Universitat de Girona

A PROPOSAL OF A BEHAVIOR-BASED CONTROL ARCHITECTURE WITH REINFORCEMENT LEARNING FOR AN AUTONOMOUS UNDERWATER ROBOT

Marc CARRERAS PÉREZ

ISBN: 84-688-5254-6
Dipòsit legal: GI-48-2004

University of Girona

Department of Electronics, Informatics and Automation

PhD Thesis

A Proposal of a Behavior-based Control Architecture with Reinforcement Learning for an Autonomous Underwater Robot

Thesis presented by

Marc Carreras Pérez,

to obtain the degree of:
PhD in Computer Engineering.

Supervisor: Dr. Pere Ridao Rodríguez

Girona, May 2003

To my family, and especially, to Pepa.

Acknowledgements

I would like to express in these lines my gratitude to those people who have helped me in the accomplishment of this PhD thesis.

I must start by acknowledging my first director, Joan Batlle, for pushing me some years ago into the exciting field of research. I thank him for transmitting to me so much enthusiasm and motivation. At the same time, I would like to thank my current director and friend Pere Ridao, for helping me with the ideas and experiments of this dissertation. I appreciate his patience in listening to all my questions and problems and his effort in revising all my work.

I want also to express my gratitude to the supervisors who have welcomed me at their research centers. To Prof. Geoff Roberts from the University of Wales College, Newport, for his good advice when I was there at the beginning of this research. Also, to Prof. Junku Yuh from the University of Hawaii, for guiding me through the achievements of this thesis and for allowing me to perform my experiments in his laboratory.

I would like also to mention other researchers for their collaboration and kindness; Prof. Antonio Tiano from the University of Pavia, and Dr. Biel Oliver and his team from the University of the Balearic Islands.

I must also thank all the people of the *Computer Vision and Robotics* research group and the people of the department of *Electronics, Informatics and Automation* for creating a pleasant work environment. I would like to extend my warmest thanks to JordiF, Rafa, Xevi, Quim, JoanM and PepF, I really appreciate the friendship you extended. My special gratitude to the PhD students sharing the laboratory with me: Jessica, Tudor, Zoran, Dino, David and Andres. Thanks for your daily help and humor. And my gratitude to the other colleagues of the group: XMangui, XMuñoz, XLlado, Ela, JordiP, Radu and Carles, for their support and friendship.

My very special gratitude to the invaluable support of Josep Tomas, Lluis Magi and Toni for technical aspects, to Isela and Sonia for administrative affairs, and to Christine for revising my English. Thanks to all for attending my requests in such a short time!

A general acknowledgement goes to all the undergraduate students who have worked in the laboratory on our research projects. Among them I must mention Ferran and OscarP who contributed directly to the development of the underwater robot URIS.

I cannot forget the friends I met during my two stays in Newport and Hawaii. I must thank Ioannis and Vivianne for their hospitality when I was in Newport. Also, the people of the *Mechatronics Research Centre* for teaching me how to prepare some cakes! From my stay in Hawaii, I must acknowledge my good friend Alberto, who helped me to discover the most amazing places on the island. Also, Ursula and Sergio for their great friendship. I'm sure the success of that stay was in part because of you. Finally, my gratitude to all the people of the *Autonomous Systems Laboratory* of the University of Hawaii, for taking me in so kindly and for helping me with the experiments. My special gratitude to Hyun Taek, Side, Scott, Kaikala, Giacomo, Mick, Wesley and Song.

And last, but not least, my most especial gratitude is to my parents, to all my family and friends for trusting in me and for their support during all these years. Finally, the most important acknowledgement goes to Pepa for her support and her patience. This thesis is especially dedicated to you.

Contents

List of Figures

List of Tables

Glossary

AFSM	Augmented Finite States Machines
AI	Artificial Intelligence
ASD	Action Selection Dynamics
AUV	Autonomous Underwater Vehicle
BBCA	Behavior-based Control Architectures
CMAC	Cerebellar Model Articulation Controller
DOF	Degree Of Freedom
DP	Dynamic Programming
FMDP	Finite MDP
FSM	Finite State Machine
GA	Genetic Algorithm
HHCN	Hierachical Hybrid Coordination Node
HSL	Hue Saturation Luminance
IDL	Interface Definition Language
MC	Monte Carlo
MDP	Markov Decision Process
NQL	Neural-Q_learning
PDL	Process Description Language
POMDP	Partially Observable MDP
QL	Q_Learning
RDF	Radial Basis Function
RGB	Red Green Blue
RL	Reinforcement Learning
RLP	Reinforcement Learning Problem
ROV	Remotely Operated Vehicle
SMDP	Semi MDP
SONQL	Semi-Online Neural-Q_learning
TD	Temporal Difference
UUV	Unmanned Underwater Vehicle

Chapter 1

Introduction

Technical advances in our society have demonstrated great achievements in many diverse fields like transportation, communication, manufacturing and computation. Machines help us in our daily activities, extending our natural acting capabilities. Also, a diverse set of perception systems allow us to extend our natural senses. However, there are few autonomous devices able to integrate the two concepts: *sensing* and *acting*. *Autonomous robots* is a research subject which gathers topics like sensing, actuation, powering, communication, control theory and artificial intelligence. The goal residing in the design of an autonomous robot is the development of a machine able to fulfill a task in the real world. Tasks which are similar to those humans commonly do without effort. An autonomous robot must be able to act in the world by changing its state or by moving itself through it. It must also be able to sense the state of the world, which requires the interpretation of the sensor information. Finally, it must be able to decide *what to do*, how to relate the state of the environment to its action possibilities in order to achieve a predefined goal.

The state/action relation, which is trivial for human intelligence, represents a very difficult task for a computerized system, specially in unstructured and unknown environments. To overcome this, two different problems have to be dealt with. The first is the *state interpretation* problem, that is, the understanding of what the sensors perceive. The second is the *action-decision* problem which consists of deciding the movements of the robot to accomplish the mission. To solve the action-decision problem it is assumed that the state has been correctly interpreted. The field of *Artificial Intelligence* is usually applied to autonomous robots for solving both of these problems. However, a feasible solution can be obtained in only a few set of cases.

This thesis is concerned with the field of autonomous robots and the problem of action-decision. The thesis analyzes some Artificial Intelligence

techniques which can be successfully applied to the control of an autonomous robot. In particular, the thesis is based on the topics of *Behavior-based Robotics* and the *Reinforcement Learning* theory. The experimental platform used in this research is an Autonomous Underwater Robot (AUV). Throughout this chapter the main aspects which have conditioned this research project will be overviewed. First, the research antecedents which were found will be presented in the beginning of this thesis. Then, the goal of the thesis will be pointed out. Finally, the chapter will finish with the outline of this dissertation.

1.1 Previous Research Work

The research presented in this thesis has been fulfilled in the *Computer Vision and Robotics* research group of the University of Girona. This group has been doing research in underwater robotics since 1992 (supported by several programs of the Spanish commission MCYT). The main contribution throughout the past years is the development of two *Unmanned Underwater Vehicles* (UUV). The first prototype, called GARBI, was developed in collaboration with the Polytechnical University of Catalonia. This vehicle was conceived as a *Remotely Operated Vehicle* (ROV). The second prototype, URIS, was fully developed in the University of Girona and was designed as an *Autonomous Underwater Vehicle* (AUV).

The design of an autonomous vehicle requires a solution for the action decision problem, which was introduced at the beginning of this chapter. A *Control Architecture* is the part of the robot control system in charge of making decisions. An autonomous robot is designed to accomplish a particular *mission*, and the control architecture must achieve this mission by generating the proper actions. The control architecture is also known as the *High-Level Controller* since it decides the movements to be followed by the robot. Another kind of controller, which is also present in any autonomous robot, is the *Low-Level Controller*. This one will make the movements proposed by the high-level controller and drive the required actuators. The main difference between a ROV and an AUV resides in the control architecture. In a ROV, the low-level controller will make the movements a human proposes by means of a joystick for example. In an AUV, the control system needs the presence of a control architecture or high-level controller to generate these actions.

The design of the autonomous robot URIS and previously, the adaptation of GARBI from a ROV to an AUV, led to the development of a new control architecture. The main feature of this architecture, called O^2CA^2, was the break down of the whole robot control system into a set of objects. The

parallel execution of these objects allowed a real-time execution of the control system. In addition, some of these objects represented primitive behaviors of the robot. The control architecture used these behaviors to propose the best movement at any given moment.

The development of the O^2CA^2 control architecture led to the PhD. thesis of Dr. Pere Ridao [Ridao, 2001]. The thesis presented in this document is a continuation of the work started with the O^2CA^2 architecture. In this thesis, the set of behaviors which constitute the action-decision methodology are studied. The thesis investigates some implementation aspects which influence the overall performance of the robot. To do this, the topic of *Behavior-based Robotics* was surveyed. Also, the thesis explored the use of learning algorithms to improve the efficiency of the behaviors. A learning theory, called *Reinforcement Learning*, was also overviewed and applied. Finally, the proposals of this thesis were tested with the underwater robot URIS.

1.2 Goal of the thesis

After the description of the research antecedents, the goal of this thesis is stated. The general purpose is summarized as:

"The development of a robot control architecture for an AUV able to achieve simple tasks and exhibit real-time learning capabilities"

This goal was selected in order to continue the research line started with the development of the O^2CA^2 control architecture. On the one hand, it had the intention of exploiting this architecture by refining some performance aspects. But on the other hand, it opened the new research field of robot learning, which is one of the most active topics in robotics. The application of learning algorithms in the control architecture was certainly the most important purpose of this dissertation. Finally, the fundamental goal on which this research project was based is the experimentation with real robots in real-time computation. The basic premise was to demonstrate the research advances with real experiments.

The general goal of the thesis can be divided into three more specific points:

Behavior-based control layer. Design of a behavior-based control system which will be contained in the overall control architecture with the purpose of accomplishing simple tasks. A task is intended as one of the phases in which a mission can be divided. It is assumed that the sequential achievement of a set of tasks entails the achievement of the

mission. The behavior-based control system must assure the safety of the robot while demonstrating a high control performance.

Reinforcement Learning-based behaviors. Integration of a reinforcement learning algorithm in the control architecture. This learning theory will be applied to the acquisition of the internal structure of a robot behavior. The purpose of using Reinforcement Learning is to reduce the required human work in the development of a new robot behavior. Instead of implementing the action-decision rules, the designer need only to define the goal of the behavior.

Experimentation with an AUV. Evaluation of the proposed control and learning systems with real experiments using an Autonomous Underwater Vehicle. The feasibility and limitations of these approaches must be experimentally tested with the available systems and resources.

1.3 Outline of the thesis

The contents of this thesis can be divided into three parts. The *first* part overviews the field of Behavior-based Robotics (Chapter 2) and proposes an architecture (Chapter 3). The *second* part overviews the field of Reinforcement Learning (Chapter 4) and proposes an algorithm for its application to robotics (Chapter 5). Finally, the *third* part of the thesis shows the experimental results by first describing the set-up (Chapter 6) and then the results (Chapter 7). A brief description of each chapter is next presented.

Chapter 2 *Behavior-based Control Architectures.* This presents the field of "Behavior-based Robotics" and overviews some classic architectures. A simulated task is used to implement and compare these architectures.

Chapter 3 *Hybrid Coordination of Behaviors.* This proposes a control architecture based on a behavior-based control layer. A Hybrid Coordination methodology is contained in this layer which attempts to guarantee the robustness while providing a high trajectory performance.

Chapter 4 *Reinforcement Learning.* This presents the theory of Reinforcement Learning and overviews the most important research issues among which, the generalization problem is specifically treated for its strong influence in robotics. Some methodologies to deal with this are presented.

Chapter 5 *Semi-Online Neural-Q_learning.* This proposes an algorithm to learn robot behaviors using Reinforcement Learning. The generalization problem is faced with a Neural Network and a database of the most representative learning samples.

Chapter 6 *URIS' Experimental Set-up.* This details the main features of URIS AUV, and all the systems designed to perform the experiments of the SONQL algorithm and the hybrid coordination methodology.

Chapter 7 *Experimental Results.* This shows the experimental results obtained with URIS. The SONQL is tested with a "target following" behavior. The hybrid coordination system is applied to coordinate this behavior with some manually implemented behaviors. Also, the SONQL is tested with a Reinforcement Learning benchmark to demonstrate its generalization capability.

Chapter 8 *Conclusions.* This concludes the thesis by summarizing the work and points out the contributions and future work. It also comments the research evolution and the publications accomplished during this research project.

Chapter 2

Behavior-based Control Architectures

This chapter overviews the field of Behavior-based Control Architectures also known as Behavior-based Robotics [Arkin, 1998]. It first reviews the history of control architectures for autonomous robots, starting with traditional methods of Artificial Intelligence. The most important facts are revised up to the inception of the field of Behavior-based Robotics. The chapter then describes the principles and main features to be found in a behavior-based system. After this general introduction to the field, four representative behavior-based architectures are described as they were originally designed. The architectures are *Subsumption, Action Selection Dynamics, Motor Schema* and *Process Description Language*. In addition to its description, the architectures are also tested with an experimental task, in which an AUV must be controlled. Finally, using the obtained results the chapter analyzes how the coordination methodology influences to the control performance.

2.1 Behavioral-based Robotics versus Traditional AI

The first attempt at building autonomous robots began around the mid-twentieth century with the emergence of Artificial Intelligence. The approach begun at that time is now referred to as "Traditional AI", "Classical AI", "Deliberative approach" or "Hierarchical approach". Traditional AI relies on a centralized world model for verifying sensory information and generating actions in the world. The design of the control architecture is based on a top-down philosophy. The robot control architecture is broken down into

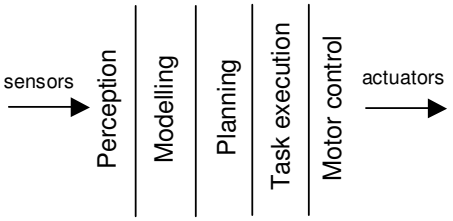

Figure 2.1: Phases of a deliberative control architecture.

an orderly sequence of functional components [Brooks, 1991a] and the user formulates explicit tasks and goals for the system. The sequence of phases usually found in a deliberative control architecture are next described; see also Figure 2.1.

1. **Perception**. In the first component, a sensor interpreter resolves noise and conflicts in the sensory data. Perception algorithms are used to find characteristics and objects within the environment.

2. **Modelling**. With the data obtained from perception, the world modelling component builds a symbolic representation of the world. This representation contains geometric details of all objects in the robot's world with their positions and orientations.

3. **Planning**. The planner then operates on these symbolic descriptions of the world and produces a sequence of actions to achieve the goals given by the users.

4. **Task execution**. This function controls the execution of the planned tasks generating the set-points for each actuator.

5. **Motor control**. This control system is used to control the actuators in accordance with the set-points.

The principal characteristics which all AI approaches have in common are:

- **Hierarchical structure**. The main goals are divided into different tasks, sub-tasks, etc, in a hierarchical manner. Higher levels in the hierarchy provide sub-goals for lower subordinate levels. The tasks are accomplished using a top-down methodology.

- **Sequential processing**. These processes are executed in serial form starting with the sense activities, moving through the modelling and planning, ending with the actuation.

- **Symbolic planner**. The planner reasons, basing itself on a symbolic world model. The world must be generated by linking the physical perceptions to the corresponding symbols.

- **Functionally compartmentalized**. There is a clear subdivision of the different tasks which must be carried out. Each component in the control architecture will be in charge of only one of these functions.

In the 1970's one of the earliest autonomous robots was built using a deliberative control architecture [Fikes and Nilsson, 1971][Nilsson, 1984]. This robot was called Shakey and inhabited a set of specially prepared rooms. It navigated from room to room, trying to satisfy a given goal. While experimenting with this robot, new difficulties were found. The planning algorithms failed with non-trivial solutions, the integration of the world representations was extremely difficult and finally the planning did not work as well as was hoped. The algorithms were improved and better results were obtained. However, the environment was adapted totally to the robot's perceptive requirements. Many other robotic systems have been built with the traditional AI approach [Albus, 1991, Huang, 1996, Lefebvre and Saridis, 1992, Chatila and Laumond, 1985, Laird and Rosenbloom, 1990]. Nevertheless, traditional approaches still have problems when dealing with complex, non-structured and changing environments. Only in structured and highly predictable environments have they proved to be suitable. The principal problems found in Traditional AI can be listed as:

- **Computation**. Traditional AI requires large amounts of data storage and intense computation. For an autonomous mobile robot this can be a serious problem.

- **Real time processing**. The real world has its own dynamics and, for this reason, systems must react fast enough to perform their tasks. Most often, traditional AI is not fast enough because the information is processed centrally. Modelling and planning are long sequential processes, and the longer they take, the more changed the world will be when the robot decides to act. The agent needs simple, multiple and parallel processes instead of only a few long sequential processes.

- **Robustness and Generalization**. Traditional AI usually lacks in generalization capacity. If a novel situation arises, the system breaks down or stops all together. Also, it does not take into consideration problems of noise in the sensory data and actuators when giving its symbolic representation.

- **The accurate world model**. In order to plan correctly, the world model must be very accurate. This requires high-precision sensors and careful calibration, both of which are very difficult and expensive.
- **The Frame problem**. This problem arises when trying to maintain a model of a continuously changing world. If the autonomous robot inhabits a real world, the objects will move and the light will change. In any event, the planner needs a model with which to plan.
- **The Symbol-grounding problem**. The world model uses symbols, such as "door", "corridor", etc, which the planner can use. The Symbol-grounding problem refers to how the symbols are related to real world perceptions. The planner is closed in a symbolic world model while the robot acts in the open real world.

In the middle of the 1980s, due to dissatisfaction with the performance of robots in dealing with the real world, a number of scientists began rethinking the general problem of organizing intelligence. Among the most important opponents to the AI approach were Rodney Brooks [Brooks, 1986], Rosenschein and Kaelbling [Rosenschein and Kaelbling, 1986] and Agre and Chapman [Agre and Chapman, 1987]. They criticized the symbolic world which Traditional AI used and wanted a more reactive approach with a strong relation between the perceived world and the actions. They implemented these ideas using a network of simple computational elements, which connected sensors to actuators in a distributed manner. There were no central models of the world explicitly represented. The model of the world was the real one as perceived by the sensors at each moment. Leading the new paradigm, Brooks proposed the "Subsumption Architecture" which was the first approach to the new field of "Behavior-based Robotics".

Instead of the top-down approach of Traditional AI, Behavior-based systems use a bottom-up philosophy like that in Reactive Robotics. Reactive systems provide rapid real-time responses using a collection of preprogrammed rules. Reactive systems are characterized by a strong response, however, as they do not have any kind of internal states, they are unable to use internal representations to deliberate or learn new behaviors. On the other hand, Behavior-based systems can store states in a distributed representation, allowing a certain degree of high-level deliberation.

The Behavior-based approach uses a set of simple parallel behaviors which react to the perceived environment proposing the response the robot must take in order to accomplish the behavior (see Figure 2.2). There are no problems with world modelling or real-time processing. Nevertheless, another difficulty has to be solved; how to select the proper behaviors for robustness

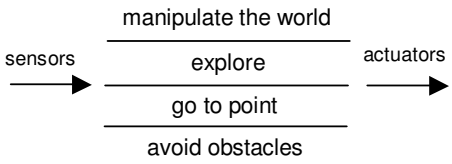

Figure 2.2: Structure of a Behavior-based control architecture.

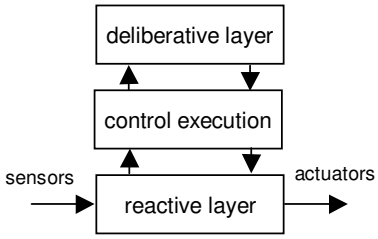

Figure 2.3: The hybrid control architecture structure.

and efficiency in accomplishing goals. New questions also appeared which Traditional AI was not taking into consideration; how to adapt the architecture in order to improve its goal-achievement, and how to adapt it when new situations appear. This powerful methodology demonstrated simplicity, parallelism, perception-action mapping and real implementations.

Behavior-based Robotics has been widely used and investigated since then. This new field has attracted researchers from many and divers disciplines such as biologists, neuroscientists, philosophers, linguists, psychologists and, of course, people working with computer science and artificial intelligence, all of whom have found practical uses for this approach in their various fields of endeavor. The field of Behavior-based Robotics has also been referred to as the "New AI" or, under a more appropriate denomination for the fields mentioned above, "Embodied Cognitive Science".

Finally, some researchers have adopted a hybrid approach between Traditional AI and Behavior-based Robotics. Hybrid systems attempt a compromise between bottom-up and top-down methodologies. Usually the control architecture is structured in three layers: the deliberative layer, the control execution layer and the functional reactive layer (see Figure 2.3). The deliberative layer transforms the mission into a set of tasks which perform a plan. The reactive behavior-based system takes care of the real time issues related to the interactions with the environment. The control execution layer interacts between the upper and lower layers, supervising the accomplishment of the tasks. Hybrid architectures take advantage of the hierarchical

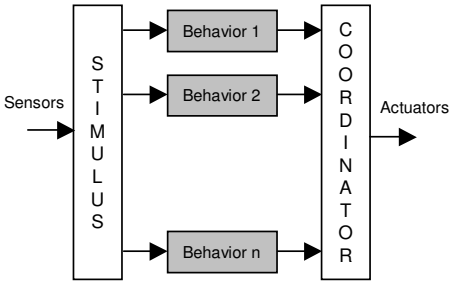

Figure 2.4: Structure of behavior-based CA.

planning aspects of Traditional AI and the reactive and real time aspects of behavioral approaches. Hybrid architectures have been widely used. Some of the best known are AuRA [Arkin, 1986], the Planner-Reactor Architecture [Lyons, 1992] and Atlantis [Gat, 1991] used in the Sojourner Mars explorer.

2.2 Fundamentals of Behavior-based Robotics

Behavior-based Robotics is a methodology for designing autonomous agents and robots. The behavior-based methodology is a bottom-up approach inspired by biology, in which several behaviors act in parallel achieving goals. Behaviors are implemented as a control law mapping inputs to outputs. They can also store states constituting a distributed representation system [Mataric, 1999]. The basic structure consists of all behaviors taking inputs from the robot's sensors and sending outputs to the robot's actuators, see Figure 2.4. A coordinator is needed in order to send only one command at a time to the motors.

The internal structure of a behavior can also be composed of different modules interconnected by sensors, various other modules and finally, the coordinator [Brooks, 1986]. However, behaviors must be completely independent of each other. The global structure is a network of interacting behaviors comprising low-level control and high-level deliberation abilities. The latter is performed by the distributed representation which can contain states and, consequently, change the behavior according to their information.

The parallel structure of simple behaviors allows a real-time response with low computational cost. Autonomous robots using this methodology can be built easily at low cost. Behavior-based robotics has demonstrated its reliable performance in standard robotic activities such as navigation, obstacle avoidance, terrain mapping, object manipulation, cooperation, learn-

ing maps and walking. For a more detailed review refer to [Arkin, 1998, Pfeifer and Scheier, 1999].

2.2.1 Principles

There are a few basic principles which have been used by all researchers in Behavior-based Robotics. These principles provide the keys to success of the methodology.

- **Parallelism**. Behaviors are executed concurrently. Each one can run on its own processor. Parallelism appears at all levels, from behavioral design to software and hardware implementation. This characteristic contributes to the speed of computation and consequently to the dynamics between the robot and the environment.

- **Modularity**. The system is organized into different modules (behaviors). The important fact is that each module must run independently. This important principle contributes to the robustness of the system. If, for example, one behavior fails due to the break down of a sensor, the others will continue running and the robot will always be controlled. Another important consequence of modularity is the possibility of building the system incrementally. In the design phase, the priority behaviors are first implemented and tested. Once they run correctly, more behaviors can be added to the system.

- **Situatedness/Embeddedness**. The concept of "Situatedness" means that a robot is situated in and surrounded by the real world. For this reason it must not operate using an abstract representation of reality, it must use the real perceived world. "Embeddednes" refers to the fact that the robot exists as a physical entity in the real world. This implies that the robot is subjected to physical forces, damages and, in general, to any influence from the environment. This means that the robot should not try to model these influences or plan with them. Instead it should use this system-environment interaction to act and react with the same dynamics as the world.

- **Emergence**. This is the most important principle of Behavior-based Robotics. It is based on the principles explained above and attempts to explain why the set of parallel and independent behaviors can arrive at a composite behavior for the robot to accomplish the expected goals. Emergence is the property which results from the interaction between the robotic behavioral system and the environment. Due to emergence,

the robot performs behaviors that were not pre-programmed. The interaction of behavior with the environment generates new characteristics in the robot's behavior which were not pre-designed. Numerous researchers have talked about emergence. Two examples are "Intelligence emerges from interaction of the components of the system" [Brooks, 1991b] and "Emergence is the appearance of novel properties in whole systems" [Moravec, 1988].

2.2.2 Behavior features

A behavioral response is a functional mapping from the stimulus plane to the motor plane. The motor plane usually has two parameters, the strength (magnitude of the response) and the orientation (direction of the action). According to Arkin [Arkin, 1998], a behavior can be expressed as (S, R, β) where:

- ***S*: Stimulus Domain**. S is the domain of all perceivable stimuli. Each behavior has its own stimulus domain.
- ***R*: Range of Responses**. For autonomous vehicles with six degrees of freedom, the response $r \in R$ of a behavior is a six-dimensional vector: $r = [x, y, z, \phi, \theta, \psi]$ composed of the three translation degrees of freedom and the three rotational degrees of freedom. Each parameter is composed of strength and orientation values. When there are different responses r_i, the final response is $r_i' = g_i \cdot r_i$, where g_i is a gain which specifies the strength of the behaviour relative to the others.
- β**: Behavioral Mapping**. The mapping function "β" relates the stimulus domain with the response range for each individual active behavior: $\beta(s) \rightarrow r$. Behavioral mappings β can be :
  - *Discrete*: numerable sets of responses.
  - *Continuous*: infinite space of potential reactions.

2.2.3 Coordination

When multiple behaviors are combined and coordinated, the emergent behavior appears. This is the product of the complexity between a robotic system and the real world. The two primary coordination mechanisms are:

- **Competitive methods**. The output is the selection of a single behavior, see Figure 2.5a. The coordinator chooses only one behavior to

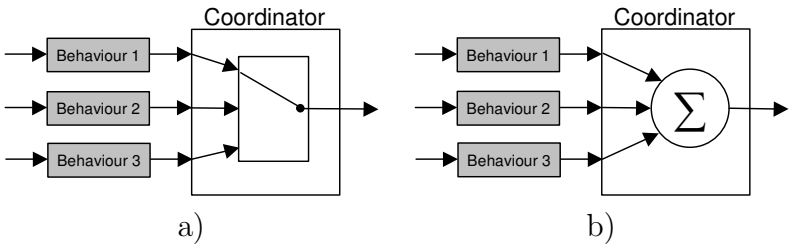

Figure 2.5: Coordination methodologies: a) competition b) cooperation.

control the robot. Depending on different criteria the coordinator determines which behavior is best for the control of the robot. Preferable methods are suppression networks such as Subsumption architecture [Brooks, 1986], action-selection [Maes, 1990] and voting-based coordination [Rosenblatt and Payton, 1989].

- **Cooperative methods**. The output is a combination function off all the active behaviors, see Figure 2.5b. The coordinator applies a method which takes all the behavioral responses and generates an output which will control the robot. Behaviors which generate a stronger output will impose a greater influence on the final behavior of the robot. Principal methods are vector summation such as potential fields [Khatib, 1985], and behavioral blending [Saffiotti et al., 1995].

Basic behavior-based structures use only a coordinator which operates using all the behaviors to generate the robot's response. However, there are more complex systems with different groups of behaviors coordinated by different coordinators. Each group generates an output and with these intermediate outputs, the final robot's response is generated through a final coordinator. These recursive structures are used in high level deliberation. By means of these structures a distributed representation can be made and the robot can behave differently depending on internal states, achieving multi-phase missions.

2.2.4 Adaptation

One of the fields associated with Behavior-based robotics is Adaptation. Intelligence cannot be understood without adaptation. If a robot requires autonomy and robustness it must adapt itself to the environment. The primary reasons for autonomous adaptivity are:

- The robot's programmer does not know all the parameters of the behavior-based system.
- The robot must be able to perform in different environments.
- The robot must be able to perform in changing environments.

And the properties which adaptive systems in robotics must contemplate are [Kaelbling, 1999]:

- Tolerance to sensor noise.
- Adjustability. A robot must learn continuously while performing its task in the environment.
- Suitability. Learning algorithms must be adequate for all kinds of environments.
- Strength. The adaptive system must posses the ability to influence the control of the robot in a desired place in order to obtain the desired data.

However, adaptation is a wide term. According to McFarland there are various levels of adaptation in a behavior-based system [McFarland, 1991]:

- **Sensory Adaptation**. Sensors become more attuned to the environment and changing conditions of light, temperature, etc.
- **Behavioral Adaptation**. Individual behaviors are adjusted relative to the others.
- **Evolutionary Adaptation**. This adaptation is done over a long period of time inducing change in individuals of one species, in this case robots. Descendants change their internal structure based on the success or failure of their ancestors in the environment.
- **Learning as Adaptation**. The robot learns different behaviors or different coordination methods which improve its performance.

Many parameters can be adapted in a behavior-based robotic system. At the moment there are only a few examples of real robotic systems which learn to behave and there is no established methodology to develop adaptive behavior-based systems. The two approaches most commonly used are Reinforcement learning and Evolutionary techniques [Ziemke, 1998, Arkin, 1998,

Dorigo and Colombetti, 1998, Pfeifer and Scheier, 1999]. Both have interesting characteristics but also disadvantages like convergence time or the difficulties in finding a reinforcement or fitness function respectively. In many cases they are implemented over control architectures based on neural networks. Using the adaptive methodologies, the weights of the network are modified until an optimal response is obtained. The two approaches have demonstrated the feasibility of the theories in real robots in all levels of adaptation. The basic ideas of the two methodologies are described next.

- **Reinforcement learning**
  Reinforcement learning (RL) is a class of learning algorithm where a scalar evaluation (reward) of the performance of the algorithm is available from the interaction with the environment. The goal of a RL algorithm is to maximize the expected reward by adjusting some value functions. This adjustment determines the control policy that is being applied. The evaluation is generated by a reinforcement function which is located in the environment. Chapter 4 gives a detailed description of Reinforcement Learning and its application to robotics. Main references about RL are [Kaelbling et al., 1996, Sutton and Barto, 1998].

- **Evolutionary Robotics**
  Evolutionary learning techniques are inspired by the mechanisms of natural selection. The principal method used is Genetic Algorithms (GAs). Evolutionary algorithms typically start from a randomly initialized population of individuals/genotypes encoded as strings of bits or real numbers. Each individual encodes the control system of a robot and is evaluated in the environment. From the evaluation, a score (fitness) is assigned which measures the ability to perform a desired task. Individuals obtaining higher fitness values are allowed to reproduce by generating copies of their genotypes with the addition of changes introduced by various genetic operators (mutation, crossover, duplication, etc.). By repeating this process over several generations, the fitness values of the population increase. Evolutionary robotics has shown good results in real robots. Usually they are used over neural networks modifying the weights of the nodes. Evolutionary algorithms have demonstrated more reliable solutions than reinforcement learning when the reinforcement frequency is low. However, evolutionary approaches also have problems due to the longer time necessary to converge into a reasonable solution. For a complete introduction to Evolutionary Robotics refer to [Nolfi, 1998, Harvey et al., 1997].

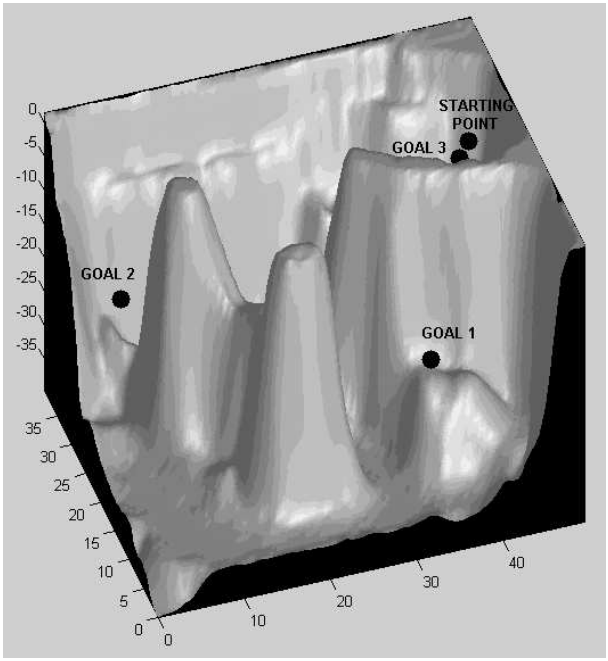

Figure 2.6: Simulated underwater environment in which the AUV must reach three goal points. The dimensions, in meters, of the environment can also be seen.

2.3 Experimental task: control of an AUV

In order to test some behavior-based control architectures, an experimental task was designed. The task consisted of achieving three "goal-points", one after the other, avoiding obstacles and avoiding becoming trapped. The starting point of the task and the three goal points can be seen in Figure 2.6.

A behavior-based control architecture with three behaviors was designed to fulfill the proposed task. All the architectures were implemented with these behaviors changing implementation aspects but maintaining their structure. These behaviors use different inputs but generate an established output. The output was a three-dimensional vector representing the speed the vehicle should follow. The vector $v_i = (v_{i,x}, v_{i,z}, v_{i,yaw})$ is composed of two linear velocities ($v_{i,x}$ with respect to the X axis and $v_{i,z}$ with respect to the Z axis) and an angular velocity ($v_{i,yaw}$ respect Z axis). The three behaviors are next described:

"Go to" behavior. The purpose of this behavior is to drive the vehicle towards the goal point. It proposes a speed vector with a constant

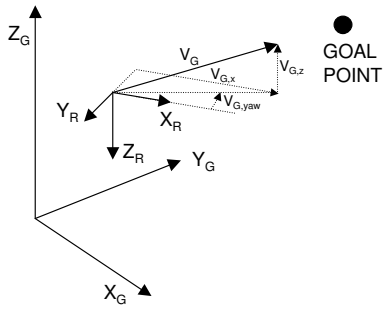

Figure 2.7: Response, V_G, of the "Go To" behavior.

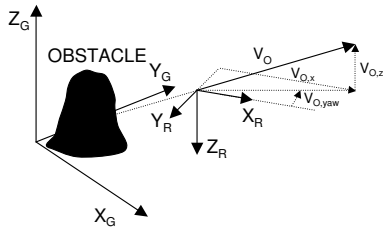

Figure 2.8: Response, V_O, of the "Obstacle Avoidance" behavior.

module. The direction of the vector is the one which joins the position of the vehicle with the goal point, see Figure 2.7. The input of the vector is the position and orientation of the vehicle. The goal points are stored by the behavior and changed when the robot gets close.

Obstacle avoidance behavior. This behavior is used to keep the robot from crashing into obstacles. The inputs of the behavior are various sonar measures. The vehicle is simulated with seven sonar transducers: three at the front, one on each side, one at the back and another on the bottom. Each sonar direction has a maximum range. If one of the distances is less than this range, the behavior will generate an opposite response. The behavior computes all distances and creates a 3D speed vector which indicates the direction the robot must take to avoid obstacles detected by the transducers, see Figure 2.8.

"Avoid trapping" behavior. The avoid trapping [Balch and Arkin, 1993] behavior is used to depart from situations in which the robot becomes trapped. The input of the behavior is the position of the robot which is used to save a *local map* of the recent path of the vehicle. The map is centered on the robot's position and has a finite number of cells.

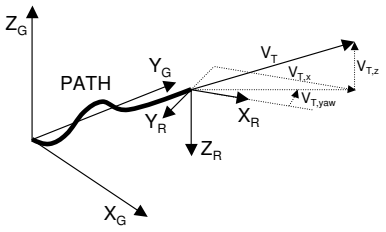

Figure 2.9: Response, V_T, of the "Avoid Trapping" behavior.

Each cell contains the number of times the robot has visited the cell. If the sum of all the values is higher than a threshold, the behavior becomes active and a speed vector is generated. The direction of the vector will be the one opposed to the gravity center of the local map. The module will be proportional to the sum of the cell values. The cells are incremented by a configurable sample time and saturated on a maximum value. However, if cells are not visited, the values are decreased allowing going back to a visited zone.

This behavior becomes active in two specific situations. The first is when the vehicle is trapped in front of obstacles. In this case the behavior will take control of the vehicle and drive it away to another zone. The second situation is when the vehicle is navigating very slowly due to the interaction of the other behaviors. In this case the cells will increase in value rapidly and the behavior will become active driving the vehicle away from the path, see Figure 2.9.

These behaviors were implemented with each tested architecture constituting the control architecture. In addition to the control architecture, other modules were required such as low-level controllers, a accurate dynamics model of the AUV and a graphical interface with an underwater environment, see Figure 2.10. All these components were implemented through a simulated environment. It is specially relevant the identification of the dynamics model of the underwater robot, which was previously performed with real experiments. The simulation of the experimental task was considered to be enough to evaluate the different aspects of the control architectures. It is important to note that typical problems of real robots (position estimation, noise in signals, faults in sonar distances, etc.) were not simulated. The non-consideration of these aspects breaks several principles of Behavior-based Robotics. However, these simulations intend only to test the performance of the control architectures not the principles. We assume that if the architectures were implemented in a real robot, the features of Behavior-based

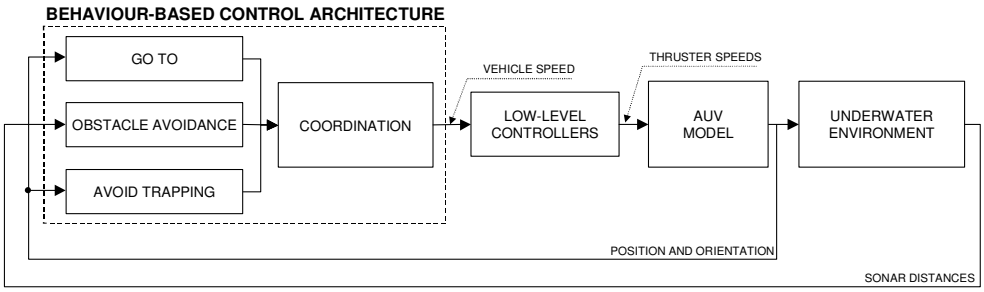

Figure 2.10: Overall control system used in the simulated task.

Control Architecture	Behavioral choice and design	Behavioral encoding	Coordination method	Programming method
Subsumption architecture	Experimentally	Discrete	Competitive, arbitration via inhibition and suppression	AFSM, Behavioral Language or behavioral libraries
Action Selection Dynamics	Experimentally	Discrete	Competitive, arbitration via levels of activation	Mathematical algorithms
Schema-based approach	Ethologically	Continuous, Potential fields	Cooperative via vector summation and normalization	Parameterized behavioral libraries
Process Description Language	Experimentally	Continuous	Cooperative via values integration and normalization	Process Description Language

Table 2.1: Four relevant Behavior-based approaches.

Robotics would assure robustness when faced with these problems.

2.4 Some Behavior-based approaches

Many proposals have appeared since the field of Behavior-based Robotics began in 1986 with Subsumption architecture. From among them, four architectures have been chosen which represent the main principles of Behavior-based Robotics. These architectures were pioneers in the field and their feasibility was demonstrated with real robots by their designers. The architectures and their basic characteristics can be seen in Table 2.1.

In the next subsections, each architecture is first described as it was originally designed and then tested with the experimental task described in the previous section. For each architecture, a discussion of the main advantages and disadvantages is given according to some qualitative properties. Refer to Table 2.2 for the list and description of these properties.

Property	Description
Performance	Quality of the trajectory generated by the vehicle. A control architecture with a good performance will have an optimized and smooth trajectory without big jumps on the vehicle's heading.
Modularity	Property of being able to add new behaviors without having to modify (gains or parameters) the current ones.
Robustness	Property of being able to preserve the integrity of the vehicle and the fulfilment of the task when some small changes in the parameters or in the environment occur.
Development time	Time needed to implement the control architecture, from the design until the programming phase.
Tuning time	Time needed to find the parameters which maximize the performance.
Simplicity	Simplicity of the methodology, including the design and programming phases.

Table 2.2: Qualitative properties used in the evaluation of the control architectures.

2.4.1 Subsumption architecture

Description

The Subsumption architecture was designed by Rodney Brooks in the 1980s at the Massachusetts Institute of Technology. His work opened the field of Behavior-based Robotics. To overcome the problems encountered in Traditional AI when designing real robotic systems, Brooks proposed a completely different methodology. He questioned the centralized symbolic world model and proposed a decentralized set of simple modules which reacted more rapidly to environmental changes. To accomplish this, he presented the Subsumption architecture in 1986 with the paper "A Robust Layered Control System for a Mobile Robot" [Brooks, 1986]. Later on, he modified a few aspects of the architecture as a result of suggestions from J.H. Connell. The final modifications on the Subsumption architecture can be found in [Brooks, 1989, Connell, 1990]. Subsumption Architecture has been widely applied to all kinds of robots since then. Some further modifications have also been proposed. However, in this report, the original Subsumption approach will be described.

Subsumption Architecture is a method of reducing a robot's control architecture into a set of task-achievement behaviors or competences represented as separate layers. Individual layers work on individual goals concurrently and asynchronically. All the layers have direct access to the sensory information. Layers are organized hierarchically allowing higher layers to inhibit or suppress signals from lower layers. Suppression eliminates the control signal from the lower layer and substitutes it with the one proceeding from the

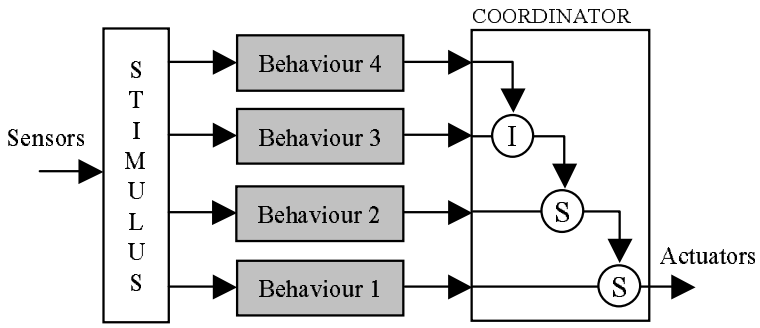

Figure 2.11: Subsumption control architecture. Coordination through suppression and inhibition nodes.

higher layer. When the output of the higher layer is not active, the suppression node does not affect the lower layer signal. On the other hand, only inhibition eliminates the signal from the lower layer without substitution. Through these mechanisms, higher-level layers can subsume lower-levels. The hierarchy of layers with the suppression and inhibition nodes constitute the competitive coordination method, see Figure 2.11.

All layers are constantly observing the sensory information. When the output of a layer becomes active, it suppresses or inhibits the outputs of the layers below, taking control of the vehicle. The layer has internal states and timers which allow it to generate an action which depends on the current state and input. Also, the timers allow the robot to maintain the activity for a period of time after the activation conditions finish.

This architecture can be built incrementally, adding layers at different phases. For example, the basic layers, such as "go to behavior" or "avoid obstacles behavior", can be implemented and tested in the first phase. Once they work properly, new layers can be added without the necessity of redesigning previous ones.

The layers of the subsumption architecture were originally designed as a set of different modules called Augmented Finite States Machines (AFSM). Each AFSM is composed of a Finite State Machine (FSM) connected to a set of registers and a set of timers or alarm clocks, see Figure 2.12. Registers store information from inside FSM as well as from the outside sensors and other AFSM. Timers enable state changes after a certain period of time while finite state machines change their internal state depending on the current state and inputs. An input message or the expiration of a timer can change the state of the machine. Inputs from the AFSM can be suppressed by other machines and outputs can be inhibited. AFSM behave like a single FSM but

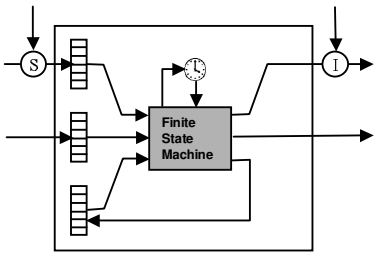

Figure 2.12: Augmented Finite State Machine (AFSM).

with the added characteristics of registers and timers.

Using a set of augmented finite state machines a layer can be implemented to act like a behavior. A layer is constructed as a network of AFSM joined by wires with suppression and inhibition nodes. Figure 2.13 shows the AFSM network designed by Brooks for a mobile robot with the layers "avoid objects", "wander" and "explore", [Brooks, 1986]. As the figure shows, designing the network so as to accomplish a desired behavior is not exactly clear. For this reason, Brooks developed a programming language, the Behavioral Language [Brooks, 1990], which generates the AFSM network using a single rule set for each behavior. The high-level language is compiled to the intermediate AFSM representation and then further compiled to run on a range of processors.

One of the principles of the Subsumption architecture is independence from the layers. The implementation methodology, as stated above, consists of building the layers incrementally once the previous layers have been tested. Nevertheless, in Figure 2.13 there are some wires which go from one layer to another breaking the independence. This fact was shown by Connell [Connell, 1990] who proposed a total independence of the layers until the coordination phase. This assures the possibility of implementing the layers incrementally without redesigning the previous ones. This is also useful in order to map each layer into a different processor in the robot. Connell and other researchers have also proposed other formalism instead of AFSM to implement the layers. Usually, computer programs are used for simplicity in programming rules without the use of FSM. AFSM must be considered as the formalism Brooks chose, for its simplicity and rapid processing, to implement the Subsumption architecture, not as part of it.

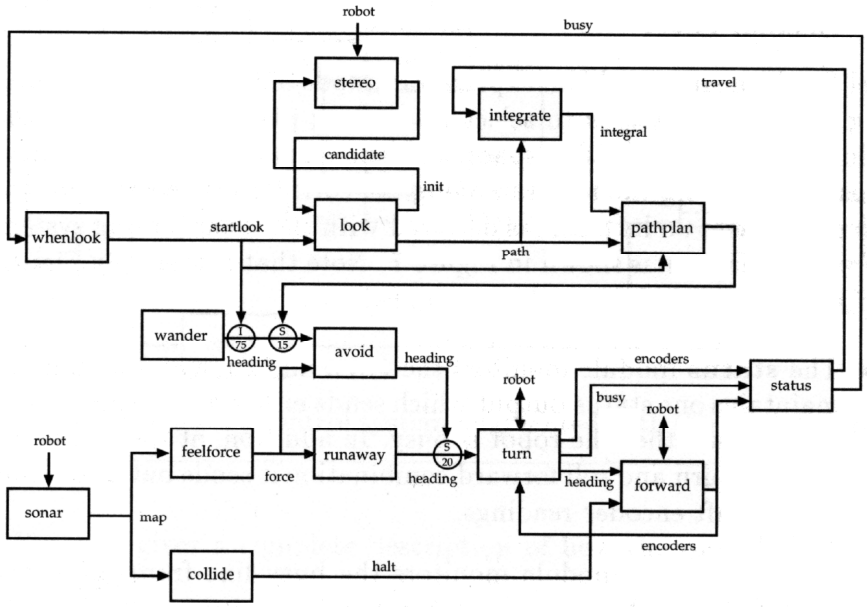

Figure 2.13: AFSM network designed by Brooks for a mobile robot.

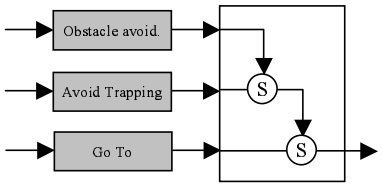

Figure 2.14: Task implementation with Subsumption architecture.

Implementation

Subsumption Architecture was implemented to accomplish the experimental task described in Section 2.3. The three behaviors were implemented in three different functions which use sensory information as inputs and, as outputs, the 3-dimensional velocity the vehicle should follow. Also, two suppression nodes were used to coordinate behaviors. AFSM was not used due to the simplicity of the functions. However, as mentioned before, the implementation method is not the most important fact in the subsumption architecture. The three behaviors were implemented as shown in Figure 2.14.

The hierarchy of behaviors was constructed as follows: at the top, the "Obstacle avoidance" behavior, followed by the "Avoid trapping" and "Go

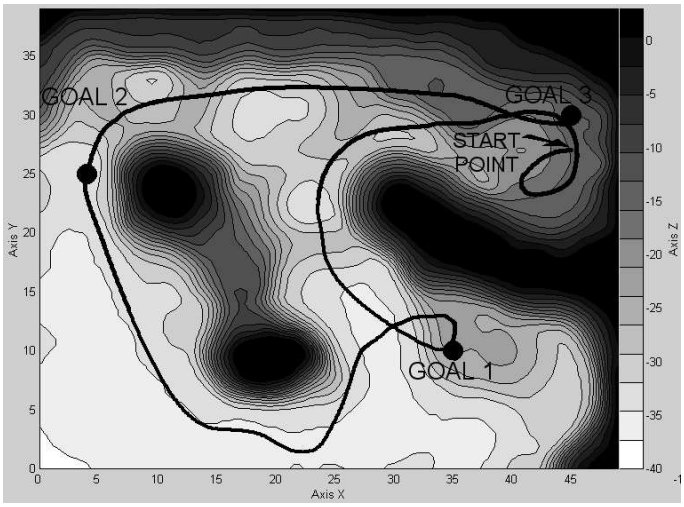

Figure 2.15: Top view of the trajectory carried out by the AUV with Subsumption architecture.

to" behaviors. This hierarchy primarily assures that the vehicle stays away from obstacles. As the task never requires proximity to obstacles, the "Obstacle avoidance" in the top level assures the safety of the vehicle. At the second level, the "Avoid trapping" behavior takes control over the "Go to" behavior if the vehicle becomes trapped. As in AFSM, a timer was used to maintain activity in the behaviors for a short time after the activation conditions were finished.

Some graphical results of the Subsumption approach can be seen in the next three figures. Figure 2.15 shows a top view of the simulation and Figure 2.16 shows a vertical view. Finally Figure 2.17 shows a three-dimensional representation of the simulation.

Given the results, it can be said that the principal advantages of the Subsumption approach are robustness, modularity and easy tuning of the behaviors. Behaviors can be tuned individually and, once they work properly, can be mixed. The design of the hierarchy is very easy once the priorities of the behaviors are known (a difficult task when working with a large architecture). This architecture is very modular, every behavior can be implemented with a different processor with the responses coordinated as a final step. A sequential algorithm is not necessary as all behaviors are completely independent. The principal disadvantage is the non-optimal trajectories, due to the competitive coordination method, with a lot of jumps in the vehicle's heading when the active behavior changes. Table 2.3 summarizes the prin-

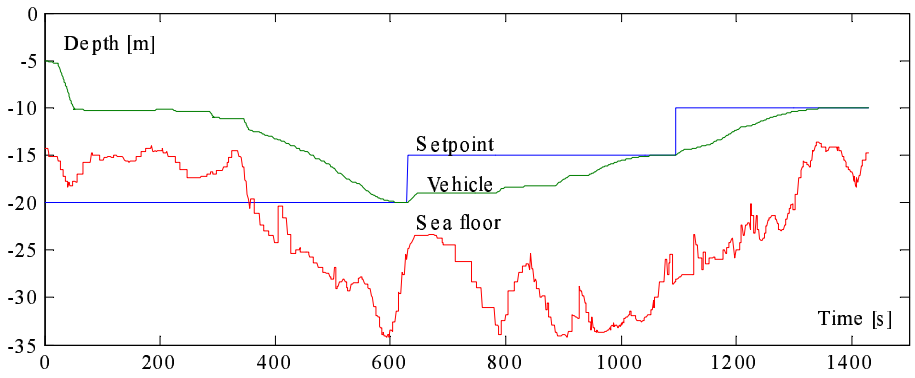

Figure 2.16: Vertical view of the trajectory carried out by the AUV with Subsumption architecture. The set-point graph is the depth of the goal to be achieved. The vehicle graph is the depth of the vehicle during the whole simulation. And the sea floor graph is the depth of the sea floor directly under the vehicle.

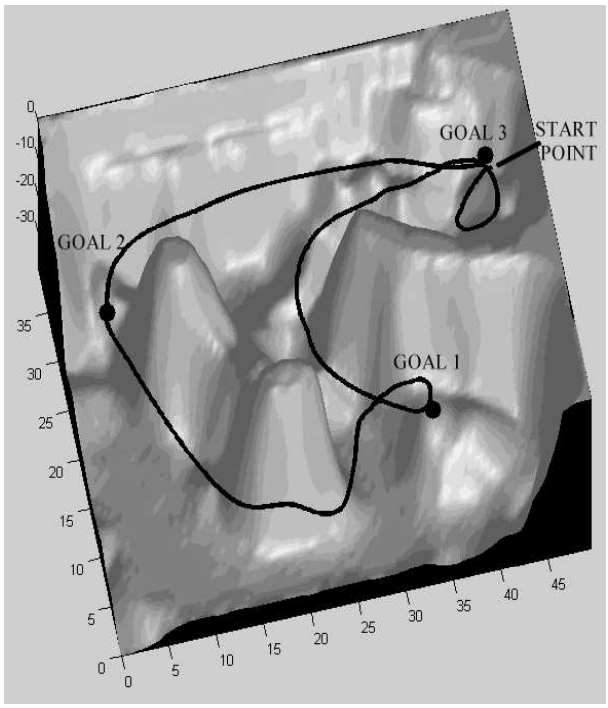

Figure 2.17: Three-dimensional view of the trajectory carried out by the AUV with Subsumption architecture.

SUBSUMPTION ARCHITECTURE	
Developer	Rodney Brooks, Massachusetts Institute of Technology
References	[Brooks, 1986, Brooks, 1989, Connell, 1990]
Behavioral encoding	Discrete
Coordination method	Competitive, arbitration via inhibition and suppression
Programming method	AFSM, Behavioral Language or behavioral libraries
Advantages	Modularity, Robustness and Tuning time
Disadvantages	Development time and performance

Table 2.3: Subsumption architecture features.

cipal characteristics of Subsumption Architecture. For a description of the advantage/disadvantage terms refer to Table 2.2.

2.4.2 Action Selection Dynamics

Description

Action Selection Dynamics (ASD) is an architectural approach which uses a dynamic mechanism for behavior (or action) selection. Pattie Maes from the AI-Lab at MIT developed it toward the end of the 1980s. Principal references are [Maes, 1989, Maes, 1990, Maes, 1991]. Behaviors have associated activation levels which are used to arbitrate competitively the activity which will take control of the robot. Other approaches for action selection have been proposed [Tyrell, 1993], however, ASD is the most well known and most commonly specified.

Action Selection Dynamics uses a network of nodes to implement the control architecture. Each node represents a behavior. The nodes are called *competence modules*. A network of modules is used to determine which competence module will be active and, therefore, control the robot. The coordination method is competitive, only one module can be active at any moment. To activate the competence modules some binary states are used. Each competence module has three lists of states which define its interaction within the network. The first list is the *precondition list* and contains all the states which should be true so that the module becomes executable. The second list is the *add list* and contains all the states which are expected to be true after the activation of the module. Finally, the third list is the *delete list* and contains the states which are expected to become false after the execution of the module.

The states are external perceptions of the environment gathered by the sensors. Usually some kind of processing will be necessary to transform the analogue outputs of the sensors to a binary state. For example, for the state "No_Obstacle", all the values provided by the sonar have to be processed

to determine if there are nearby obstacles. The states can also be internal assumptions or motivations of the robot. The state "Way-point-Reached" could be one example. The states are also used to determine the *goals* and *protected goals* of the robot. The goals would be the states which are desired to be true. The protected goals are the goals already achieved and therefore retained. The task of the robot is defined by the assignment of the goals to some states.

Once all the states and competence modules are defined, the decision network can be built. Different links appear between the nodes based on the precondition, add and delete lists: ·

- **Successor link**: For each state which appears in the add list of module A and in the precondition list of module B, a successor link joins A with B.

- **Predecessor link**: A predecessor link joints B with A if there is a successor link between A and B.

- **Conflicter link**: For each state which appears in the delete list of module B and in the precondition list of module A, a conflicter link joins A with B.

In Figure 2.18 successor links can be seen as solid line arrows and conflicter links as solid lines with a big dot instead of an arrow. Note that predecessor links are inverted successor links.

Activation of competence modules occurs depending on the quantity of energy they are given. The energy is spread in two phases. In the first phase three different mechanisms are used:

1. *Activation by the states*: if at least one state in the precondition list is true, activation energy is transferred to the competence module.

2. *Activation by the goals*: if at least one state in the add list belongs to a goal state, activation energy is transferred to the competence module.

3. *Activation by the protected goals*: if at least one state in the delete list belongs to a protected goal state, activation energy is removed from the competence module.

The spread of energy in phase one is shown in Figure 2.18 with dotted lines. On the other hand, phase two spreads energy from competence modules. Three mechanisms are also used:

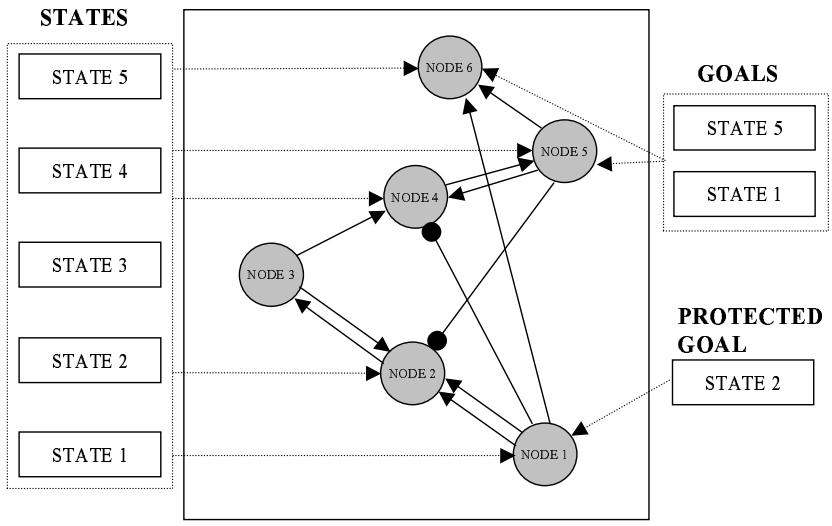

Figure 2.18: Example of an Action Selection Dynamics network.

1. *Activation of Successors*: Executable modules spread a fraction of their own energy to successors which aren't executable if the state of the link is false. The goal is to increase activation of modules which become executable after the execution of the predecessor module.

2. *Activation of Predecessor*: Non-executable modules spread a fraction of their own energy to the predecessor if the state of the link is false. The goal is to spread energy to the modules so that, through their execution, the successor module becomes executable.

3. *Inhibition of Conflicters*: Competence modules decrease the energy of conflicter modules if the state of the link is true. The goal is to decrease the energy of the conflicters which, by becoming active, make the preconditions of the module false.

In each cycle the competence modules increase or decrease their energy until a global maximum and minimum level are reached. The activated module has to fulfil three conditions:

1. It has to be executable (all preconditions have to be true).

2. Its level of energy has to surpass a threshold.

3. Its level of energy has to be higher than that of the modules accomplishing conditions 1 and 2.

State	Description
WAY-POINT	There is a way-point to go
NO_WAY-POINT	There is not any way-point
OBSTACLE	There is a nearby obstacle
NO_OBSTACLE	There is not any nearby obstacle
TRAPPED	The vehicle cannot depart from the same zone
NO_TRAPPED	The vehicle is moving through different zones

Table 2.4: ASD States to fulfill the task.

When a module becomes active, its level of energy is re-initialized to 0. If none of the modules fulfil condition 2, the threshold is decreased. Several parameters are used for the thresholds and the amount of energy to be spread. Also, normalization rules assure that all modules have the same opportunities to become active. Note that the energy of the modules is accumulated incrementally, then the sample time becomes very important because it determines the velocity of the accumulation. For a mathematical notation of this algorithm refer to [Maes, 1989].

The intuitive idea of Action Selection Dynamics is that by using the network and the spreading of energy, after some time, the active module is the best action to take for the current situation and current goals. Although Action Selection Dynamics is complex and difficult to design, it has been successfully tested in real robotic systems.

Implementation

Action Selection Dynamics Architecture was used to accomplish the experimental task described in Section 2.3. In the implementation of the ASD network, the three behaviors represent three competence modules. Each behavior was implemented in a function. The coordination module which contains the ASD network was also implemented in another function following the mathematical notations found in [Maes, 1989]. Each module used different binary states in its lists. In Table 2.4 all the states are shown. Note that from the six states, three are the negation of the other three. This is to simplify the ASD algorithm. The goal of the robot is the state "No_Way_Point". When this state is true the robot has passed through all the way-points and therefore the task is complete. After the state description, the preconditions list, add list and delete list were defined, see Table 2.5.

Following the lists of each competence module the network can be implemented, see Figure 2.19. Note that the design phase for the ASD architecture consists of specifying the states and the lists. After this, the entire network

Go To	**STATES:**		
PRECONDITION LIST	WAY-POINT	NO_OBSTACLE	NO_TRAPPED
ADD LIST	NO_WAY_POINT	OBSTACLE	TRAPPED
DELETE LIST	WAY-POINT		
Obstacle Avoidance	**STATES:**		
PRECONDITION LIST	OBSTACLE		
ADD LIST	NO_OBSTACLE		
DELETE LIST	OBSTACLE		
Avoid Trapping	**STATES:**		
PRECONDITION LIST	WAY-POINT	NO_OBSTACLE	TRAPPED
ADD LIST	NO_TRAPPED	OBSTACLE	
DELETE LIST	TRAPPED		

Table 2.5: Precondition, add and delete lists for the 3 competences.

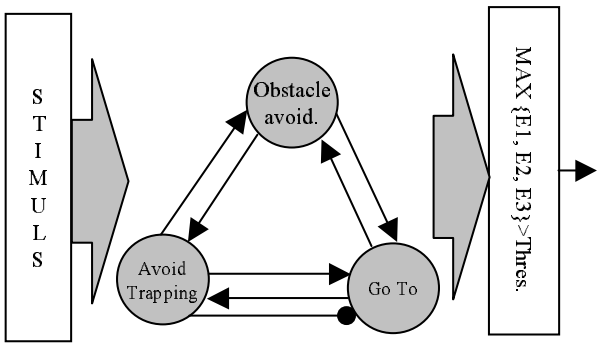

Figure 2.19: Task implementation with Action Selection Dynamics.

can be generated automatically and only some parameters must be tuned.

The ASD network spreads energy from the states, the goal and between the competence modules. The competence module, which is executable and has more energy than the others and the threshold, will activate and control the robot until the next iteration. The parameters of the decision network that have been used can be seen in Table 2.6. The spreading of energy between the modules is determined by the relationship between the different parameters.

Some graphical results of the ASD approach can be seen in the next three figures. Figure 2.20 shows a top view of the simulation and Figure 2.21 shows a vertical view. Finally, Figure 2.22 shows a three-dimensional representation of the simulation. As can be seen, the trajectory obtained with Action Selection Dynamics is quite optimal. The principal advantages of this method are robustness of the architecture and automatic coordination, once the network

Parameter	Description	Value
π	Maximum level of energy per module	40 units
ϕ	Amount of energy spread by a state	10 units
γ	Amount of energy spread by a goal	20 units
δ	Amount of energy spread by a protected goal	0 units
θ	Threshold to becoming active	15 units
T_s	Sample time	1 second

Table 2.6: Parameters of the ASD implementation found experimentally.

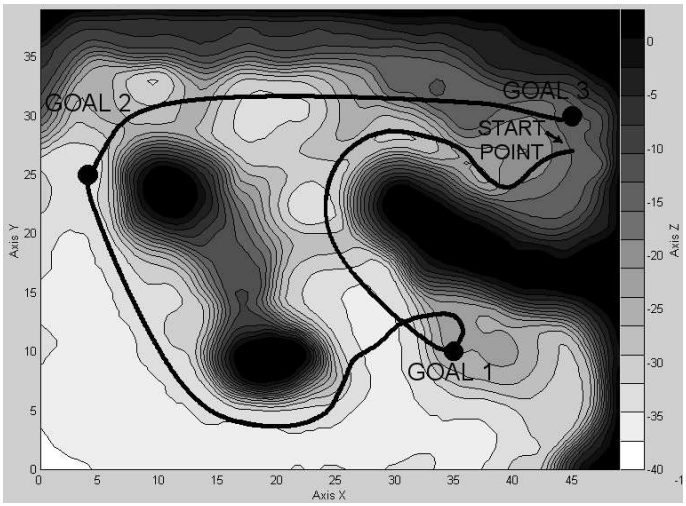

Figure 2.20: Top view of the trajectory carried out by the AUV with Action Selection Dynamics.

has been generated. However, the design and implementation phases are very complex and difficult. Table 2.7 summarizes the principal characteristics of Action Selection Dynamics. For a description of the advantage/disadvantage terms refer to Table 2.2.

2.4.3 Motor Schemas approach

Description

Schema-based theories appeared in the eighteenth century as a philosophical model for the explanation of behavior. Schemas were defined as the mechanism of understanding sensory perception in the process of storing knowledge. Later on, at the beginning of the twentieth century, the schema

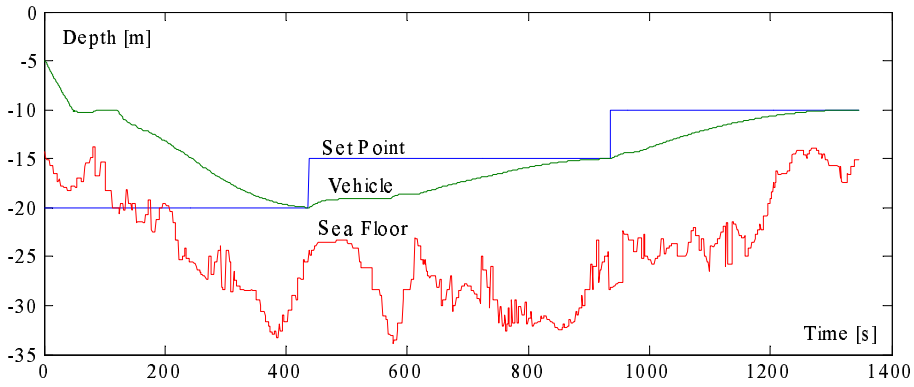

Figure 2.21: Vertical view of the trajectory carried out by the AUV with Action Selection Dynamics. The depths of the goal-point, the vehicle and the sea floor are shown.

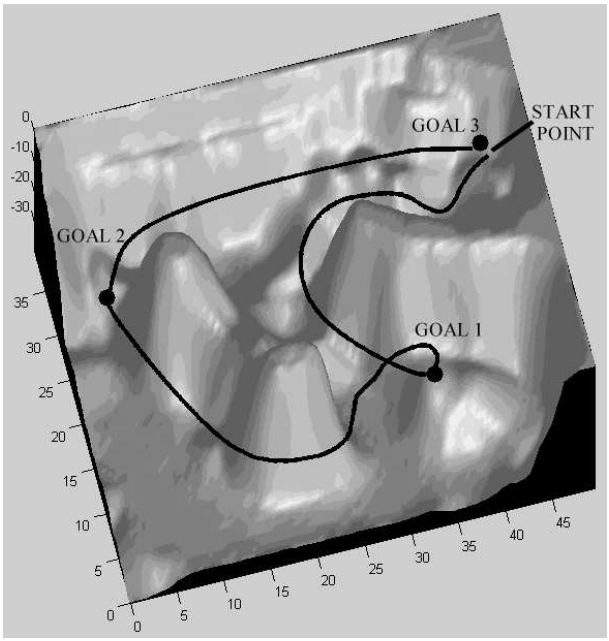

Figure 2.22: Three-dimensional view of the trajectory carried out by the AUV with Action Selection Dynamics.

ACTION SELECTION DYNAMICS	
Developer	Pattie Maes, Massachusetts Institute of Technology
References	[Maes, 1989, Maes, 1990, Maes, 1991]
Behavioral choice and design	Experimentally
Coordination method	Competitive, arbitration via levels of activation
Programming method	Mathematical algorithms
Advantages	Modularity and Robustness
Disadvantages	Development time and No Simplicity

Table 2.7: Action Selection Dynamics features.

theory was adapted in psychology and neuroscience as a mechanism for expressing models of memory and learning. Finally in 1981, Michael Arbib adapted the schema theory for a robotic system [Arbib, 1981]. He built a simple schema-based model inspired by the behavior of the frog to control robots. Since then, schema-based methodologies have been widely used in robotics. The principal proposal is Motor Schemas developed by Ronald Arkin at Georgia Institute of Technology, Atlanta. Arkin proposed Motor Schemas [Arkin, 1987] as a new methodology of Behavior-based Robotics.

From a robotic point of view "a motor schema is the basic unit of behavior from which complex actions can be constructed; it consists of the knowledge of how to act or perceive as well as the computational process by which it is enacted" [Arkin, 1993]. Each schema operates as a concurrent, asynchronous process initiating a behavioral intention. Motor schemas react proportionally to sensory information perceived from the environment. All schemas are always active producing outputs to accomplish their behavior. The output of a motor schema is an action vector which defines the way the robot should move. The vector is produced using the potential fields method [Khatib, 1985]. However, instead of producing an entire field, only the robot's instantaneous reaction to the environment is produced, allowing a simple and rapid computation.

The coordination method is cooperative and consists of vector summation of all motor schema output vectors and normalization. A single vector is obtained determining the instantaneous desired velocity for the robot. Each behavior contributes to the emergent global behavior of the system. The relative contribution of each schema is determined by a gain factor. Safety or dominant behaviors must have higher gain values. Normalization assures that the final vector is within the limits of the particular robot's velocities. Figure 2.23 shows the structure of motor schema architecture.

Implementation of each behavior can be done with parameterized behavioral libraries in which behaviors like "move ahead", "move-to-goal", "avoid-static-obstacle", "escape" or "avoid-past" can be found. Schemas have inter-

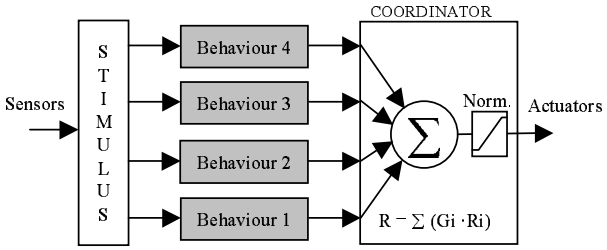

Figure 2.23: Example of a control architecture with Motor Schema approach.

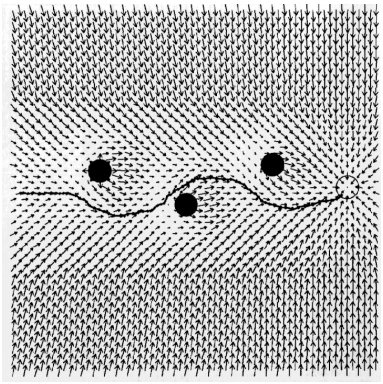

Figure 2.24: Potential field generated with Motor Schema approach.

nal parameters depending on the behavior and an external parameter, the gain value. Each schema can be executed into a different processor. Nevertheless, the outputs must have the same format in order to be summed by the coordinator. For two-dimensional vehicle control refer to [Arkin, 1989] and for three-dimensional control to [Arkin, 1992]. For a set of positions, each behavior generates a potential field which indicates the directions to be followed by the robot in order to accomplish the behavior. The merging of the behaviors in different robot positions, provides a global potential field, see Figure 2.24, which gives an intuitive view of the motor schema architecture performance.

Implementation

Motor Schema Architecture was applied to accomplish the experimental task described in Section 2.3. Each behavior was implemented in a different function. A simple coordination module was used to sum the signals and normalize the output. The structure of the control architecture can be seen in

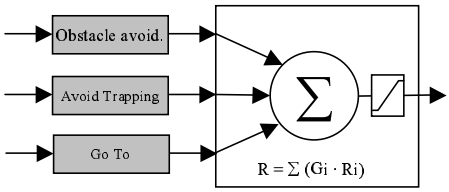

Figure 2.25: Task implementation with Motor Schema approach.

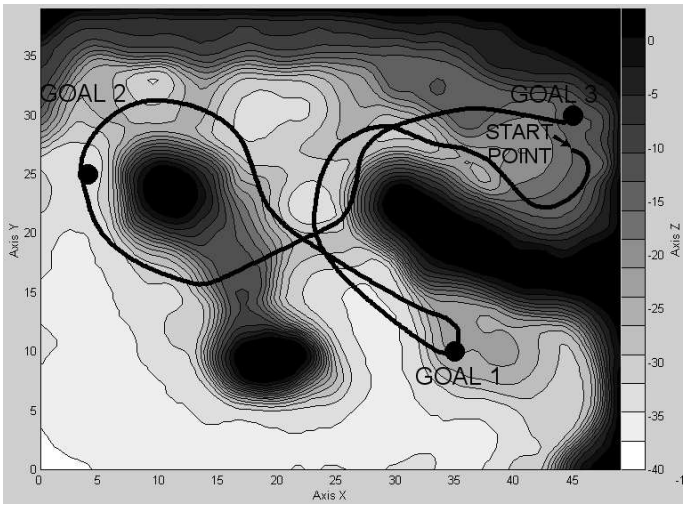

Figure 2.26: Top view of the trajectory carried out by the AUV with Motor Schema approach.

Figure 2.25. After tuning the system, "Obstacle avoidance" behavior had the highest gain value, followed by "Avoid trapping" and "Go to" behaviors. As in Subsumption architecture, higher priority was given to the safety behavior "Obstacle avoidance", followed by "Avoid trapping" to take control over "Go to" when necessary.

Some graphical results of the Motor Schema approach can be seen in the next three figures. Figure 2.26 shows a top view of the simulation and Figure 2.27 shows a vertical view. Finally, Figure 2.28 shows a three-dimensional representation of the simulation.

After reviewing the results, it can be said that the principal advantages are simplicity and easy implementation, as well as optimized trajectories. The architecture can be implemented in different processors because the algorithm is fully parallel. However, difficulties appeared in tuning the gain values. The values are very sensitive and have to be tuned together. When

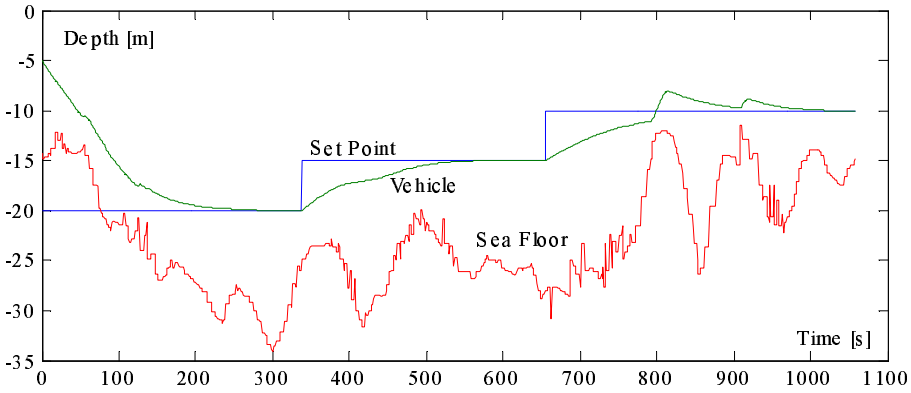

Figure 2.27: Vertical view of the trajectory carried out by the AUV with Motor Schema approach. The depths of the goal-point, the vehicle and the sea floor are shown.

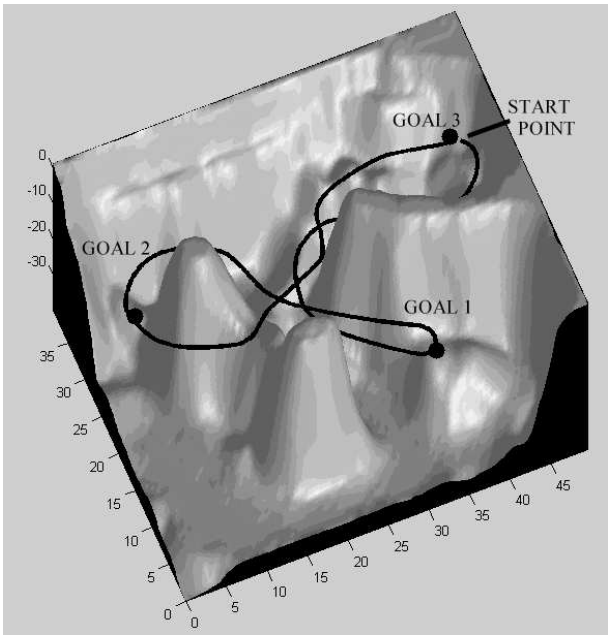

Figure 2.28: Three-dimensional view of the trajectory carried out by the AUV with Motor Schema approach.

MOTOR SCHEMA APPROACH	
Developer	Ronald Arkin, Georgia Institute of Technology
References	[Arkin, 1987, Arkin, 1989, Arkin, 1992]
Behavioral encoding	Continuous using potential fields
Coordination method	Cooperative via vector summation and normalization
Programming method	Parameterized behavioral libraries
Advantages	Development time and simplicity
Disadvantages	Tuning time, robustness and modularity

Table 2.8: Motor Schema approach features.

new behaviors are added, re-tuning is necessary because the sum of the responses of some behaviors can cancel the effect of others, such as "Obstacle avoidance". For this reason, robustness and modularity are very low. Table 2.8 summarizes the principal characteristics of Motor Schema approach. For a description of the advantage/disadvantage terms refer to Table 2.2.

2.4.4 Process Description Language

Description

Process Description Language (PDL) was introduced in 1992 by Luc Steels from the VUB Artificial Intelligence Laboratory, Belgium. PDL [Steels, 1992, Steels, 1993] is intended as a tool to implement process networks in real robots. PDL is a language which allows the description and interaction of different process constituting a cooperative dynamic architecture.

PDL architecture is organized with different active behavior systems, see Figure 2.29. Each one is intended as an external behavior of the robot like "explore", "go towards target" or "obstacle avoidance". Each behavior also contains many active processes operating in parallel. Processes represent simple movements which the behavior will use to reach its goal. Processes take information from sensors and generate a control action if needed. The control action is related to the set-points which must reach several actuators of the robot. A process output is an increment value which will be added to or subtracted from some set-points of the actuators. This means, for example, that the process "turn right if the left bumpers are touched" will add a value to the left motor set-point speed and subtract it from the right, if the necessary conditions are true (in this case, touching the left bumpers). The contribution of all the processes will be added to the current set-points and then a normalization will assure a bounded output. This simple methodology constitutes the cooperative coordination method.

Process description language proposes the language used to implement such processes. The functions are very simple allowing high speed processing.

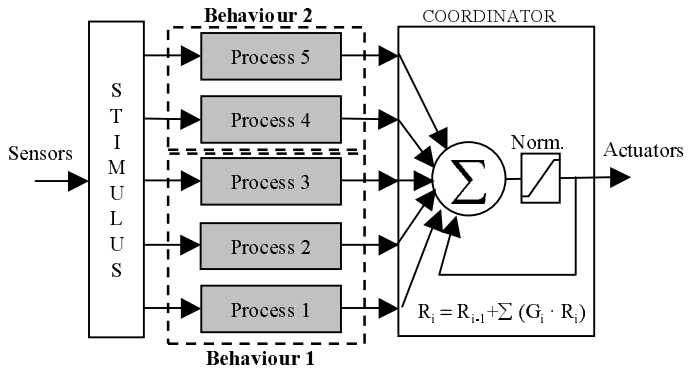

Figure 2.29: Example of a control architecture with Process Description Language.

For example, the process "turn right if the left bumpers are touched" would be implemented as:

```
void turn_right(void)
{
      if(bumper_mapping[3]>0{
            add_value(left_speed, 1);
            add_value(right_speed, -1);}
}
```

The relative contribution of each process is determined by the value added to or subtracted from the set-points. Processes with large values will exert a greater influence on the robot. The ultimate direction taken by the robot will depend on which process influences the overall behavior in the strongest way. It must be noted that these values are added each time step. For this reason it is very important that the ranges of these values be related to the sample time in which the architecture is working. It is possible that with small values and a big sample time, the architecture might not be able to control the robot. The dynamics of the architecture must be faster than the dynamics of the robot. This is due to the fact that PDL works by manipulating derivatives of the set-points implying a fast control loop to assure the system's stability. It's important to remember that although PDL is structured in simple and fast processes, the dynamics will always have to be faster than that of the robot or vehicle to be controlled.

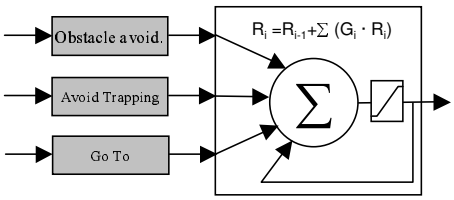

Figure 2.30: Task implementation with Process Description Language.

The overall execution algorithm is defined by the following recursive procedure:

1. All quantities are frozen (sensory information and set-points).
2. All processes are executed and their relative contribution are added or subtracted.
3. The set-points are changed based on the overall contribution of the processes.
4. The set-points are sent to the actuator controllers.
5. The latest sensory quantities are read in.

Implementation

Process Description Language Architecture was used to accomplish the experimental task described in Section 2.3. Each behavior was implemented in a different function. The low level processes of each behavior were assembled and the behavior only generated a response. The response is a vector which changed the current velocity of the vehicle in the direction desired by the behavior. The coordinator is a simple module which sums the current velocity with those generated by the behaviors. The final vector was normalized. Figure 2.30 shows the structure of this architecture.

The module of the vectors is used to give priority to some behaviors over others. In this case, the maximum speed of the robot and the modules of the behaviors can be seen in Table 2.9. If the response of the coordinator exceeded the maximum speed value, the response was saturated. This means that when the "Obstacle avoidance" behavior became active, it affected the overall behavior in the strongest way. And the "Avoid trapping" only dominated the "Go to" behavior to depart from possible entrapment situations. All these values are closely related to the sample time of the control architecture. PDL is a methodology which works with derivatives of the speed, in

Vector	Maximum Magnitude [m/s]
Robot speed	0.5
"Obstacle avoidance"	0.15
"Avoid trapping"	0.04
"Go to"	0.03

Table 2.9: Maximum values for the robot speed and behaviors. Found experimentally.

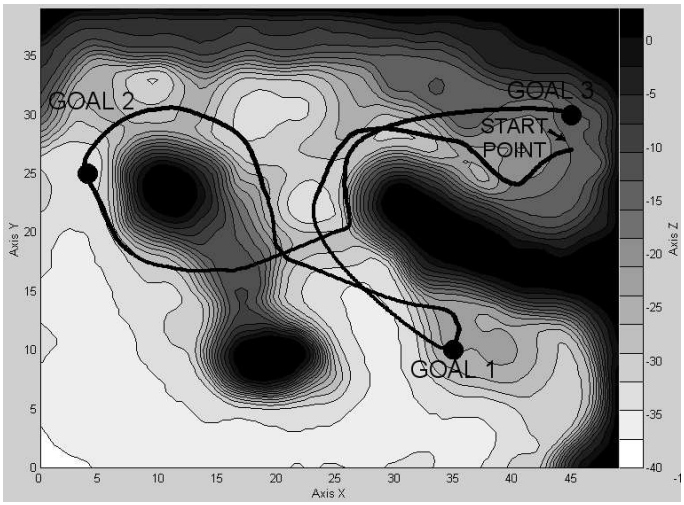

Figure 2.31: Top view of the trajectory carried out by the AUV with Process Description Language.

this case. This means that the dynamics of PDL should be faster than the dynamics of the robot. Experiments with PDL were initially carried out with the same sample time as with the other evaluated architectures. However, the control architecture was not fast enough and the final sample time had to be considerably reduced.

Some graphical results of Process Description Language approach can be seen in the next three figures. Figure 2.31 shows a top view of the simulation and Figure 2.32 shows a vertical view. Finally, Figure 2.33 shows a three-dimensional representation of the simulation.

After reviewing the simulated results it can be said that PDL provides an easy tool to implement a control architecture. Advantages are simplicity and optimized trajectories when the architecture is tuned. However, as

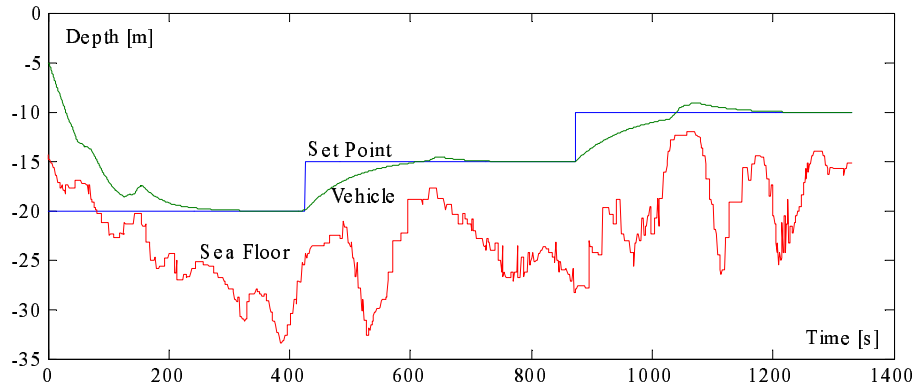

Figure 2.32: Vertical view of the trajectory carried out by the AUV with Process Description Language. The depths of the goal-point, the vehicle and the sea floor are shown.

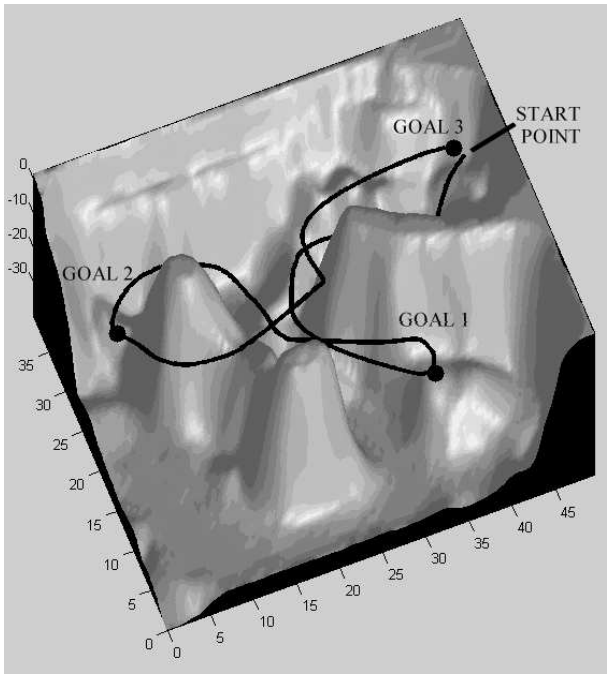

Figure 2.33: Three-dimensional view of the trajectory carried out by the AUV with Process Description Language.

PROCESS DESCRIPTION LANGUAGE	
Developer	Luc Steels, VUB Artificial Intelligence Laboratory
References	[Steels, 1992, Steels, 1993]
Behavioral encoding	Continuous
Coordination method	Cooperative via values integration and normalization
Programming method	Process Description Language
Advantages	Development time and simplicity
Disadvantages	Small sample time, Tuning time, Robustness and Modularity

Table 2.10: Process Description Language features.

Property\Architecture:	$\mathbf{1}^{st}$	$\mathbf{2}^{nd}$	$\mathbf{3}^{rd}$	$\mathbf{4}^{th}$
Performance	SCHE.	PDL	ASD	SUBS.
Modularity	SUBS.	ASD	PDL	SCHE.
Robustness	SUBS.	ASD	PDL	SCHE.
Development time	SCHE.	PDL	SUBS.	ASD
Tuning time	SUBS.	ASD	PDL	SCHE.
Simplicity	SCHE.	PDL	SUBS.	ASD.

Table 2.11: Behavior-based architectures rankings.

the coordinator method is cooperative, the tuning was very difficult. When new behaviors are added, re-tuning is necessary. Also there is the problem of the sample time, which must be faster than with the other approaches. Moreover, since the final velocity vector is obtained incrementally, the architecture acts sequentially and has a low modularity. Table 2.10 summarizes the principal characteristics of the Process Description Language approach. For a description of the advantage/disadvantage terms refer to Table 2.2.

2.5 Coordination Methodology Comparison

Once the four behavior-based architectures had been implemented and tested some conclusions were drawn. It should be noted that the architectures were implemented for a simple but representative task. However, this task was felt to be sufficient to discover the attractiveness and deficiencies of each architecture. As commented above, each architecture exhibited its own advantages and disadvantages and for this reason each would be well suited for a particular application. For each evaluated property, a ranking of architectures has been established, see Table 2.11. In this table, the architecture listed first is the one which best fits the corresponding property.

Looking at the rankings table, it can be seen that properties can be grouped in accordance with the coordination method, as summarizes Ta-

COMPETITIVE METHODS Subsumption and ASD		**COOPERATIVE METHODS** Motor Schema and PDL	
Advantages	*Disadvantages*	*Advantages*	*Disadvantages*
Modularity	Performance	Performance	Modularity
Robustness	Development Time	Development Time	Robustness
Tuning time	No Simplicity	Simplicity	Tuning Time

Table 2.12: Properties according to the coordination methodology.

ble 2.12. *Competitive* methods (Subsumption and ASD) have robustness, modularity and easy tuning. This is due to the fact that they only have one active behavior at any given moment. Therefore, robustness is preferable because in dangerous situations only a safety behavior will act and the danger is avoided. Modularity should also be considered an important property of competitive methodologies because more behaviors can be added without influencing the old ones. Only the coordinator will have to be adapted to the new input. For this reason the tuning time is very short. Behaviors are tuned independently and once they work properly they never have to be re-tuned.

However, competitive methods have disadvantages as well, mainly in the coordinator design. In order to choose only one active behavior, a complex method must be used (ASD) or a clear understanding of the hierarchy of behaviors is necessary (Subsumption). Once this is done, the final tune-up is very easy. For this reason the development time is usually long and the coordinator can become very complex. Another negative property is slow performance due to the non-instant merging of behaviors. Big changes in the vehicle's heading occur when more than one behavior is acting consecutively.

As competitive approaches, the two methodologies studied, Subsumption and Action Selection Dynamics, possess all these properties. However, they have quite different philosophies. Subsumption is more a low-level approach. The hierarchy of behaviors has to be known and then the network has to be designed. Subsumption offers a series of tools, the suppression and inhibition nodes, to build the network. For this reason, the implementation will be simple, but perhaps the design will be quite difficult. On the other hand, Action Selection Dynamics is a high level approach to building an architecture. All the competence modules are completely described and the design consists of filling in all the module lists. Once all the behaviors are perfectly described, the network will automatically choose the best behavior. In ASD the design will be easier but the implementation is more difficult.

In contrast to competitive approaches, methods with a *cooperative* coordinator have other properties like simplicity, performance and development

time, see Table 2.12. Due to the fact that all behaviors are active, the response will always be a merging of these behaviors. This means that the trajectory described by the robot will be smoother than that described by a competitive method. For this reason performance will be a common property.

Another property is simplicity. The coordinator will be very simple, because the final output is usually a kind of sum of all the behaviors multiplied by the gain factors. In relation to simplicity, the development time will be small. However, this simplicity causes great difficulties in tuning the priority gains as the values are very sensitive and critical. In extreme situations, non-safe behaviors can cancel the safe ones. Unfortunately, the modularity will be very bad as a result, because each new behavior will cause the re-tuning of all the priority gains.

The differences between the two cooperative approaches studied, Motor Schema and Process Description Language, are in the level of abstraction and in the coordinator. In Motor Schema, behaviors are implemented individually with the behavioral library. It's more a high-level design. However, in PDL behaviors are implemented as low-level processes which change the set-points of the actuators a little. Implementation will be simple but the design will be more complex. Nevertheless, the principal difference between the two approaches is found in the coordinator. In Motor schema, the output is obtained every time step from the outputs of the behaviors. On the other hand, in PDL the output is an integration of all the outputs generated before. This implies that a small sample time is needed to assure the stability of the system, which can be problematic if there is a lot of computation to do.

Concluding this comparison, it can be said that, depending on the exigencies of the robot to be controlled, one method can prove to be more appropriate than another. Once the architecture has been implemented, a cooperative method can be more suitable if better performance is necessary. Or a competitive method if robustness is the basic premise. The control architecture could also depend on hardware availability, sensorial system and compatibility with adaptive algorithms.