotics Series*, 2009, Springer, Berlin/Heidelberg, Vol. 56/2009, pp. 163–230.

193 **McBride, J., Ivan, J., Rhode, D., Rupp, J., Rupp, M., Higgins, J., Turner, D.,** and **Eustice, R.** A perspective on emerging automotive safety applications, derived from lessons learned through participation in the DARPA Grand Challenges. In *The DARPA Urban Challenge, Springer Tracts in*

Advanced Robotics Series, 2009, Springer, Berlin/Heidelberg, Vol. 56/2009, pp. 549–593.

194 **Miller, I., Campbell, M., Huttenlocher, D., Nathan, A., Kline, F.-R., Moran, P., Zych, N., Schimpf, B., Lupashin, S., Garcia, E., Catlin, J., Kurdziel, M.,** and **Fujishima, H.** 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, pp. 275–304.

195 **Patz, B., Papelis, Y., Pillat, R., Stein, G.,** and **Harper, D.** A practical approach to robotic design for the DARPA Urban Challenge. In *The DARPA Urban Challenge, Springer Tracts in Advanced Robotics Series*, 2009, Springer, Berlin/Heidelberg, Vol. 56/2009, pp. 305–358.

196 **Cremer, J., Kearney, J.,** and **Papelis, Y.** HCSM: a framework for behavior and scenario control in virtual environments. *ACM Trans. Model. Comput. Simulation*, 1995, **5**(3), 242–267.

197 **Papelis, Y.** and **Ahmad, O.** A comprehensive microscopic autonomous driver model for use in high-fidelity driving simulation environments. In Proceedings of the Annual Transportation Research Board Meeting, Washington, DC, 2001.

198 **Michon, J.** A critical view of driver behaviour models: What do we know, what should we do? In *Human behaviour and traffic safety* (Eds L. Evans and R. Schwing), 1985 (Plenum, New York).

199 **Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., Huhnke, B., Johnston, D., Klumpp, S., Langer, D., Levandowski, A., Levinson, J., Marcil, J., Orenstein, D., Paefgen, J., Penny, I., Petrovskaya, A., Pflueger, M., Stanek, G., Stavens, D., Vogt, A.,** and **Thrun, S.** Junior: The Stanford Entry in the Urban Challenge. In *The DARPA Urban Challenge, Springer Tracts in Advanced Robotics Series*, 2009, Springer, Berlin/Heidelberg, Vol. 56/2009, pp. 91–123.

200 **Rauskolb, F., Berger, K., Lipski, C., Magnor, M., Cornelsen, K., Effertz, J., Form, T., Graefe, F., Ohl, S., Schumacher, W., Wille, J.-M., Hecker, P., Nothdurft, T., Doering, M., Homeier, K., Morgenroth, J., Wolf, L., Basarke, C., Berger, C., Gülke, T., Klose, F.,** and **Rumpe, B.** Caroline: An autonomously driving vehicle for urban environments. In *The DARPA Urban Challenge, Springer Tracts in Advanced Robotics Series*, 2009, Springer, Berlin/Heidelberg, Vol. 6/2009, pp. 441–508.

201 **Rosenblatt, J. K.** DAMN: a distributed architecture for mobile navigation. *Journal of Experimental & Theoretical Artificial Intelligence*, 1997, **9**(2), 339–360.

202 **Reinholtz, C., Hong, D., Wicks, A., Bacha, A., Bauman, C., Faruque, R., Fleming, M., Terwelp, C., Alberi, T., Anderson, D., Cacciola, S., Currier, P., Dalton, A., Farmer, J., Hurdus, J., Kimmel, S., King, P., Taylor, A., Van Covern, D.,** and **Webster, M.** 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, pp. 125–162.

203 **Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Clark, M., Dolan, J., Duggins, D., Galatali, T., Geyer, C., Gittleman, M., Harbaugh, S., Hebert, M., Howard, T., Kolski, S., Kelly, A., Likhachev, M., McNaughton, M., Miller, N., Peterson, K., Pilnick, B., Rajkumar, R., Rybski, P., Salesky, B., Seo, Y.-W., Singh, S., Snider, J., Stentz, A., Whittaker, W., Wolkowicki, Z., Ziglar, J., Bae, H., Brown, T., Demitrish, D., Litkouhi, B., Nickolaou, J., Sadekar, V., Zhang, W., Struble, J., Taylor, M., Darms, M.,** and **Ferguson, D.** Autonomous driving in urban environments: Boss and the urban challenge. In *The DARPA Urban Challenge, Springer Tracts in Advanced Robotics Series*, 2009, Springer, Berlin/Heidelberg, Vol. 56/2009, pp. 1–59.

204 **Fregene, K., Kennedy, D., Madhavan, R., Parker, L. E.,** and **Wang, D.** A class of intelligent agents for coordinated control of outdoor terrain mapping UGVs. *Engineering Applications of Artificial Intelligence*, 2005, **18**(5), 513–531.

205 **Girault, A.** A hybrid controller for autonomous vehicles driving on automated highways. *Transp. Res. Part C, Emerging Technol.*, 2004, **12**(6), 421–452.

206 **Posadas, J. L., Poza, J. L., Simo, J. E., Benet, G.,** and **Blanes, F.** Agent-based distributed architecture for mobile robot control. *Engineering Applications of Artificial Intelligence*, 2008, **21**(6), 805–823.

207 **Vis, I. F. A.** Survey of research in the design and control of automated guided vehicle systems. *Eur. J. Opl Res.*, 2006, **170**(3), 677–709.

208 **Consolini, L., Morbidi, F., Prattichizzo, D.,** and **Tosques, M.** Leader-follower formation control of nonholonomic mobile robots with input constraints. *Automatica*, 2008, **44**(5), 1343–1349.

209 **Purwin, O., D'Andrea, R.,** and **Lee, J.–W.** Theory and implementation of path planning by negotiation for decentralized agents. *Robotics Auton. Systems*, 2008, **56**(5), 422–436.

210 **Raimondi, F. M.** and **Melluso, M.** Fuzzy motion control strategy for cooperation of multiple automated vehicles with passengers comfort. *Automatica*, 2008, **44**(11), 2804–2816.

211 **Kolp, M., Giorgini, P.,** and **Mylopoulos, J.** Multi-agent architectures as organizational structures. *Auton. Agent sand Multi-Agent Systems*, 2006, **13**(1), 3–25.

212 **Ono, Y., Uchiyama, H.,** and **Potter, W.** A mobile robot for corridor navigation: a multi-agent approach. In Proceedings of the 42nd Annual Southeast Regional Conference. ACM, Huntsville, Alabama, 2004.

213 **Innocenti, B., López, B.,** and **Salvi, J.** A multi-agent architecture with cooperative fuzzy control for a mobile robot. *Robotics and Autonomous*

Systems in the 50th Anniversary of Artificial Intelligence, Campus Multidisciplinary in Perception and Intelligence, 2007, **55**(12), 881–891.

214 **Soh, L.-K.** and **Tsatsoulis, C.** A real-time negotiation model and a multi-agent sensor network implementation. *Auton. Agents and Multi-Agent Systems*, 2005, **11**(3), 215–271.

215 **Do, K. D.** and **Pan, J.** Nonlinear formation control of unicycle-type mobile robots. *Robotics Auton. Systems*, 2007, **55**(3), 191–204.

216 **Feder, H. J. S., Leonard, J. J.,** and **Smith, C. M.** Adaptive Mobile Robot Navigation and Mapping. *International Journal of Robotics Research*, 1999, **18**(7), 650–668.

217 **Morice, C.** and **Veres, S. M.** Geometric bounding techniques for underwater navigation. In Proceedings of SYSID'09, IFAC Symposium on *System identification*. 2009, Saint-Malo: IFAC, p. 6.

218 **Kim, H. J.** and **Shim, D. H.** A flight control system for aerial robots: algorithms and experiments. *Control Engng Practice*, 2003, **11**(12), 1389–1400.

219 **Saffarian, M.** and **Fahimi, F.** Non-iterative nonlinear model predictive approach applied to the control of helicopters' group formation. *Robotics Auton. Systems*, 2009, **57**(6–7), 749–757.

220 **Yu, X.-l., Sun, Y.-r., Liu, J.-y.,** and **Chen, B.-w.** Autonomous navigation for unmanned aerial vehicles based on chaotic bionics theory. *Journal of Bionic Engineering*, 2009, **6**(3), 270–279.

221 **Zemalache, K. M.** and **Maaref, H.** Controlling a drone: comparison between a based model method and a fuzzy inference system. *Appl. Soft Computing*, 2009, **9**(2), 553–562.

222 **Budiyono, A.** and **Wibowo, S. S.** Optimal tracking controller design for a small scale helicopter. *J. Bionic Engng*, 2007, **4**(4), 271–280.

223 **Shiyu, Z.** and **Rui, Z.** Cooperative guidance for multimissile salvo attack. *Chin. J. Aeronautics*, 2008, **21**(6), 533–539.

224 **Kim, Y., Gu, D.-W.,** and **Postlethwaite, I.** Real-time path planning with limited information for autonomous unmanned air vehicles. *Automatica*, 2008, **44**(3), 696–712.

225 **Al-Jarrah, M. A.** and **Hasan, M. M.** HILS setup of dynamic flight path planning in 3D environment with flexible mission planning using Ground Station. *Journal of the Franklin Institute*, In Press, Corrected Proof, Available, ISSN 0016-0032.

226 **Heredia, G., Ollero, A., Bejar, M.,** and **Mahtani, R.** Sensor and actuator fault detection in small autonomous helicopters. *Mechatronics*, 2008, **18**(2), 90–99.

227 **Iida, F.** Biologically inspired visual odometer for navigation of a flying robot. *Robotics Auton. Systems*, 2003, **44**(3–4), 201–208.

228 **Pudas, M., Viollet, S., Ruffier, F., Kruusing, A., Amic, S., Leppävuori, S.,** and **Franceschini, N.** A miniature bio-inspired optic flow sensor based on low temperature co-fired ceramics (LTCC) technology. *Sensors and Actuators A: Physical*, 2007, **133**(1), 88–95.

229 **Aubépart, F.** and **Franceschini, N.** Bio-inspired optic flow sensors based on FPGA: application to micro-air-vehicles. *Microprocessors and Microsystems*, 2007, **31**(6), 408–419.

230 **Tisse, C.-L., Durrant-Whyte, H.,** and **Hicks, R. A.** An optical navigation sensor for micro aerial vehicles. *Computer Vision and Image Understanding*, 2007, **105**(1), 21–29.

231 **Bachmann, R. J., Boria, F. J., Vaidyanathan, R., Ifju, P. G.,** and **Quinn, R. D.** A biologically inspired micro-vehicle capable of aerial and terrestrial locomotion. *Mechanism and Machine Theory*, 2009, **44**(3), 513–526.

232 **Duan, H.-b., Zhang, X.-y., Wu, J.,** and **Ma, G.-j.** Max-min adaptive ant colony optimization approach to multi-UAVs coordinated trajectory replanning in dynamic and uncertain environments. *Journal of Bionic Engineering*, 2009, **6**(2), 161–173.

233 **Liu, C. a., Wang, L.,** and **Liu, C.** Mission planning of the flying robot for powerline inspection. *Progr. Natural Sci.*, 2009, **19**(10), 1357–1363.

234 **Chen, J., Dawson, D. M., Salah, M.,** and **Burg, T.** Cooperative control of multiple vehicles with limited sensing. *International Journal of Adaptive Control and Signal Processing*, 2007, **21**(2–3), 115–131.

235 **Metni, N., Pflimlin, J.-M., Hamel, T.,** and **Souères, P.** Attitude and gyro bias estimation for a VTOL UAV. *Control Engineering Practice*, 2006, **14**(12), 1511–1520.

236 **Xie, S.-R., Luo, J., Rao, J.-J.,** and **Gong, Z.-B.** Computer vision-based navigation and predefined track following control of a small robotic airship. *Acta Automatica Sinica*, 2007, **33**(3), 286–291.

237 **Barnes, D., Shaw, A., Summers, P., Ward, R., Woods, M., Evans, M., Paar, G.,** and **Sims, M.** Imaging and localisation software demonstrator for planetary aerobots. *Acta Astronautica*, 2006, **59**(8–11), 1062–1070.

238 **Blanc, M., Alibert, Y., André, N., Atreya, S., Beebe, R., Benz, W., Bolton, S. J., Coradini, A., Coustenis, A., Dehant, V., Dougherty, M., Drossart, P., Fujimoto, M., Grasset, O., Gurvits, L., Hartogh, P., Hussmann, H., Kasaba, Y., Kivelson, M., Khurana, K., Krupp, N., Louarn, P., Lunine, J., McGrath, M., Mimoun, D., Mousis, O., Oberst, J., Okada, T., Pappalardo, R., Prieto-Ballesteros, O., Prieur, D., Regnier, P., Roos-Serote, M., Sasaki, S., Schubert, G., Sotin, C., Spilker, T., Takahashi, Y., Takashima, T., Tosi, F., Turrini, D., van Hoolst, T.,** and **Zelenyi, L.** LAPLACE: A mission to Europa and the Jupiter System for ESA's Cosmic Vision Programme. *Experimental Astronomy*, 2009, **23**(3), 849–892.

239 **D'Arrigo, P.** and **Santandrea, S.** APIES: A mission for the exploration of the main asteroid belt using a swarm of microsatellites. *Acta Astronautica*, 2006, **59**(8–11), 689–699.

240 Ollivier, M. Towards the spectroscopic analysis of Earthlike planets: the DARWIN/TPF project. *Comptes Rendus Physique*, 2007, **8**(3–4), 408–414.

241 Noor, A. K., Doyle, R. J., and **Venneri, S. L.** Autonomous, biologically inspired systems for future space missions. *Adv. Engng Software*, 2000, **31**(7), 473–480.

242 Rayman, M. D., Varghese, P., Lehman, D. H., and **Livesay, L. L.** Results from the Deep Space 1 Technology Validation Mission. *Acta Astronautica*, 2000, **47**, 475–487.

243 Muscettola, N., Nayak, P. P., Pell, B., and **Williams, B. C.** Remote Agent: to boldly go where no AI system has gone before. *Artificial Intelligence*, 1998, **103**(1–2), 5–47.

244 Chien, S., Sherwood, R., Rabideau, G., Castano, R., Davies, A., Burl, M., Knight, R., Stough, T., Roden, J., Zetocha, P., Wainwright, R., Klupar, P., Gaasbeck, J. V., Cappelaere, P., and **Oswald, D.** The Techsat-21 autonomous space science agent. In Proceedings of the 1st international joint conference on *Autonomous agents and multiagent systems: part 2*, 2002, ACM, Bologna, Italy, pp. 570–577.

245 Thanapalan, K. K. T. and **Veres, S. M.** Agent based controller for satellite formation flying. In Proceedings of the 2005 International Conference on *Intelligent sensors, sensor networks and information processing*, 2005 Melbourne, Australia, pp. 385–389.

246 Veres, S. M. and **Luo, J.** A class of BDI agent architectures for autonomous control. In Proceedings of CDC 43rd IEEE Conference on *Decision and Control*, 2004, Paradise Islands, Bahamas, pp. 4746–4751.

247 Bonnet, G. and **Tessier, C.** Collaboration among a satellite swarm. In Proceedings of the 6th International Joint Conference on *Autonomous agents and multiagent systems*. ACM: Honolulu, Hawaii, 2007, pp. 1–8.

248 Izzo, D. and **Pettazzi, L.** Self-assembly of large structures in space using intersatellite Coulomb forces. In Proceedings of 57th International Astronautical Congress, Valencia, Spain, 2006.

249 Izzo, D. and **Pettazzi, L.** Autonomous and distributed motion planning for satellite swarm. *J. Guidance Control and Dynamics*, 2007, **30**(2), 449–459.

250 Izzo, D., Pettazzi, L., and **Ayre, M.** Mission concept for autonomous on orbit assembly of a large reflector in space. In Proceedings of 56th International Astronautical Congress, Fukuoka, Japan, 2005.

251 Tripp, H. and **Palmer, P.** Stigmergy based behavioural coordination for satellite clusters. *Acta Astronautica*, 2009, **66**(7–8), 1052–1071.

252 Pei, C., Zongji, C., Rui, Z., and **Chen, W.** Autonomous control reconfiguration of aerospace vehicle based on control effectiveness estimation. *Chinese Journal of Aeronautics*, 2007, **20**(5), 443–451.

253 Ren, W. Distributed attitude alignment in spacecraft formation flying. *Int. J. Adaptive Control Signal Processing*, 2007, **21**(2–3), 95–113.

254 Lincoln, N. K. and **Veres, S. M.** Components of a vision assisted constrained autonomous satellite formation flying control system. *Int. J. Adaptive Control Signal Processing*, 2007, **21**(2–3), 237–264.

255 Pettersson, O. Execution monitoring in robotics: a survey. *Robotics Auton. Systems*, 2005, **53**(2), 73–88.

256 Wolf, J. C. and **Bugmann, G.** Understanding rules in human–robot instructions. In Proceedings of the 16th IEEE International Symposium on *Robot and human interactive communication, RO-MAN'07*. Jeju Island, Korea, 2007.