The pursuit ability of a robot ‘pet’

Julien Roux

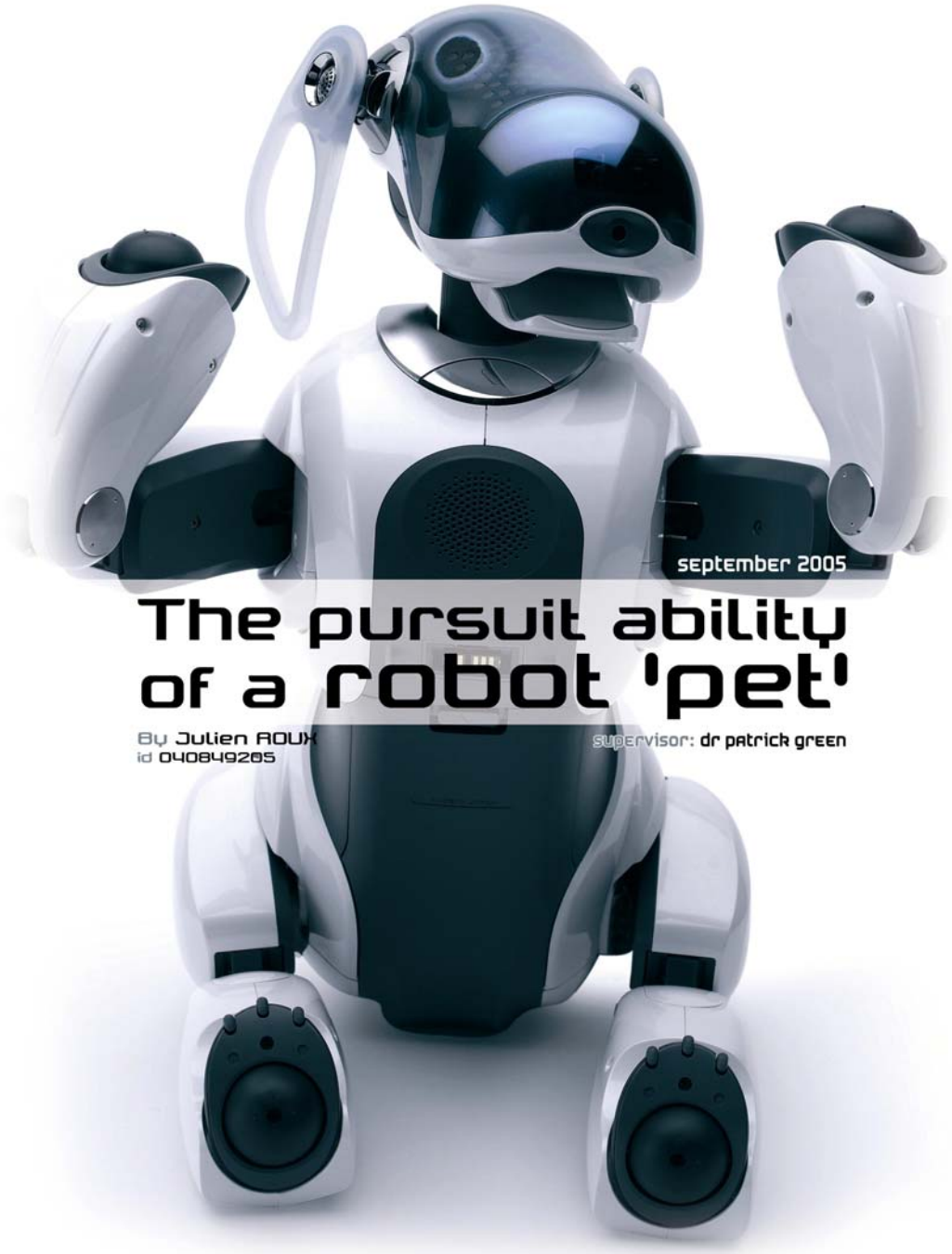
MSc Dissertation
Distributed and Multimedia Information Systems
Academic Year 2004/2005
Supervisor: Patrick Green

Declaration

I, Julien Roux, confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the words of other authors in any form e.g. ideas, equations, figures, text, tables, programs etc are properly acknowledged. A list of references employed is included.

Signed: _____

Date: _____


september 2005
The pursuit ability of a robot 'pet'
By Julien ROUX
id 040849205
supervisor: dr patrick green

Abstract

This report presents the development of a new behavioural pattern for a robotic dog, Sony Aibo ERS-7, which aims to find a form of prey to chase and eventually catch. In our case, the robot will wander around for a short period, then look for the pink ball and, once located, Aibo will chase it until it is close enough to what he considers the prey as caught. The general behaviour has been implemented using a hierarchy of Finite State Machines and covers several topics in robotics: object detection and recognition, navigation in an unknown and changing environment and, ability to follow a target. The idea of this project is to investigate how the robot achieves behaviour selection, based on stimuli from the environment. Several different approaches for decision making for autonomous agents have been studied and the state machine approach has been chosen for the implementation of the behaviour.

Acknowledgements

I would like to thank my supervisor **Dr Patrick Green** for his support at all times during the development of this project.

Also big thanks to **Peter Killisperger**, **Delphine Giner**, **Fabien Leboiteux** for their help and their support during this year, and **Clément Chazarra** for lending his camera for the experiment.

I also think to my **family** who have been a great support during all my studies.

Special thanks to **Ethan Tira-Thompson** from the Carnegie Mellon University and **Pr. Austin Tate** from Edinburgh University for their help, advices and availability.

And finally a very special thanks to **Lynsey Cormack** who supported me through this project and has been a precious help. It has not been easy. Merci Beaucoup!

This project has been really interesting and motivating. I enjoyed working on Aibo and in the end, it appeared to be a very good support for working and developing personal behaviour. At the begining of the project, I had many problems regarding the availability of Aibo, the memory sticks and the reader needed for the developement. Then I had problem to connect the Aibo and my laptop and finally, the environment showed some difficulties to be setted up correctly. But once this has been achieved, the developement was natural and intuitive.

The people working on Aibo all over the world are gathering their work to improve their capacities. Therefore, Professor Tate and Ethan Tira-Thomson have been a precious help as have been the users of the website Aibo-friends.com and the Yahoo ! group development mailling list dedicated to Aibo.

Thanks all.

Table of contents

Table of figures

For more details, see chapter 10 credits.

1 Introduction

This report is the result of a Master of Science project in Distributed and Multimedia Information Systems at the School of Mathematical and Computer Sciences of Heriot Watt University, Edinburgh. This work was conducted between May and September 2005 and is composed of two parts, this report and a program. The report focuses on how to implement a new behaviour for a Sony Aibo ERS-7, to simulate and test theories about the outcomes of interactions between entities when a group of animals hunt prey in an environment and how to implement them in the context of the Robocup Soccer challenge.

The Aibo robot is used as a platform to test the interactions during co-ordinated actions. The outcomes of this project should be used as a basis for building a framework for the Robocup soccer [Robocup, 2005] using similarities between the behaviour of a dog hunting for a prey and a football player looking for the ball.

1.1 Background and context

Nowadays, robots are not only considered as slaves primarily intended to dangerous, repetitive or difficult scenario [Rico, 2004]. Called social robots, they are now used for entertainment, with no other aim than to mimic a companion, be it human, animal or abstract, attempting to provide company to human beings [Kaplan, 2001]. The most popular of these entertainment robots is the Aibo, launched by Sony in 1999. This robot pet, resembling a dog, was created by the Digital Creature Laboratory for Sony in Japan after six years of research lead by Doctor Toshitada Doi [Jerome, 2004]. It basically fulfils the most current constraints arising when we consider autonomous robots. It can be active without being linked to a computer and it does not need a permanent plug to a power supply other than its battery.

Many researches about artificial intelligence and robotics have been done using Aibo [Rico, 2004] and one of the main outcomes is the Robocup [Robocup, 2005]. It is a tournament organized every year gathering impassioned of programming and robotics from all over the world and team from universities. The aim of this cup is have robots playing a full game autonomously. Each team uses the same physical platform for the Four-Legged Robot League, i.e. the Aibo, but two models are currently allowed, theERS-7 and the older ERS-210 [4Legged, 2005] [Bie, 2004]. Aibo robots were not intended to be football players at first. Thus, their entire behaviour has to be adapted to give good

results. Moreover, interactions between robots have to be improved to avoid the game to turn into a mess where every player would just run to the ball without playing as a team.

Robots are regularly used to tests theories about the outcomes of complex interactions between agents. This area of research is called 'Biorobotics' and can be seen as the intersection of biology and robotics. Creating a robot which is supposed to behave like an animal should take into account that the robot and the animal will suffer the same constraints: they are both behaving systems requiring an autonomous control system to interact with their environments and fulfil tasks. Therefore, robots are used in biology to model animals and their behaviour in a given environment [Webb, 2001].

Sony provides a framework called Open-r available for those who wish to design and develop new behaviours to improve the capabilities of the dog [Fred, 2004] [Open-r, 2005]. It allows the possibility of programming some specific features such as new movements and actions, while using varying music and lights. The Aibo robot simulates the behaviour of a dog: it can wander around, emit noises and seek care but even after more than ten years of development, the robot still requires massive improvement in terms of fluidity of movement and response to environmental interactions. At the moment, the dog can recognise its pink ball and its Aibone that it can play with, and its station for power supply when the battery is low. It has the capacity of recognising its owner visually and by voice and can read a collection of cards to receive orders [Aibo, 2005].

1.2 Objectives

1.2.1 Purpose of this report

This dissertation aims to study the pursuit ability of a robotic dog, Sony Aibo, in simulating the chase of a prey. An additional goal of this project is to study different models of behaviour selection and image recognition to design a system which is credible from a human point of view.

1.2.2 Scope

The main objective of this project was to design and implement a new behaviour to extend the capacities of the Aibo robot. The original aim of this project was to simulate a group of Aibo robots attacking a moving prey in a co-ordinated pack hunting style. This would have involved the robots recognising others from their group and moving to keep their necessary individual attack positions in their efforts to get close to the prey. This

aspired to be used as a basis for creating defence behaviour for use in the Robocup. Robots could behave as a pack to get to the ball or to the opponent and then achieve a better defence by allying efforts as a human team would do. Due to the short time allowed to realise this project, and also due to delay in having the robot and the development tools available, the scale of the project has been reduced to a single predator chasing a prey in an open environment. Finding the ball and controlling the action with respect to the target is the basis of hunting just as it is when playing football.

Therefore, it should be taken into account the different kinds of interactions that can happen between an animal hunting, the surrounding environment and the hunted prey. The main part of this project will be to study interactions between the robot and the environment and how the robot reacts to any kind of stimuli. We need to bear in mind that “a model is a representation of reality" [Lamb, 1987]: so the behaviour of the dog should model the reality of hunting a prey in an open environment.

1.3 Structure of the report

The report presents a new behavioural pattern that is designed to instruct the robot to explore the surrounding world, look for prey and, consequently, chase it.
The report begins with a background on the Aibo robot including its history and specifications. Additionally, this background covers a presentation of the Robocup and the work achieved by other teams taking part in the competition, along with different studies undertaken on Aibo and its capabilities. Chapter 3 discusses design problem analysis: it covers the alternatives and choices for behaviour selection and the software framework. Chapter 4 explains the technical background with an overview of the behaviour selection system, the software framework and the operating system running on Aibo.

1.4 Methodology

Several alternatives for the software framework and the behaviour selection system have been evaluated according to the project requirements. Following this, a system has been designed and implemented based on the results from the evaluation which has then been evaluated. This methodology is based on the “Evaluate, redesign and re-evaluate” model. This report presents the design of usability experiments and how they have been conducted. Finally the results are analysed with a presentation of further work to be undertaken.

2 Project background

2.1 Introduction

Aibo is an entertainment robot designed by Sony. This four-legged robot has the shape of a dog and is used to entertain people, but it is also used in research to test theories about the outcomes of complex interactions between agents. The model used in this project is the ERS-7M2.

2.2 The Aibo Robot

2.2.1 Origins of robots

For some time, robots were simply large, heavy machines fixed on a base. They were primarily intended for dangerous, dull, dirty, or difficult scenarios or to perform specific tasks, often repetitive or precise. Working with heavy or oversized materials was often the reason for using robots in industry. Research brought new possibilities and capacities for robots in terms of movement and wandering, they could now work in dangerous or inaccessible areas. For this, designers had to make them smaller, lighter and improve the technologies to make the robots work without being linked to a based power supply. One of the main issues in the research is autonomy [Rico, 2004]. Even if robots could be autonomous in terms of power supply, at least for a given time, they were still driven by humans, using keyboards or joysticks, and could not behave independently of man, especially not in unpredictable situations [Kaplan, 2001]. If robots were supposed to perform tasks, they needed a human to devise a meticulous step-by-step program and to think of any possible scenarios that could happen [Payen, 2005]. Thus, generating autonomous and unpredictable behaviours relies on writing the adequate program for the robots [Rico, 2004].

So robots were just tools and not used as human labour replacement. This notion introduced the concept of autonomous robots, which are easier to use as they do not need a permanent instruction from a man. The user can give an order to the machine, which then performs the task by elaborating its own strategy, trying to achieve it in its own way. It can receive information about its environment and to analyse it before interaction. Interacting with a robot means being able to communicate with it. It is a particular characteristic of humans and animals, and a great challenge when it comes to robots. That is why many of the robots were designed to resemble humanoids or pets.

Creating humanoid robots means having them behave in human environments with human constraints [Payen, 2005]. Thus, walking means being able to maintain equilibrium and to counteract gravity. It also means being able to perceive environments and obstacles on the path. Additionally, human communication is performed using gestures and sounds. So humanoid robots should be able to understand vocal orders and to recognise such gestures and other non-verbal communication. If a robot resembles a dog, it needs to be able to be trained the same way. Thus, if somebody presses on the back of the robot dog, it should sit down or if somebody presses on the chest, it should go backward [Yamada, 2004].

All of these are issues for laboratories who are trying to achieve this. Honda has created a robot capable of walking and running at a speed of 3 Km/h and able to communicate with a human through speech. Another laboratory, Kawada, has created a robot which is able to fall and recover from this fall without external help. Fujitsu has a laboratory working on a robot able to stand in equilibrium on its head.

2.2.2 Before Aibo

Robots were primarily used for work. Karel Capek, a Czech writer, invented the term 'robot' in 1920 in his novel "R.U.R." [Capek, 1920]. In this story, a genius called Rossum wanted to create an artificial creature to avoid people having to work. So he created the term robot from the Slavic word 'robota', which means work [Kaplan, 2001]. Therefore, robots were slaves dedicated to work and accomplishing tasks humans did not want to or could not accomplish. In more recent times, the entertainment industry took an interest in robots. They have created robots that can be useless if their existence is based only on satisfying the human need for company and entertainment. They aim solely to behave like a new entity in an existing world with an unpredictable and adaptable behaviour. This is when what is called "social robot" appeared. A social robot needs to interact with its environment. This would mean environmental interaction such as avoiding walking into things. The word social in itself has several meanings. It encapsulates the will to spend time with fellow humans and the capacity to adapt to the surrounding society [Ihse, 2004]. Thus, for a robot to qualify as being social, it needs to be able to adapt and change its behaviour according to its surrounding environment or the human user. Also, it needs to express some emotions to demonstrate its capacity to understand what is happening. This is why robots where created as virtual companions.

"*First, he wanted to create an artificial dog…*"
K. Capek, R.U.R. [Kaplan, 2001]

The first robot for company, even though it cannot be considered properly as a robot because it is only a computer simulation and has no effectors, was the Tamagotchi (figure 1). This small object, the size of a watch, ovoid with a screen and few buttons [Kaplan, 2001], was launched in 1995 by the Japanese company Bandai [Mystic, 2004]. It differed from other portable electronic equipment typical of this period, such as the Nintendo Game Boy, by the fact the user had neither goal nor aim. The Tamagotchi is just a virtual animal that comes to life, matures and eventually dies. The user is supposed to have some interaction with it, feeding it and cleaning the virtual environment the Tamagotchi lives in. But there was no real purpose apart from trying to keep it alive and watching it grow. This was the very first 'useless' electronic toy and even though it was fairly simple, it encountered huge success, especially from youth. Far from being a slave, it was, on contrary, rather a burden for the user. Some were even asking for the services of baby-sitters to take care of their Tamagotchi when they were unable to do so themselves. It quickly became an entity in the family like a pet and took an important place in some people's lives.

Figure 1- Tamagotchi [source: firebox.com]

Following the success of the Tamagotchi, several companies launched their own products and toys based on the same principle. For example, tiger had a very remarkable success in 1998 with a toy called Furby (figure 2). It was a kind of bird produced in several colours [Phoebe, 1998]. At first sight, it reminded consumers of the 'Duracell Bunny' which appeared in the battery company's advertisements during the 1970s and 1980s [Kaplan, 2001]. The main difference was the fact the Furby had some interactions with the user. It was seeking care and if nobody was playing with it when it was requested, it could start screaming and crying. In the end it would not grow up as it should, for example by refusing to speak a normal language and only trying to communicate in its

own fictional language called "furbish". This toy aimed to develop a relationship with its user and had no further goals.

Figure 2 – Furby [source: pinkangel15.tripod.com]

The first of these to really be considered as a robot was the Aibo ERS-110 launched by Sony in 1999. This four-legged robot resembled a dog and aimed to replicate animal behaviour [Roberty, 2005]. At first sight, it is a little motorised dog. It can play with a ball, wander around, likes to be touched and seeks attention. The action of playing is achieved by a set of patterns like the action of seeking attention. That is where the main difference between a robot and the animal it is imitating lies. All behaviours are modelled by a programmer, randomly achieved or are in function of the environment. Progress in artificial intelligence (AI) still does not allow for proper instinctive and unpredictable behaviour.

In order to be accepted as social, a robot has to interact with the surrounding world and needs some specific behavioural skills. Therefore, it has requirements which differ from those needed for industrial robots. Visual cues such as gestures or facial expression are very important in social interaction [Breazeal, 2000] [Ihse, 2004]. This highlights the importance for a robot to have a body. A social robot also needs believability. It should develop life-like qualities to make its behaviour realistic from the point of view of the human user. This means that the behaviour should not follow a given path, but be more adaptive to the interaction with the surrounding world. Therefore, it needs to respond adequately to human interaction. Another requirement is readability. The user should be able to understand the robot. This quality is in some ways linked to the believability. It means the sounds emitted by the robot along with the facial expressions, gesture and general behaviour should be understandable by the user [Ihse, 2004].

On the other hand, the robot should also be able to perceive social cues and respond to them according to implicit social rules. The reaction time should be adapted. If the robot reacts too quickly to a stimulus, the user could feel stressed and it might result in the robot being intimidating. If the reaction time is too long, the robot will be boring. Also, the robot should reflect its capacities by its appearance. According to the Uncanny Valley study published by Masahiro Mori in 1970 [Uncanny, 2005], users expect more capabilities from a robot with a humanlike appearance than from a robot looking unrealistic. From the human point of view, a robot which looks real yet lacks certain lifelike capabilities can be regarded as daunting.

Aibo is the first real entertainment robot by definition. It perceives the world around it through a miniature camera located in its mouth along with a set of microphones and touch sensors. It can thus perceive the surrounding world and behave accordingly. The most recent model also includes infrared and a range of sensors such as pressure, acceleration, vibration, and temperature for a better analysis of the Aibo's environment.

2.2.3 Aibo as an autonomous and social robot

The history of Aibo the robot pet began in the 1990s. Thanks to major developments in AI and image recognition, the idea of an autonomous robot could become a reality providing that the creation of lightweight, miniature components was feasible [Jerome, 2004]. The idea originally came from Doctor Toshitada Doi, who led Sony's Digital Creature Laboratory in Japan in 1992. The research and development started in 1993 and they had to face many factors that could be technical or related to the design of the robot. Never before the Aibo, had a walking robot been equipped with cameras. Movements during the walk were an issue as it blurred the image from the camera and thus image recognition was difficult to achieve. Also, the robot needed to be light and small, so components had to be reduced to a minimum size. Finally, the robot had to behave in environments within differing places, using varying lights and walking surfaces and be able to avoid obstacles whilst managing navigation

Figure 3 – Mutant [source: vieartificielle.com]

The first prototype was demonstrated in 1997. It was called Hexapode and it displayed a grand performance after five years of intensive research. Robots using wheels were common and could behave using cameras to obtain information from the surrounding environment, but this was a major issue for legged robots as the camera was not stable nor could it produce clear images. This was problematic for image recognition and object detection thus constricting the robot for using this information. The main idea was to have the robot moving on six legs to obtain smooth motion avoiding disturbances for the camera [Frederic, 2004]. Then, in 1998, Sony presented to the press a new model called Mutant (fig. 3) during the Robocup in Paris. It was walking on four legs and had sixteen degrees of freedom for a weight of approximately 1.25kg with batteries. It was the very first prototype of this kind and it made a huge impression as it was behaving freely even though it still encountered some technical problems. In particular, the prototype had difficulties for image recognition. In a laboratory, with walls uniformly white, recognising an object was easier than in a house with varying environments. The ball it was following was orange and it was sometimes difficult to identify, and that is why the colour has since been changed to flash pink as it is far less used in traditional houses.

In June 1999, Sony launched the first commercialised model with the denomination ERS-110 [Jerome, 2004]. The design was made by Hajime Sorayama, a very popular Japanese artist [Sorayama, 2005]. This first version was sold only on the Web in Japan and was limited to 3000 units. They sold out in twenty minutes. In the United States, where 2000 units were stocked, every single one was sold after four days [Roberty, 2005]. In November of the same year, a new model denominated ERS-111 (fig. 4) was commercialised. It was very close to the previous one, with minor modifications, for example the tail had been reduced as it was too fragile. There were 10,000 units produced but it was far from enough for the 135,000 unit demands.

One year later, in November 2000, Sony unveiled the ERS-210. The shape was quite different with smaller ears and thinner lines. It was an immediate success due to its design and to the large selection of software available. One of those, called "Aibo Life", introduced the concept of educating the robot from the behaviour of a puppy to that of a grown up dog. The behaviour could differ significantly depending on how the robot was treated. Another improvement was on the technical side. At the rear of the machine, a PCMCIA slot allowed the installation of a WIFI card for remote control from a computer. It was the first of the Aibos to be produced in an unlimited series and this helped for its success. Soon, it would become the mascot of Sony, appearing in advertisements on television and printer to promote other products from the company.

Figure 4 - Aibo ERS-220 [source Aibo-fr.com]

Two new models appeared in 2001. They were called Latte and Macaron from their colours; white and maroon. These two robots were less expensive than the ERS-210 and to achieve this, designers had made them smaller, simpler and modified their design to look more like teddy bears. The success was lower than expected. After this, the ERS-220 was launched in November 2001 (figure 4). Very different from the previous ones, its shape abandoned the idea of trying to look soft and turned to be more robotic rather than puppy like. The main targets for this model were teenagers and young adults. This model was more curious than previous ones, investigating, wandering around, and it used lights to communicate with users. After this, both ERS-210 and ERS-220 were upgraded in 2002 for better performances. A new model called ERS-31L joined the series being based on the same idea as Latte and Macaron.

Last of the series of the Aibo is the ERS-7 (figure 5). This model of the four-legged robot manufactured by Sony was released in September 2003 [Bie, 2004] [Sony, 2005]. A major improvement, when compared to the previous editions, lies in the new degree of freedom on the neck which allows Aibo the movement necessary for 'playing' with the toys such as a ball and bone shaped toy Aibone. Also a WIFI connection was incorporated to the system for remote control. The system is controlled by new software called Aibo Mind. It controls the behaviour of the robot and all the applications related to it. The new version of the software called Aibo Mind 2 came out in October 2004 with the ERS-7 Pearl Black which keeps the same design but with a new pearl black colour. This version of the software is a great improvement in the behaviour of the robot dog: it is no longer limited to simply pushing objects; it can simulate playing with them.

2.2.4 The Aibo ERS7-M2

This document will now only refer to the ERS-7 model as it was the one used during the development of this project.

Figure 5 - Aibo ERS-7 White and Black Pearl with the pink ball and the Aibone [source: converj.com]

The Aibo is a complex machine composed of a wide range of sensors and effectors. It has the capacity to feel, hear and view its surrounding environment. The feeling is achieved using two kinds of sensors: electrostatic and pressure [Bie, 2004]. There is one electrostatic sensor on the head and three on the back. They light up when they perceive a contact. The pressure sensors are under the paws and the chin. It allows Aibo to detect if the four paws are on the ground or if something is under the chin, out of range of the camera. Also, acceleration, temperature and vibration sensors give Aibo a more accurate rendering of the environment [Sony, 2005] [Rico, 2004]. Acceleration is measured to prevent the robot from falling down due to acceleration when moving or to detect if the robot is abruptly stopped, for example, when encountering an obstacle or being grabbed by the user [Technostuff, 2005]. Vibration sensors analyse linear velocity, displacement and proximity and are also used in measuring acceleration [Globalspec, 2005].

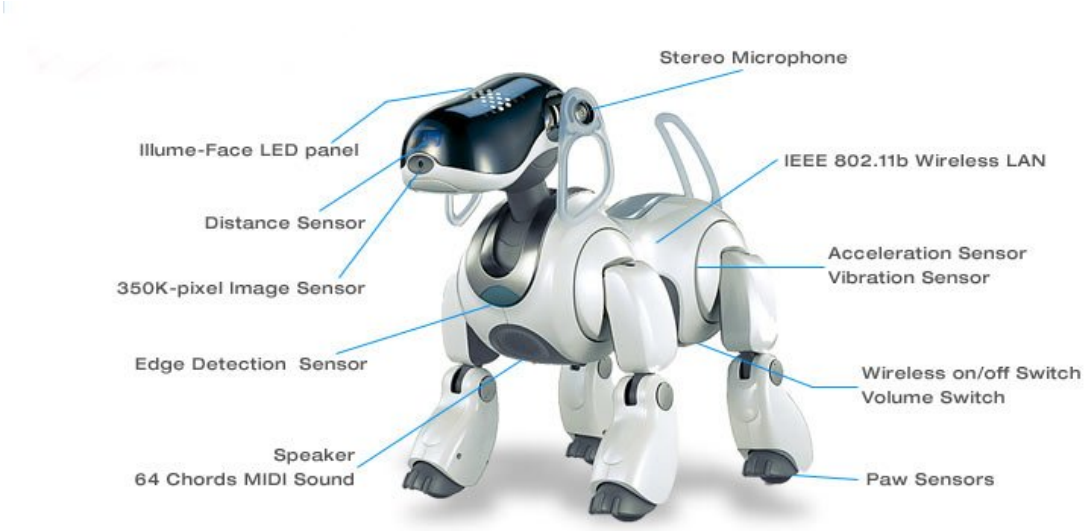


Figure 6 The hardware of the Aibo ERS7, front view [source: Sony website]

The robot can see through a camera located in its nose. It has three resolutions 208x160, 104x80 and 52x40 for a horizontal angle of view is 56.9 degrees and a vertical one of 45.2 degrees. Two distance sensors placed in its nose and its chest, with operating distances from 10cm to about 90cm, help gather information about the presence of obstacles. Two microphones located on its head would function as ears achieving the dog's hearing. The sound is recorded in stereo at 16.000Hz in 16bits linear pulse code modulation [Onrobo, 2003].

Figure 7 The hardware of the Aibo ERS7, rear view [source: Sony website]

To interact with the surrounding environment, whether it is the human user or simply the ball or the Aibone, Aibo uses several effectors for movement and communication. The movement is performed using four legs with three joints in each to elevate, rotate and bend. The neck is also composed of three joints to tilt, pan and nod. The tail has two joints to tilt and pan. The mouth has one joint to be opened and closed. The ears also move when flicking up and down [Bie, 2004] [Sonystile, 2005].

To communicate, Aibo uses three types of effectors. They are visual, audible and wireless. A wide range of LEDs with several colours are located on the forehead of the robot. Also the electrostatic sensors light up to express feelings. The sound is displayed by a miniature speaker on the chest playing polyphonic sounds. The wireless connection is an IEEE 802.11b wireless Ethernet interface with a range to up to three hundred feet.

2.3 The Robocup challenge

The Robocup is an international tournament aiming to promote artificial intelligence, robotics and related fields [Robocup, 2005]. By providing a standard problem to different research groups, it intends to further knowledge and skills for specific IT related problems for the benefit of all the researchers involved. It compares techniques and methods used to choose the best solutions. The main challenge takes place in the form of a soccer game. The goal being that the innovation gained from the challenge can then be applied in industry for every-day life. The final aim of this project is to develop by 2050 a team of fully autonomous robots which could play and win against the human world champions under the official rules set by the FIFA (Fédération Internationale de Football Association, the organisations setting the rules for official competitions around the world). Various technologies are used during the contest including design principles of autonomous agents, multi-agent collaboration, strategy acquisition, real-time reasoning, robotics, and sensor-fusion [Robocup2005, 2005]. There are three major competitions during the Robocup: the RoboCupJunior, the RoboCupRescue, and the RobocupSoccer.

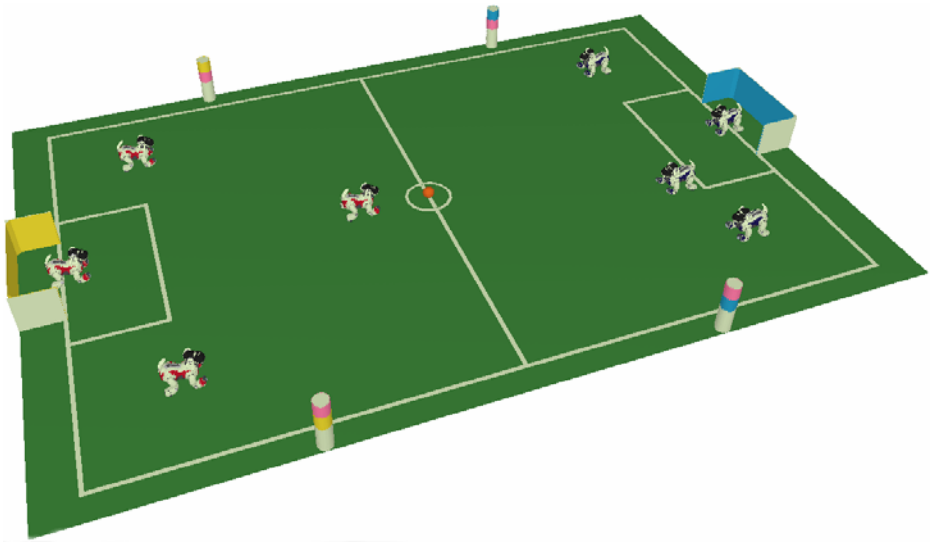


Figure 8 Field with players for the Robocup 2005 [source: Rules of the competition 2005. Available on the Robocup website]

The RobotCupJunior began in 1998 with a demonstration at Robocup-98 in Paris. It is a project-oriented educational initiative intended for young students [RobocupJunior, 2005] [Robocup2005, 2005]. There are several challenges during the competition: students can compete on soccer, dance, and rescue. The RoboCupRescue aims to gather research on rescues for large scale disasters. It is an effort to gather technologies on autonomous agents able to act in dangerous and barely accessible areas [RobocupRescue, 2005].

The RoboCupSoccer is the soccer challenge. There are five different leagues in this challenge. The simulation league concerns a computer simulated soccer game. The small robot league is for robots with a maximum diameter of 18cm and maximum height of 15cm. The middle size league is for robots with a maximum diameter of 50cm and a maximum height of 80cm. The humanoid league is for bipede robots with human appearance. The last league is the four legged league [4legged, 2005]. This one concerns the Aibo robots. They are competing against each other in a soccer game. They have to be autonomous and act as a team. There are three field players and one goalkeeper in each team. The field is 5.4m by 4m wide with goals 0.8m large. To allow robots to locate themselves and each other on the fields, there are coloured landmarks on the side indicating with a specific pattern where they are. The robots can be the Aibo ERS210 (or the evolution ERS210A) or the Aibo ERS-7 (or the evolution ERS-7M2).

The basic rules are similar to human soccer, with two halves of 10 minutes each and a break of the same length between. There is extra time, golden goal penalty shoot-out as in normal competitions. Some rules have been introduced regarding the physical and

mental capabilities of the Aibo. Pushing and obstructing are forbidden as it is for a field player to enter with more than two paws in the penalty area. It is also forbidden to hold the ball for more than three seconds for a field player or five seconds for a goalkeeper. If a robot commits a foul, the referee will remove it from the field for 30 seconds.

In addition to the soccer competition, there are three technical challenges open to the team. The variable lighting challenge intends to improve the vision capabilities of the robots by providing a penalty shoot-out competition with changing lights. The robot has three minutes to score as many goals as possible with two immobile opponents on the field and the light changing in an unspecified way. The "almost SLAM challenge" tries to improve the self localisation capabilities of the robot. The robot has one minute on the field to get itself localised and then the landmarks are removed and the robot has two minutes to get to a series of points on the field. The last challenge is the open challenge. Each team has three minutes on the Robocup field to demonstrate their work. The entrants vote to elect the winner.

2.4 Review of previous research on Aibo

Aibo robots are regularly used in projects for academic institutions all around the world. Many of the research projects focus on the Robocup or intend to simulate and test theories about the outcomes of interactions between entities when 'pack-hunting'.

2.4.1 The German Team

The German Team gathered students and researchers from four German universities: the Humboldt-University of Berlin, the University of Bremen, the Technische University of Darmstadt, and the University of Dortmund. They won the cup in 2005 in Osaka [Germanteam, 2005]. The work from the German Team is one of the most complicated for somebody not experienced. The have participated since 2001 and have encountered great success. They provide good publications about the Aibo and how to program and improve its abilities.

First, the architecture produced by the German Team is platform independent to be able to run on several models and different Robocup leagues. This has also lead to the development of a simulator which can be used to test the code without having to boot the robot. This can save a lot of time when developing a piece of code for the Aibo. Alongside of the platform independence, the project needed to support multiple team

work. Therefore, the architecture includes the notion of entity to support the collaboration between universities. Each entity, called module, has its own interface and a specific purpose. They are also exchangeable. This way of programming allows the changing of one module in the whole architecture without affecting the others.

To build an image of the surrounding world, the German Team has decided to split the information from the sensors in two groups. The first group takes information from all the sensors except the camera and keeps it in a buffer to calculate an average sensor value over a given lap of time or to access information at a given time and put it in relation with other information, usually an image from the camera. The vision module performs the processing of the images from the camera. It uses a technique called "horizon aligned grid" developed by the Humboldt University of Berlin [Juengel, 2004]. This technique determines by calculating the position of the horizon which is then used to determine the robot position and those of the opponents. The behaviour control is performed using an improved version of the Extensible agent behaviour specification language XABSL [Lötzsch, 2004].

This work represents a huge collaboration between developers over a country. This shows how a cooperative work should be conducted. The technology used has been proved to be successful and the architecture is well defined. Additionally, the team's official website delivers good information for beginners.

2.4.2 The Team Chaos 2004 (formerly Team Sweden)

Team Chaos has been competing in the Robocup since 1999. It is a collaboration between the University of Murcia, Rey Juan Carlos University in Madrid, University of Alicante from Spain and Örebro University from Sweden. Their work is focused on the creation of a behaviour based control for ERS-7 robots and is based on the Tekkotsu framework developed by the Carnegie Mellon University. This framework is going to be explained in detail in the chapter 4.3. They intended to make something fast and robust with exchangeable modules to ease cooperative work. The framework is composed of six modules: Tekkotsu, Vision, Localization, Wireless, Behaviour and Worldstate [Bie, 2004]. The code written for the project is supposed to be general enough to be reused in many different applications. This study compared different approaches for decision making on autonomous agents. They compared the possibilities of using state machines approach, hybrid automata, hierarchical structure of decision trees and symbolic planning. The framework is based on a finite state machine approach. It is part of the reactive paradigm regularly used in robotics and computer science. A reactive paradigm is a way to simulate intelligent behaviour by having a direct link between sense and act. This is a

biologically based behaviour [Murphy, 2004]. In a finite state machine, the condition of the robot or the content of the memory at a given time is represented by the states. The Moore machine, used as a finite state automaton has the outputs defined by the current state without considering the input [Moore, 2005]. In this case, the output is the corresponding behaviour. Therefore, the state of the machine represents the behaviour.

They finished the tournament at the last position of the qualification but their work is interesting as they started a new behaviour from nothing and have it well documented [Bie, 2004]. The Team Chaos took part in the Robocup 2005 and had better results even though they were not ranked.

2.4.3 The NUbots 2004

The NUbots from the University of Newcastle in Australia [Newcastle, 2005] have taken part in the Robocup and finished 3rd in 2002, 2003 and 2004. They also took part in the recent Robocup in Osaka and finished 2nd, beaten in final by the German Team. The framework is based on a sense-think-act cycle architecture which is another planning strategy [[NUbots, 2004]. In this architecture, the robot receives information from the sensors, then analyses them and tries to plan the best way to achieve the goal. Then the action is undertaken. This architecture will be explained in detail in chapter 3.2.2. As with Team Chaos, NUbots have created a whole framework for their robots, but with more success. Therefore, the vision is one of the major areas in the project. This is very important as determining the position of the ball and the position of each robot remains a weakness for all teams at the moment. Their work concentrated was especially on robot recognition and multiple lookup tables. The robot recognition is used to determine where the other robots on the field are thus it will have a better set of information when deciding which action to undertake whether it be for defending or attacking purpose. Multiple lookup tables were used to have a better precision for colour recognition. It can then adapt to different lighting or shadowing: for instance, avoid having the orange ball used in the 2004 competition being mistaken for the red team stickers on the robot team members. Finally, the NUbots worked on extra landmark recognitions like sidelines or penalty box. This is another way to help self localization on the field.

This work is interesting as it shows a team that has great success in the competition. It also shows another planning strategy to be analysed.

2.4.4 Top dog

The project Top Dog is conducted at the Rennselaer Polytechnic Institute [RPI, 2005]. It is the first time this university is taking part on a project like this using Aibo. This project is supposed to be a basis for a future framework which could be used in Robocup. They started on the statement that the team-work among Aibos was minimal if not inexistent. They noticed that for most of the teams, attacking was performed by one of the robots while the others were sitting in defence. Thus, they aimed to have robots passing the ball to each other in order to have an actual cooperation between them to achieve the goal. They intended to use finite state machine architecture to enable Aibos to keep track of each other and the ball and to respond to occurring events [Top dog, 2005]. Their project needed to have a revised goal since passing appeared to be more difficult than first anticipated. The final project succeeded by having two robots passing the ball back and forth to each other. This implies looking for the ball as well as the target and performing an efficient kick to actually force the ball to the other robot.

In parallel, another project was also conducted at same university. This group was working on the way the Aibo walks. They intended to implement a new way of running which closer resembled the way a dog would run and would therefore be more efficient and quicker. They intended to mimic natural animal gaits to have the robot reaching more than the 180mm/s performed by the Tekkotsu gait. They intended to have the robot perform a trot which could be a lot quicker than the former way of walking. They could not implement an upright gait and finally further developed a crouched gait.

This project is quite interesting because it shows how a project has been conducted since the beginning and all the difficulties that can be encountered. It also demonstrated the innovation that the Robocup can bring to the programming environment and all the major improvements needed in the soccer simulation.

2.4.5 Dog-like Behaviour Selection by Christina Ihse Bursie

This master thesis report, written by a student from the department of numerical analysis and computer science from the Royal Institute of Technology of Stockholm, Sweden, is about dog ethology and behaviour and how to implement this on an Aibo robot dog [Ihse, 2004]. The work relies on the Breazeal model for the Kismet robot from the MIT [Breazeal, 2000] and the Tekkotsu framework. The selection of the appropriate behaviour is based on the perception and the current emotion of the robot. This work intends to get a close representation of the emotional behaviour of a dog. A large part of the project is about the perception and the distance to the objects. The study reviews dog behaviour in different kind of situations such as dominance, fear, or friendly

interaction. Behaviours have then been reproduced with the Aibo robot. This work is mainly focused on how to choose the right emotion to coincide with particular circumstances and stimuli.

Therefore this project is especially interesting because Team Chaos made use of Tekkotsu using another planning strategy and incorporated testing theories about animal behaviour.

2.5 Experimental design

The main point in this project was to assess the ability of the robot to recognise objects and to behave adequately. The robot should be able to recognise two types of object and to behave according to them. The robot should pursuit a 'prey' and stop when it has 'caught' it. The 'prey' will be simulated by the pink ball moved by a human person.

The Aibo should be able to take the adequate decision according to the situation after analysis of the environment and evaluation of the possibilities. The robot should stop chasing the ball once it has 'caught' it, i.e. once the robot touches the prey with its mouth or at least when the ball is between the paws of the robot. Also the robot should look again for the 'prey' after a given time.

2.6 Project plan and risk management

The project is composed of a succession of studies, each being a follow up from the previous one. Thus, the risk is limited as the failure of one study will just bring the need for another one. In case something would go wrong, the topic is wide open to perform research in other directions.

Technical failure is possible. It can happen with any problem concerning the Aibo available from university. In this case, the project would move from programming to full study of the capacities of the robot by analyse of the code and simulation.

3 Design problem analysis

3.1 Introduction

An autonomous agent willing to perform an action and aiming to fulfil a goal in a dynamic environment, has to adapt its behaviour by choosing the correct action at the appropriate time to either respond to a specific stimulus or change in the surrounding environment.

> "Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment and by doing so realize a set of goals or tasks for which they are designed."
>
> Pattie Maes [Maes, 1995]

The world is too complex to be completely predicted and thus not all possibilities can be programmed and fed into the robot. Furthermore, agents are limited in terms of time and capacities when calculating the next action to perform. Thus, action selection is never optimal but should be as fast and efficient as possible [Maes, 1990].

3.2 Alternatives for the behaviour selection

3.2.1 Introduction

First, we have to understand what a behaviour is. It can be seen as many things: moving an arm or grabbing a pen. The action to get up in the morning and go to work can also be seen as behaviour. Indeed, even the absence of action is behaviour. The dictionary gives the following definition:

> "**Behavior** (or **behaviour**) refers to the actions or reactions of an object or organism, usually in relation to the environment. Behavior can be conscious or unconscious, overt or covert, and voluntary or involuntary."
>
> Definition on Wikipedia [Wikipedia, 2005]

Concerning the Aibo, or any other kind of autonomous agent, a behaviour refers the same way to an action or a sequence of actions that accomplishes something. The agent has to decide when, how and which action it should perform. It can be seen as what

should be done under given circumstances or under which circumstances a given behaviour should be activated. Therefore, as it is impossible to predict all that can happen in an open environment, the second solution will be more suitable. Many solutions have been designed to implement decision making for autonomous agents. Behaviours, be they in robotics or zoology, can be divided into three categories. Reflexive behaviours are direct reactions to the input without the control from a third party. For instance, eyes will instinctively maintain a level of moisture supply to keep from drying by means of blinking. Reactive behaviours are also used without real conscious awareness but, unlike reflexive behaviours, these are learned. Humans learn how to ride a bike or how to swim and, once learned, they will perform such activities without thinking about how to do it. The last type of behaviour is the one most complex. Conscious behaviours imply that the agent performing them has to be conscious of planning them, for instance, when playing a game or cooking.

Researchers in Artificial Intelligence spent decades trying to model human-like intelligence. The focus was on knowledge-based systems with symbolic representation and thus, all research was based on human-like approaches. The strategies developed under this paradigm were quite elaborate but they lacked autonomy and were often unable to cope with a changing environment [Bruemmer, 2005]. Therefore, intelligence was seen as the strategy capable of processing information while, according to Arkin [Arkin, 1998], a behaviour-based approach would make proof of its intelligence through meaningful and purposeful action in a given environment. Additionally, the behaviour-based approach does not intend to represent cognition or a human-like decision process. As shown by Brooks [Brooks, 1986], humans are trying to recreate something they are not fully aware of and proposed one of the first behaviour-based strategies. He argues that cognition is a reaction to information received through an observer and biased by its perspective on the environment [Brooks, 1991]. A behaviour based control system consists of a collection of behaviours [Mataric, 2004]. Each of them can receive stimuli from sensors or from other behaviours as an input and send outputs to the effectors or other behaviours. Therefore, it can be seen as a structured network of behaviours interacting with each other. This feature does not constrict behaviour based systems in their expressive and learning capabilities. Systems based on a behaviour architecture attempt to make a representation with a uniform time-scale. Representations are parallel, distributed and active which accommodates the requests in real time from other behaviours or other parts of the system.

The design of behaviour based controls for autonomous agents is also subject to debate. Three kinds of architectures have been used since the beginning of artificial intelligence

design. They are defined by the behaviour selection system used to make the robot behave without following a precise and pre-programmed order [Logan, 2004]. The hierarchical architecture, also called deliberative architecture, has been used since the late 1960s with 'Shakey the robot', which was the first agent to demonstrate reasoning about its action [Logan, 2004]. It was based on a 'sense then think then act' design. The idea was to plan the next action by generating alternatives before choosing between them. After this, the reactive architecture appeared in the 1980s. This new architecture removed the planning part considered as too consuming, either in terms of resources or computation. Sensor readings are directly translated into actions. Finally, another type of architecture appeared in the 1990s merging the previous designs. Hybrid architectures decompose tasks into subtasks and then decide what the suitable behaviour for each subtask should be.

Moreover, having multiple agents perform a task together whilst showing signs of collective behaviour is a big challenge. They have to avoid unpredictable or harmful behaviour yet they lack a global point of view of the problem [Sycara, 1998]. This criterion means that when several agents have to cooperate, such as during the Robocup, the design becomes more complex and the agents have to be organised.

3.2.2 Deliberative architectures

The principle in a deliberative architecture is that the agent contains an internal model of the surrounding world, and a planner [Ihse, 2004]. The planner is used to decide which action should be performed next, using short and long term predictions of the outcomes from this action. These predictions are based on the world model the agent has created from information, either fed into the robot in advance or gathered during the previous actions. This model can also be updated using new sensor readings.

The world model can be of many types: such as a map of the surrounding world, a list of objectors' knowledge of how to use a particular object. This can also concern knowledge of how the world is perceived under different conditions [Ihse, 2004]. But the model depends also on the robot's limits in terms of calculation capacity and time, sensor accuracy and also what needs there are. Deliberative architectures depend on changes to the surrounding world. The model has to be very accurate otherwise the model can be a source of faulty predictions. On the other hand, the model cannot be too precise or else it could be too time-consuming to update and be difficult to use in a changeable environment.

Deliberation is about considering several alternatives for one action. Therefore, the agent has to generate different alternatives in order to choose between them. However, the agent does not always solely decide on how to achieve a goal but sometimes whether or not to achieve it at all [Logan, 2004]. This is the work of the planner. Its structure often follows a hierarchical model with modules on different levels. The world model is structured in the same way: lower levels usually deal with the sensor readings while higher levels work on the global knowledge. Likewise, the planner has lower modules used for real time and focusing on nearest parts of the world while global problems needing a solution less urgent are treated by higher level modules.

3.2.3 Reactive architectures

In a reactive architecture, the agent does not have a representation of the world: instead it responds to a stimulus. An agent is constantly calculating how to perform the next action but it does not keep history of what happened nor are plans made on any further action even though we can find use of short term memory in some reactive architecture [Ihse, 2004]. Therefore, stimulus and response are closely coupled and can exhibit complex behaviour from simple interactions with other agents. Additionally, the agent does not need to generate a model of the surrounding world and record the changes, making it more robust and fault tolerant. Another quality is that it allows behaviour development in small incremental steps.

A reactive architecture will be suitable when the environment is unpredictable or cannot be properly modelled [Ihse, 2004]. However, a reactive architecture has its limitations. As it has no knowledge of the world, it is heavily dependant on the sensors and if something is not caught by a sensor, the robot will be unaware of it. Hence, it cannot recover from silent faults, i.e. stopping abruptly without sending any signals to the main system. Therefore, the perception the robot has of the world is the input for the behaviours. A behaviour will react to a set of stimuli, and only to these, and respond by performing an action. If the stimulus is too weak or wrong, no action will be produced. Furthermore, the agent only takes decisions based on local information brought by the sensors and cannot take into consideration the global behaviour. Thus, the relationship between the agents considered as individual, the surrounding environment and an overall behaviour cannot be predicted making it difficult for agents to work together as a team.

Notions of reactive architecture first appeared in Brooks's criticism of deliberative architecture [Brooks, 1991]. He noticed that:

> "Intelligence is the product of the interaction of an agent and its environment and intelligent behaviour emerges from the interaction of various simpler behaviours organised in a layered way through a master-slave relationship of inhibition"
>
> AI Magazine, summer 1998 [Brooks, 1991].

Another way to see a reactive architecture is to design it as the result of a group of entities competing to get control of the robot. This was first proposed in the **society of mind** published by Marvin Minsky [Minsky, 1986] [Sycara, 1998]. Tasks are set in conflict and only one can be active at a time. In this case, a task is seen as a high-level behavioural sequence. A coordinator is used to decide which action should be carried out next. It can be either competitive or cooperative. A cooperative coordinator will allow several behaviours at a time to affect the robot usually using superposition. A competitive coordinator will select one behaviour which could be triggered by one stimulus and reject the others. The selection is performed in one three ways. If the behaviour is organised in a hierarchy, the coordinator will choose the behaviour with the highest level. In action-selection coordination, the coordinator assigns activation levels to the behaviours according to both the robot's goal and the external stimuli; consequently, the one with the highest level is activated. Finally, the coordinator can let behaviours vote for each other's behaviour responses and the one with the most votes is chosen.

3.2.4 Hybrid architectures

Completely removing the planning section appeared to be a bit drastic. Reactive architectures can be more efficient and flexible when incorporating certain forms of knowledge. That is how hybrid architectures were created, which appeared in the 1990s [Bie, 2004] [Ihse, 2004]. Thus, reactive behaviour can be configured according to some knowledge of the surrounding world. Some known problems or weaknesses of reactive architectures can be handled by adapting the behaviour to the world. A major problem when it comes to reactive behaviour is their tendency to get into a deadlock. When subject to two opposite and equally strong stimuli, the agent cannot react.

Hybrid architecture consists of three parts: a reactive layer, a planner and a layer linking them together. It is suppose to be a combination of the two time-scales and representations from a reactive and a deliberative architecture with a middle layer. Thus, hybrid architectures are often defined as three layer architectures. Many implementations of hybrid architectures have been designed and, according to Nwana [Nwana, 1996], they generally suffer from two high level problems. First, most of the architectures created with a hybrid model appear to be application specific. Secondly, the theory on

hybrid systems is still vague and designers will have differing ideas of it. Additionally, the scope about hybrid architectures is quite narrow. Many other possibilities rather than reactive/deliberative should be explored, for instance combining interface and mobile agents together [Nwana, 1996]. [Not clear what this means – say more]

3.3 Choice for the behaviour selection design

The Aibo is supposed to behave in a rapidly changing environment. It shall adapt quickly to unexpected events and react accordingly. Researchers still debate over the interest of having an internal model of the surrounding world. Some think that robots cannot mean any of their actions without having a representation of them with semantically related contents [Pylyshyn, 1988]. Others, like Rodney Brooks [Brooks, 1987], believe the planner is not compulsory and is just an abstraction barrier, as it is only used to bring a level of abstraction for the system designer and is not really used by the robot. In our case, the strategy does not require a built model of the world and needs to operate quickly. This makes the pure deliberative architecture less adequate and brings the interest on a reactive architecture with a control paired between stimulus and response.

If the environment where the agent will behave is bounded and known by the strategy designer, it is possible to create a set of stimulus-response pairs covering all possible events [Bruemmer, 2005]. This kind of approach can be applied to our system if the robot is behaving in a laboratory or an empty space. It becomes too restricted when considering the Robocup and it appears necessary to find some way of compromising between reactive and deliberative architecture. Having no internal representation at all bans the system from tasks having to deal with memory while a system relying on an internal model will be subject to errors in a new environment.

In the architecture developed by Brooks in 1986 [Brooks, 1986], the reactive system is structured from the bottom up with layers and related rules. It shows that some basic behaviour, such as avoiding collision take precedence over some others like achieving a specific goal. Therefore, the first one should be on a bottom layer with highest priority while the second should be on a high level layer and might be built from behaviour located on lower layers or by satisfying them. A key point in this approach is that the interactions between behaviours should be kept to a minimum. Each of them should behave simultaneously but asynchronously and independently to reduce interferences.

The main point in designing such architecture lies in the behaviour selection called arbitration. Even if the behaviours have been assigned priorities by layering the system, a coordinator should be used to manage the transitions between the states. Rosenschein

and Kaelbling defined in 1990 [Jennings, 1995] a reactive system where finite state automata are used as a strategy to identify stimulus and react to events. The aim in this strategy was to have real-time control in a bottom-up design with both the capacity of learning through incremental growth and a tight coupling between sensors and effectors [Bruemmer, 2005]. Representation is avoided but there is a link between symbolic meaning and action. Using hybrid approaches, designers have attempted to subordinate planning to reactivity as a way to guide the system at a high level. According to Arkin in 1989,

> "The false dichotomy that exists between hierarchical control and reactive systems should be dropped"
>
> Arkin, 1989 [Bruemmer, 2005]

As seen in the autonomous robot architecture called AURA, designed by Arkin [Ina, 2004], it is possible to design a representation based hierarchical system working whilst coupled with a reactive component. This validates the decision to use hybrid architecture to design the project. The agent will be in several states with different aims. It should be hungry or satisfied, aware of the position of the prey or not, and close enough, or not, to feast. It can also aim to find, chase or eat its chosen prey. One way of designing such a system is to use a finite state machines model, as was developed in Robocup by Team Chaos in 2004 [Bie, 2004]

3.4 Alternatives for the software framework

3.4.1 Introduction

Many development environments have been developed for the Aibo robot. They all have different capacities and are used for different purposes. The following is a review of those most common.

3.4.2 R-Code SDK

R-Code SDK is a tool provided by Sony to allow programming using the R-Code language. This language is dedicated to the ERS-7 model and tools can be downloaded free of charge from Sony's SDE website [Sony, 2005]. A program designed using the R-Code language is a list of commands that the Aibo can understand and closely resemble BASIC programs [Tellez, 2004]. The user needs the Master Studio delivered with the Aibo but there is a wide range of software available to develop some R-Code scripts without much trouble [R-Code, 2004]. Among them, YART, which stands for 'Yet Another R-Code

Tool', appears to be the most simple. It can be used by beginners and allows the creation of behaviours for the Aibo, using a drag and drop technique without the need for any programming experience [Yart, 2005]. AiboPet's Performance Editor and Skitter Performance Editor can also be used in replacement of the Master Studio.

The fact that the R-Code is a scripting language makes it easier to write commands using a simple text editor, but it also allows high level commands to be sent. Therefore, the language is easy to learn but with drawbacks of having narrower possibilities for controlling the robot than with a typical low level language like C++, even though it still allows the implementation of complicated behaviours [Tellez, 2004]. For example, Aibo can be driven to sit, stand, lie, walk, turn around, track its ball, kick and find a face or the Aibone.

3.4.3 URBI

URBI stands for 'Universal Robotic Body Interface'. This library allows the control of any kind of robot using a powerful script language protocol. The programming can be done by implementing the adequate library with one of the several language supported (C++, Java and Matlab) under several operating systems (Windows, Mac OSX, Linux). It uses client/server architecture, meaning the robot is linked to the computer used as a server and acts as a client to it. URBI is a low level programming language which directly reads and sets sensors and effectors. It uses time oriented control mechanisms and command separators to parallelise commands and serialise them [Baillie, 2005]. Behaviours are created with event based programming. For the user, the main advantage with URBI appears to be its simplicity: the system is supposed to be understood in a few minutes. Furthermore, the URBI system and the libraries are distributed under a GNU Licence allowing anybody to work with it.

3.4.4 The OPEN-R SDK

OPEN-R is Sony's interface to the Aibo hardware allowing users to program behaviours for the robot [Open-r, 2005] [Ihse, 2004]. This extends the capabilities of social robots. It is a modularised and object oriented software kit based on a layered architecture. It allows concurrently running objects to communicate with each other and supports Wireless LAN and the TCP/IP protocol [Open-r, 2005]. The OPEN-R SDK is the cross development environment for OPEN-R based on the 'gcc' compiler for C++ and can be downloaded free of charge from the Sony's SDE website. The environment can run under Linux or Cygwin for Windows. The OPEN-R API allows access to the low level functions of

the robots by writing programs charged into the memory card and then into the robot. It is possible to move joints, turn the LEDs on or off, play audio data, acquire information from sensors and receive images from the camera.

The OPEN-R architecture is composed of two layers: the system layer and the application layer. The system layer is used as an interface to the robot hardware using a range of services. These services are provided by three objects from the system layer: OVirtualRobotComm is used to handle the sensors, effectors and the camera: OVirtualRobotAudioComm handles the audio communication and: ANT is used to perform the TCP/IP communication. The program is written in the application layer which interfaces the system layer to access the robot's hardware. It is used as an input to the system layer's services. Once they have processed the last sent data, they send an event to the object indicating their capacity to receive data.

The system layer is composed of OPEN-R modules called objects, but these are not like the standard objects found in C++ even though they have the same basic functionalities. They contain some extensions such as multiple entry points. All objects communicate via messages passed through inter-object communication using pre-defined communication channels. The object sending the messages is called the subject whilst the module receiving them is called the observer. All OPEN-R programs are constituted of one or more concurrently running modules. As the system is modularised and object-oriented, there are no difficulties made when replacing a module, subsequently, there is no need to recompile the whole program. Another beneficial feature of OPEN-R lies in the 'Remote Processing Utility' which makes it possible to execute parts of the program on different platforms for testing purposes.

3.4.5 The Tekkotsu development framework for Aibo

Tekkotsu is a high level application framework developed in C++. It can be used to control different kinds of robotic platforms and it handles low level functions and routine tasks. It has been released under 'Open Source' licence by the Carnegie Mellon University (CMU) and can be downloaded, free of charge, from the Tekkotsu website. 'Tekkotsu' is a Japanese term meaning iron bones but this is also a term to define the structural framework within a building. This analogy is here to show that the Tekkotsu framework has been designed to be used as a structure for the robotic applications to be built on [Ihse, 2004] [Tekkotsu, 2005].

It is based on top OPEN-R and inherits its capacities by making full use templates. The Tekkotsu framework saves from developing low level functionalities and aids the focus on

programming advanced behaviour instead. As OPEN-R, Tekkotsu is object-oriented and an event-passing architecture. It also makes use of three system processes [Bie, 2004]:

- MainObj handles most of the bulk processing including vision, decision making and tracking the state of the world.
- MotoObj is used to control the positions of joints and to turn the LEDs on and off.
- SoundPlay, as its name suggests, is used to mix and send sound data to the system.

Already implemented in Tekkotsu, are several interesting features: colour segmentation used for object recognition and the low walking style used by the CMU team in 2001 allowing the robot to 'dibble' forward with the ball.

3.5 Choice of software framework

As the control system chosen is quite elaborate, the software framework used had to present object-oriented programming capabilities with the possibility of developing modules. This excludes the utilisation of R-Code based systems. URBI can fit but the lack of documentation and users makes it difficult for a beginner to use.

OPEN-R and Tekkotsu are both based on C++ programming language. The difference for the programmer lies in the work he/she wants to achieve. If the aim is to directly control the joints and have access to low level functions, OPEN-R would be suitable. But in our case, the focus is on the development of a high level and Tekkotsu would save design time by providing inbuilt solutions to control the robot. This way, the design can focus on the implementation of finite state machines. Additionally, the Tekkotsu framework provides tools to easily handle finite state automata which are generalised finite-state machines for modelling hybrid systems.

4 Technical Background

4.1 Introduction

This chapter covers the technical knowledge which the reader will need in order to understand the system. First, it presents the Aperios Operating System running on the Aibo. Then, a deeper overview of Tekkotsu explains the way it works followed by an explanation of decision trees. This chapter finishes with an outline of Finite State Machines and hybrid automata.

4.2 The Aperios Operating System

Aibo is run by the operating system Aperios (formerly known as Apertos). It is used in many of Sony's products, such as the bipede Qrio [Tan, 2004] and the play station. Based on metal-level architecture, this object-oriented and distributed real-time operating system is interesting for embedded systems due to its small size: approximately 100KB. The only constituent of Aperios is the object. The operating system is open to evolve and adapt to its environment to allow objects freedom of movement within a distributed environment. Thus, each object encapsulates the state, the methods to access the state, and a virtual processor used to execute such methods. Therefore, an object can change its semantics and properties at running-time. That is where the object/meta-object separation lies. An object is no more than a container for the information whilst a meta-object contains the definition of its semantics.

An object owns its group of meta-objects giving abstract instructions or meta-operations defining its semantics. A meta-object is also an object; therefore it can also have meta-objects. This defines a meta-hierarchy as illustrated in figure 9. The behaviour of an object is thus defined using a meta-hierarchy of meta-objects. The group of meta-objects used by an object is called its meta-space. Also, if an object cannot continue its execution when it is evolving or changing, it can change its meta-space to a new one or migrate to keep running.

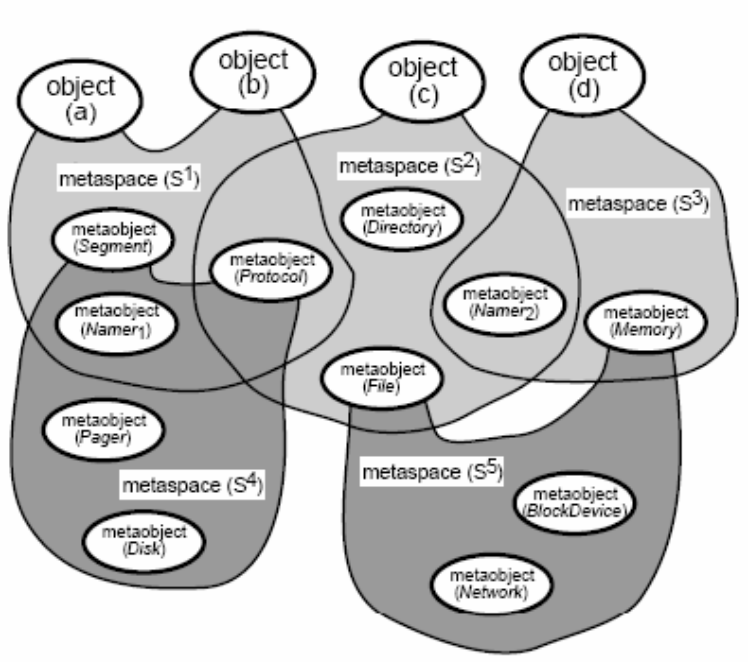


Figure 9 Aperios: Representation of the Object/Meta-Objects hierarchy
[source: Team Chaos (Bie, 2004)]

When an Aperios application is running, it is supported by one of many meta-spaces. Each meta-space contains at least one meta-object, called the reflector. A reflector acts like a gateway: it intercepts incoming requests to the meta-space and forwards them to the appropriate meta-object.

4.3 Overview of Tekkotsu

Tekkotsu is a large framework with many possibilities. The structure of the framework is represented in figure 10. In addition to the three main objects explained in chapter 3.4.5, some structures have to be considered [Ihse, 2004] [Bie, 2004] [Tekkotsu, 2005].

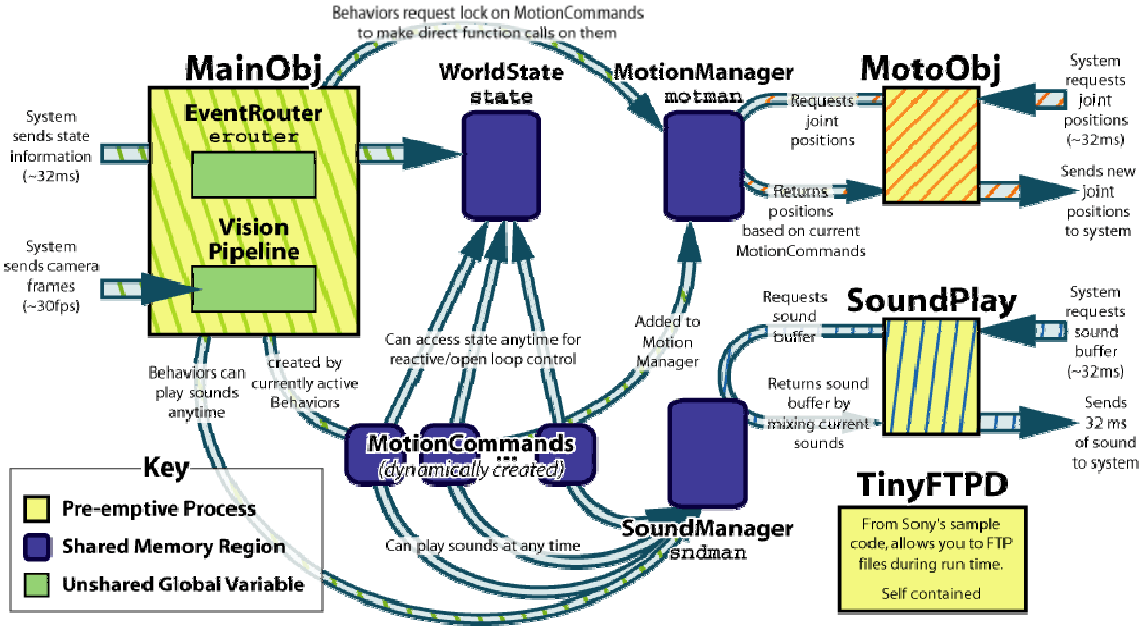


Figure 10 The structure of Tekkotsu [source: Tekkotsu website]

4.3.1 Worldstate

This module is where the state of the dog at a given time is stored. This is a shared memory region holding information about joint positions and torques, buttons and power status, distance reported by the sensors, accelerometer, temperature, LED values, PID settings and ear positions.

4.3.2 MotionManager

This class is used to serialise requests from behaviours to move the joints in the legs, the head or the tail. It is based on a singleton called 'MotMan' which means the motion manager has to insure that the 'Main' and 'Motion' processes have mutually exclusive access to the command and it provides simultaneous execution of several motion primitives concurrently. A motion primitive is a shared memory space based on 'MotionCommand'.

If a behaviour requests moving a joint, a request is sent to the lock of the appropriate motion command from the MotionManager. It sets new values for the joint position and the changes are performed by the effectors and the lock is then released. This module communicates to the MotoObj and MainObj processes described in chapter 3.4.5.

4.3.3 SoundManager

This module is similar to the MotionManager but is dedicated to the sound diffused by the robot. Similar to the previous module, it makes use of a singleton this time named 'SndMan'. Every time a sound has to be played, the playfile method is called in the sound manager and sounds from various sources are mixed and sent to the SoundPlay process described in chapter 3.4.5.

4.3.4 StateNode

This module is a subclass of Behaviorbase (described below) and is used to implement state machine nodes, as well as controllers, which permit the implementation of multilevel state machines.

4.3.5 Transition

Like StateNode, Transition is a subclass of BehaviorBase and this is a base class used for implementing transitions between state nodes and state machines. A transition has to subscribe for events like for instance a timer to be informed when the activation of the transition has to take place. The activation of a transition is the transition from one state to another. Then the destination node becomes activated whilst the source node becomes deactivated.

4.3.6 Event Passing

When a sensor receives stimulus from the surrounding world or the camera detects an object, the system has to be aware of this in order to express the adequate behaviour. But the sensors, nor the vision system, are not aware of which objects to call as a result of the stimulus. Thus, the system posts the information in a message called 'event' by Tekkotsu to the event handler used to manage the distribution of messages to the objects having registered to the listener. A listener is a part of the program accessible from any class and which is constantly waiting for a signal or event from any object in the system. The event handler, a singleton like the motion and sound managers, is called 'EventRouter' and it will call the 'processEvent' method from the object that should be notified.

EventRouter is globally accessible, which means it can be called by any object from anywhere in the code. So any object can register, listen and throw events at any time. Many event generators have been built in Tekkotsu: for instance, VisionEvent is generated by the vision system. The user can define new event generator in classes inheriting the 'EventListener' class.

4.3.7 Vision and object recognition

The system used for object recognition is based on the colour segmentation code from the Carnegie Mellon University. For the Aibo robot, the vision is the primary sensor. The camera records colour images in the YUV perception model with a frame rate of 25 Hz. It then uses colour segmentation to analyse them as it is computationally inexpensive compare to other features.

4.3.8 BehaviorBase

This is a subclass of 'EventListener'. It is the base class for any application the user wants to produce. At the start and the end of an application, the Tekkotsu framework will first call the DoStart and DoStop method from the application based on the corresponding methods from the BehaviorBase.

4.4 Decision Trees

Decision trees are widely used and represent a robust solution to represent any discrete functions on discrete features [Garabadu, 2003]. A decision tree is a hierarchical structure used to represent a behaviour or a hierarchy of behaviours. The tree can be read from the top to the bottom until reaching a leaf. It is used to reach a decision from a situation entered as input to the tree. The decision corresponds to the output value corresponding to the input. The decision is performed by sequential testing. Each internal node in the tree corresponds to a test the system will perform before reaching the decision. The branches under the node correspond to the possible solution resulting from the test. And finally, every leaf will be the value to be returned when reached [Bie, 2004].

Therefore, a leaf corresponds to a simple behaviour and an internal node will correspond to a more complex one, composed of the behaviour below it. In our case, as shown in figure 11, the predator can be hungry or not. Thus, he waits or looks for some food. If he is hungry, and cannot see any prey, he will look for it. Otherwise, if he is close enough, he will eat it, or chase it if too far.

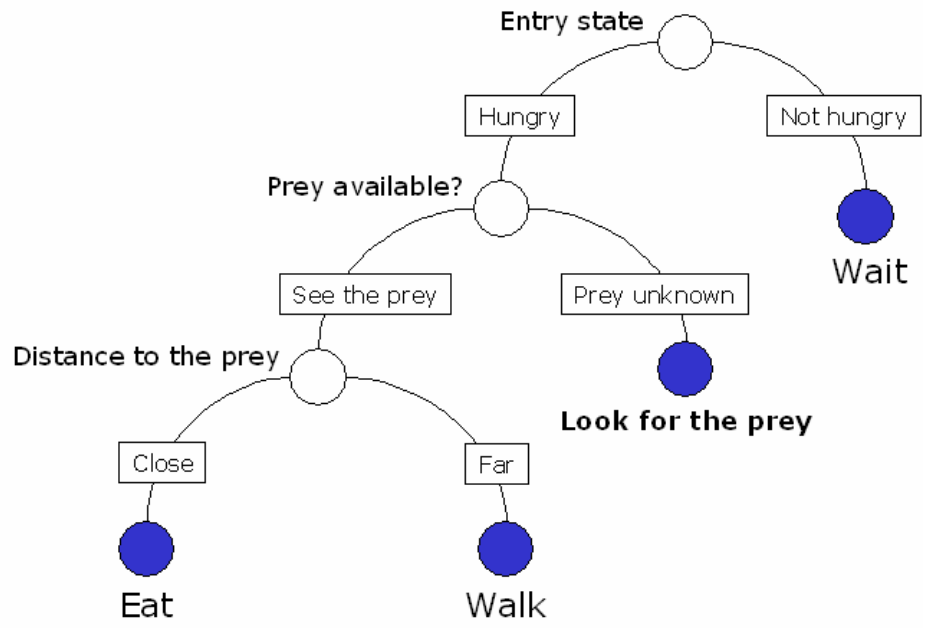


Figure 11 Decision tree for the project

4.5 Finite State Machine

A finite state machine (FSM), also known as finite state automaton, is a model of computation composed of states, transitions and actions (figure 12). It was first defined in the automata theory [FSM, 2005]. The states represent the condition of an object at a given time. Transitions correspond to a change of state for the object and they are described by a condition which triggers the transition.

Actions are the descriptions of activities performed at a given time. An action can be of different types. An entry action will be executed when the system enters the state while the exit action will be triggered when the system leaves the state. An input action can be executed or not depending on the present state and an input condition. Finally, a transition action is performed when the system is on transition [Black, 2005].

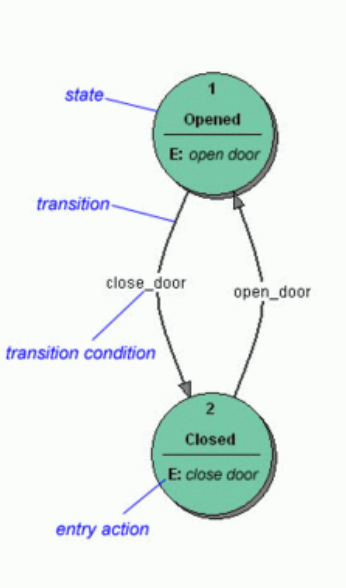


Figure 12 Example of a Finite State Machine (FSM) [source: Wikipedia]

An FSM can be represented using a diagram as shown in figure 12. This example [FSM, 2005] shows an FSM for a system opening and closing a door. The door is originally closed, but if somebody opens it, its state becomes opened. 'Open door' will be the entry action for the state 'opened' while 'close door' is the one for the state 'closed'. 'close_door' and 'open_door' are the transition conditions between the states.

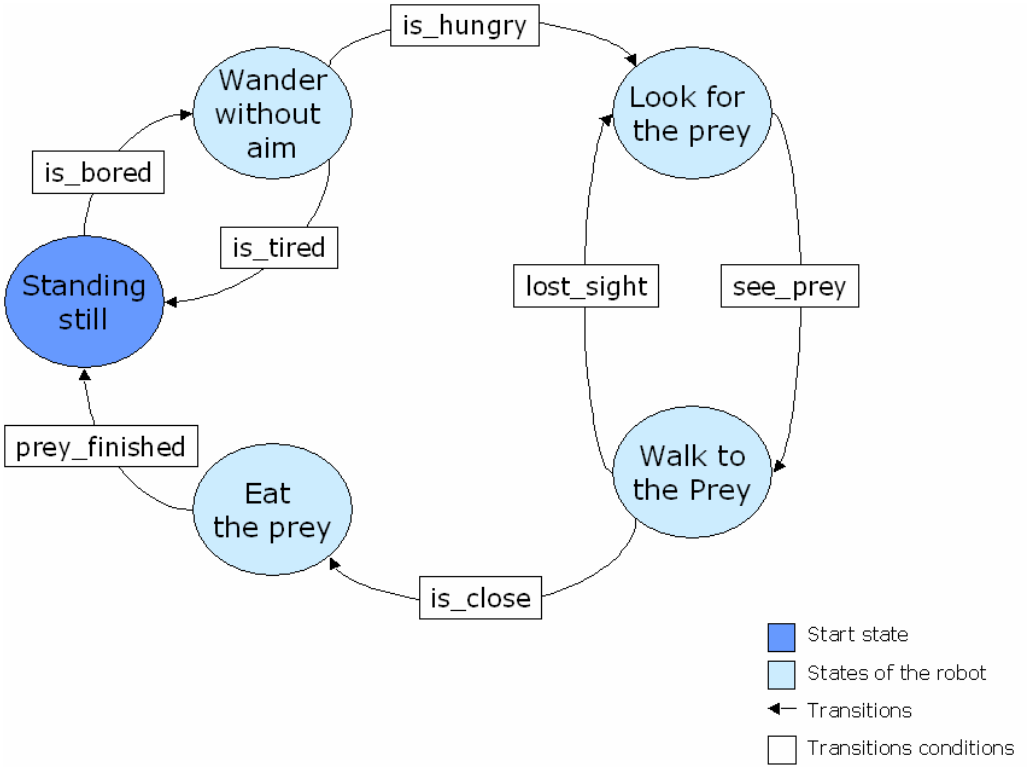


Figure 13 Simplified finite state machine for the project

Figure 13 shows a simplified version of FSM for the project. All the actions and the states of the robot dog are represented with transitions between them. They are many ways to design an FSM. The two main examples are the Mealy Machine, which has the actions associated with the transitions, and the Moore machine which associates actions to states. Also, some FSMs can have multiple start states, transitions conditioned on a null input symbol, or more than one transition for a state [Black, 2005].

4.6 Hybrid Automata

The automata theory is part of computer sciences and focuses on finite state machines using mathematical models to represent them [Automata, 2005]. A hybrid automaton combines discrete transition graphs with continuous dynamical systems [Henzinger, 2000]. It is a model and specification as well as a generalised finite state machine for modelling hybrid systems [Alur, 1993]. A hybrid automaton is an extension of timed automata allowing the use of continuous variables but with more dynamics than can be found in clocks. A hybrid system is constituted of a discrete program within an analogue environment. The discrete transitions between all the states of the system are defined by a change of the program counter, from a finite set of control locations.

They can be written using Tekkotsu with a class inheriting the basic node called *StateNode*. This node is defined as either a state machine controller or a node within a

state machine itself. This way, a node belongs to a hierarchical structure similar in some respect to a decision tree.

5 Conceptual solution

5.1 The Behaviour

As defined in chapter 3.2, a behaviour is an action or a sequence of actions that accomplishes something. To design a behaviour, we have to design an object in the Tekkotsu framework. This object is charged into the robot and can be started using the Tekkotsu user interface called TekkotsuMon; a server running on a computer and linked to the robot by a Wireless Local Area Network (WLAN) connection. Using this, the user can start and stop a behaviour manually. The specifications of the system imply that only one behaviour is supposed to be able to run at a time. The behaviour from this project is called 'Pursuit Behaviour' and deals itself with commanding the different behaviours required to perform the chase. When the behaviour is triggered, the robot starts wandering without a particular purpose and after a certain time will get hungry and start looking for its prey. Once it has located the prey, it will chase it trying to catch it.

5.2 Recognise the prey

The first module needed for this behaviour was the capability for the robot to recognise a prey. In our design, the chosen prey to be used would take the form of the pink ball which is available with the robot. Object recognition is a very important area in robotics; although humans can easily recognise an object based on a known pattern, this remains difficult for a robot as the process for object recognition goes well beyond simple image processing. To simplify this problem, the most common technique used to recognise objects is based on their colour. This implies making the assumption that an object requiring recognition has a unique colour in the environment. This is a utopia in a real environment; however, it can also be achieved in a closed environment and appears to be a viable solution in the Robocup as this is the technique used to recognise the landmarks on the side of the pitch and the goals. The main issue for this technique occurs when dealing with different types of lights. Nuances of colour are easily subject to change depending on the amount of light on an object, and even small changes can have an effect on the colour recognition. This sensitivity exists also in the human brain but the brain compensates and allows the human to recognise colours under different lighting.

We shall now assume that the ball is the only pink object in the environment and that the lights are optimal. Therefore, the robot can assume it has located the ball when it receives a signal notifying a pink object in the environment. The Tekkotsu framework provides a tool for detecting any pink object in the range of the camera. This tool

supposedly only detects the ball but in fact triggers for every pink object, be it the Aibone or a pink carpet.

5.3 Search for the prey

In order to find the ball, the robot will explore the environment looking for it. It will walk at random, hoping to get close to the ball. This is not the optimum way of exploring an area but it has the advantage to be simple and to avoid the robot following a given path or exploring the environment in a predictable way. During this move, the robot moves its head side to side to have a wider view range and therefore a better coverage of the environment. When the vision system detects a pink object, the robot then starts chasing the prey.

The robot moves in the environment using the crawl walking mode defined by the Carnegie Mellon University for the Robocup challenge. The robot is walking on its knees and elbows. This is not the natural way for a dog to walk but this mode is more stable and faster in consideration to the capacities of the Aibo robot. This also provides a better stability for the camera and therefore a better image analysis as well as object recognition. Stability for the camera is a well known issue when dealing with walking robots as seen when designing the Mutant (see chapter 2.2.3).

5.4 Chase the prey

Once the prey is located, the robot has to get closer to it to be able to attempt capture. The principal for the chase is that the camera has to keep centred on the ball and the robot goes where the head points. If the ball is static, the camera will easily keep the prey in the centre of the vision area, but dealing with a moving prey is more complicated. The robot is constantly adjusting its head position to have the prey as close as possible to the centre of the image. If the head makes an angle with the body superior to a given threshold, the robot will stop moving forward but will adapt its trajectory to go towards the prey. At the same time, the head will turn the opposite direction to keep the prey at the centre of the image. This way, the robot gets closer to the prey while looking directly at it.

An important aspect of chasing the prey is to keep the prey in sight. If the robot loses the track of the prey, it has to revert to searching for it. The robot may lose sight of the prey for many reasons; if the lighting changes as seen in chapter 5.2, the colour recognition might be inefficient. Also, if the prey is too fast for the robot, it can move to a

position out of the vision range of the robot while the robot could not move quickly enough to follow the trajectory.

While the robot is getting close to the prey, it has to decide when it launches the attack to try to catch it. In this project, when the robot receives information indicating the prey is close enough, it will stop running and start making noises resembling a dog feasting. Deciding when the prey has been caught appeared to be an issue as it is complicated to have the robot stop right on the prey and not walk past it or stop too early. The sensors for the distance detection appeared to be unreliable on occasions thus the results are not always very accurate. The shape of the ball was not helping as the curved surface of the ball might reflect the rays from the sensors in different directions making it less precise. Also, as the head is constantly moving during the chase and the sensors are located in the nose of the robot, the sensors readings are less accurate.

5.5 Completing the task

Once the robot has "caught" the prey, it simulates eating it. By this point, it has accomplished its task in the particular case of the experiment. This means the robot has stopped close enough to the prey and has emitted some noises indicating it is feasting. At this point, the system goes back to the start state and the whole behaviour is run once again.

6 System Design

6.1 Introduction

This chapter goes through the development of the modules. This represents the work achieved to program a behaviour based on the conceptual solution. Only the classes which have been designed for the behaviour are described here. Some of them are heavily inspired from the behaviour of demonstration available with the Tekkotsu framework, but they have been adapted for the needs of this project.

6.2 PANode

The PANode class is a subclass of the Tekkotsu StateNode class, itself a subclass of BehaviorBase. When the 'Pursuit Ability' behaviour starts, the PANode object is created and it creates all the objects needed: PAExplore, PAWalk, PAWalkTo and SmallSequenceNode. It also loads the sounds in the system for future use. When the behaviour stops, the class deletes all the objects it has created and releases the sounds.

The different objects created are:

- n_Start from PAWalk. The robot stands still.
- n_Wander from PAExplore. The robot wanders around.
- n_Turn from PAWalk. The robot does a u-turn.
- n_GoTo from PAWalkTo. The robot goes towards a given prey.
- g_Search from GroupNode. The robot wanders around moving its head trying to see the prey. It is composed of n_Explore from PAExplore and n_Head from SmallMotionSequence.
-

Each of these objects is considered as a node. The system uses a system of transitions to change behaviour. At each transition, the robotic dog emits a noise to signal the change. The sounds are loaded into the system at the start of this object and are as follow:

- howl.wav is the scream of a wolf. It means the behaviour is starting.
- sniff.wav is the noise made by a dog when it is sniffing around. It means the dog starts looking for the prey.
- barkreal.wav is the real bark of a dog. It means the dog has seen the prey and starts chasing it.
- whimper.wav is the noise of a sad dog. It means the dog has lost sight of the prey.
- growl.wav is the noise of an angry dog. It means the dog has caught the prey

- barkmed.wav is a happy bark of a dog. It means the dog has finish eating and starts wandering around.

6.3 PAWalk

This class is heavily based on the class WalkNode from Tekkotsu. The robot is given a direction, an angle and a velocity and move according to this information. Using this class, the robot can move in any given direction as long as it does not encounter an obstacle. If the direction is null, it is possible to have the robot doing U-turn.

6.4 PAWalkTo

This class is heavily based on the class WalkToNode from Tekkotsu. The robot is given an object to follow. It will follow it as long as it does not get too close to it, otherwise it would stop. If it loses sight of the target, it will send a signal to the system.

6.5 PAExplore

This class is heavily based on the class ExploreMachine from the demonstration behaviour of the Tekkotsu. The robot wanders around and change direction when it encounters an obstacle. This is used to have the robot to instruct the robot to explore the environment.

6.6 *Finite State Machine*

The system is based on the Finite State Machine approach. Figure 14 shows the structure of the FSM.

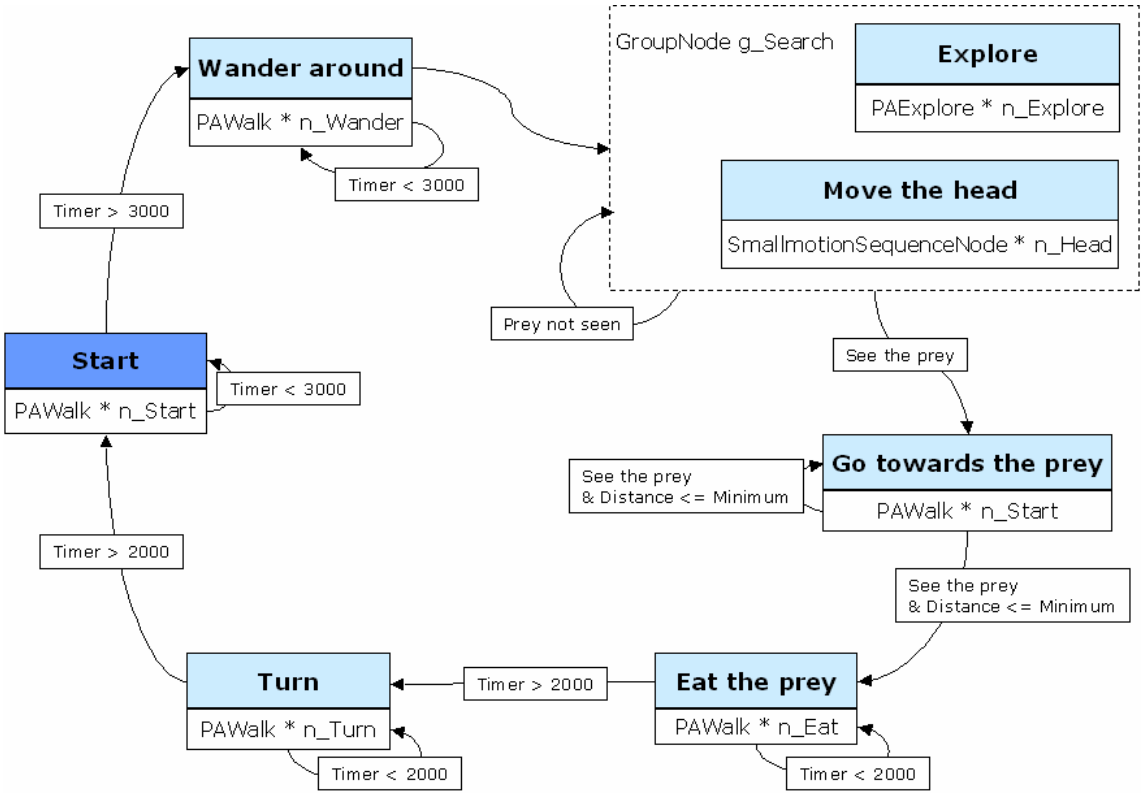


Figure 14 Finite State Machine for the project

We can notice the class SmallMotionSequenceNode for the state 'Move the head'. This class has not been described before as it is an original class from the Tekkotsu framework. It is used to load motions defining series of movement for the robot.

7 Experimental design and results

The behaviour designed for this project was to find a prey and chase it until reaching it. The purpose of this experiment is to assess the capabilities of the robot and to use the results as a start for further development. We have to bear in mind that the behaviour is just a basis for future development and was used to assess the capabilities of the robot rather than to display a real intelligence.

The experiment took place in a closed area as shown in figure 15. First experiments where performed in a square area but the robot was sometimes stuck in a corner ending the experiment. Thus, some areas located in the angles have been removed. They correspond to the green area on the figure 15. Within this environment, the robot had no difficulties to behave and was never blocked. The experiment could be undertaken efficiently.

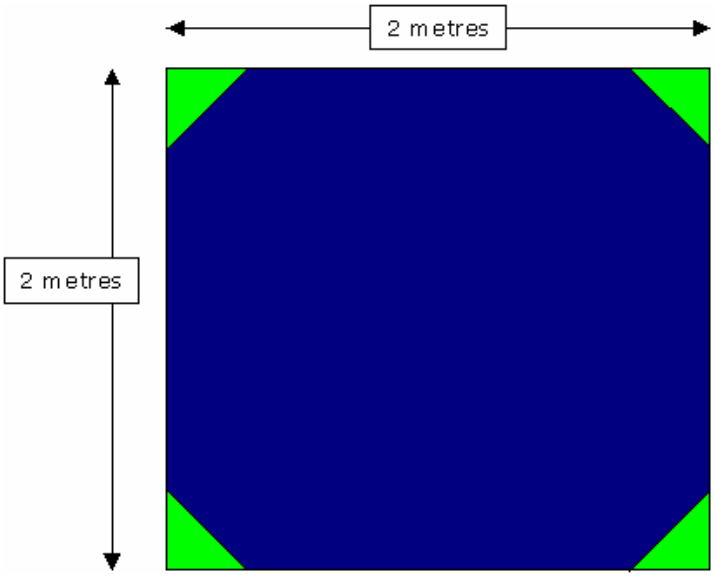


Figure 15 Experiment area

The floor in the area was dark blue and the walls white. The first series of tests were performed in another room with a purple floor and it has soon been noticed that the robot was mistaking the floor for the ball and was triggering an event as if it was seeing something pink (figure 16).

Figure 16 Pink ball on the purple floor and on the blue floor

The robot and the pink ball have been placed at random in the area. The behaviour has been started using the Tekkotsu interface as shown in figure 17. A connection by Telnet on port 59000 allows to have a contact with the robot and to receive information regarding its condition. It displays information about the hardware of Aibo and it also displays messages sent by the system to acknowledge some performances. If the robot sees the ball, a message will be displayed to the user in the Telnet window. If something goes wrong, an error message will be displayed. This helps during the development of the behaviour.

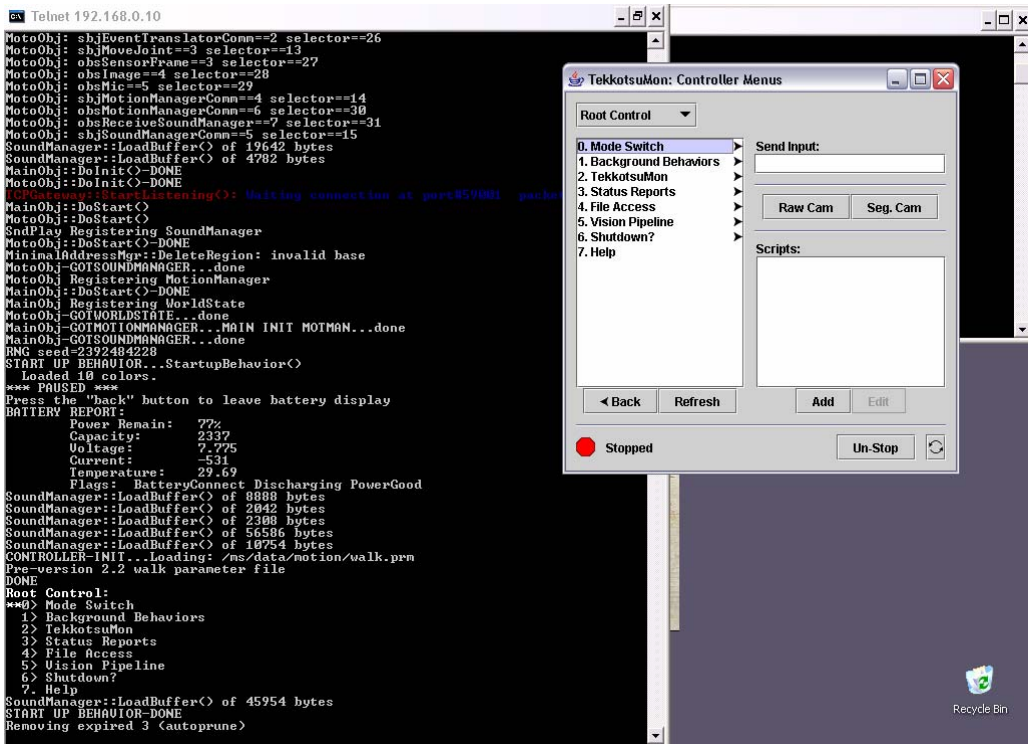


Figure 17 Telnet connection to the Aibo robot and the Tekkotsu WLAN user interface

The Aibo was connected to a laptop using a wireless connection on an ad hoc mode, i.e. connected one to one without any other devices between them. The connection on the laptop was performed using a PCMCIA WIFI card and this has been subject to many problems. The card did not communicate correctly with the Aibo robot. Often, the card had to be disabled and re-enabled to keep on working, especially after a crash from the Aibo system. This problem appeared on two different laptop using two different cards. On figure 18, we can see the material used for the experiment. We can notice on the screen a raw image of what the Aibo can see, and also the PCMCIA WIFI card on the left of the laptop.

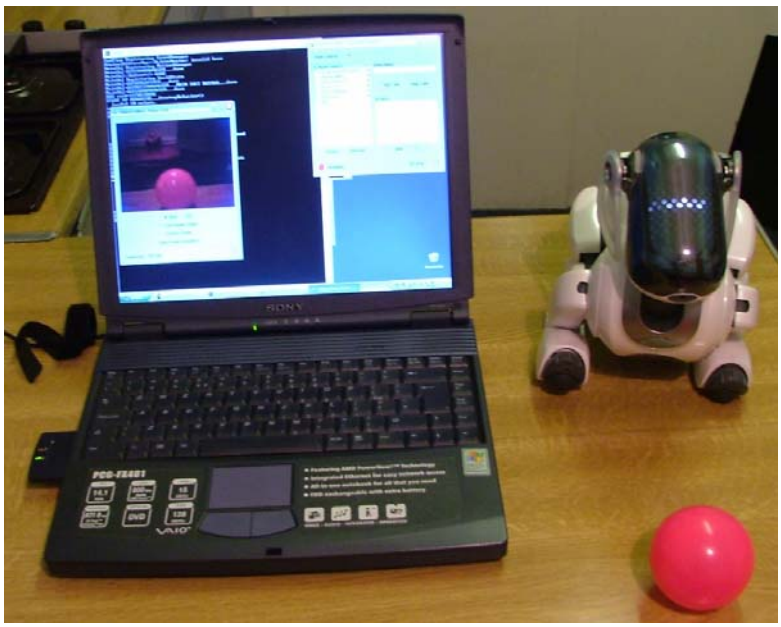


Figure 18 Aibo and the laptop used during the experiment

During the experiment, the robot has been tested under different lights, with the ball on different places, moving or immobile. Results are mitigating. When the robot is placed under too much light, like on a sunny day, it reacts as when it is used in a relatively dark room. The image recognition system analyse the image and return the region with the biggest area matching the colour needed, in our case the pink. This area returned is called a blob. When the light is too strong or when it is dark, the colours appear differently and the system cannot find any blob. Therefore, it does not recognise the ball. Under optimal conditions, the robot does not always find the prey, even when passing just next to it. We can notice that the vision is quite narrow for the robot which needs to move the head consequently to look at the environment.

Figure 19 The Aibo robot running to the ball in the experiment area

During the tests, the Aibo showed a poor capacity to recognise the pink ball. Stability of the camera seemed to be deficient as the robot was noticing the ball less than half of the times it appeared on its vision range while the robot was moving. We could assess that the ball was on the vision range as we could see it on the display in the Tekkotsu interface on the computer. Thus, we can assume that the image recognition was not optimal during the tests. The system, developed by the Carnegie Mellon University, appears to be reliable as long as the robot is standing still but when it is moving, it does always not recognise the pink ball.

To perform some deeper tests, we used the demonstration behaviour called StareAtBall. The robot looks at the pink ball and moves the head to follow the ball if the ball starts moving. During this test, the robot does not move its legs. Using these conditions, the robot was able to recognise the ball easily, even under strong lighting or in a relative darkness.

8 Conclusions

During the test, the robot showed good capabilities regarding the mobility. It can move quickly and has a good reaction time. The camera has a good rendering and shows great possibilities for implementing a concrete vision system. With the use of the other sensors from the robot, especially the distance sensors, it could be possible to design a system able to be independent in an open environment.

However, some tests showed the limitations when the robot was moving in an area not perfectly flat. When confronted to a step of one centimetre, the robot does not notice it and shows incapacity to pass it. Also, during the first series of tests, the system could be stuck in a corner and needed human intervention to be able to keep the experiment running. These limitations constrict the experiment to laboratory environment where the experimental area can be designed adequately. The ability of passing a step could be subject to further work.

The robot showed capabilities to follow the prey but with some limitations. When the prey is static, the robot had no difficulties to go straight to it once it has been localised. But when it comes to deal with a moving prey, the robot shows some difficulties to cope. If the prey is too quick and goes outside the vision area, the robot will lose track of it and will not be able to find it. As long as the prey remains in the middle of the robot's vision range, everything is fine, but if it goes quickly to the side and reach the border of the range, the robot can barely cope with the movement. The vision system seems to be adequat to rendre image of a quickly moving prey and the joints in the head appear to be good enough to follow it but the image treatment does not appear strong enough. The system used for image recognition is quite fragile and would need further work. For every team taking part in the Robocup, the vision treatement is one of the major research and many different solutions have been designed, each with good and bad sides.

The robot also showed good features when exploring the environment, being quick and efficient regarding the reaction to stimulus. The system of sensors used to detect the distance to an obstacle showed good feature with large obstacle like a wall. It did not detect small obstacles like a step or anything located under its chin. This is due of the location of the distance sensor : the nose of the robot. Because of this feature, the robot does not detect an obstacle being close to the paws. Further work could be done on this feature by having the robot consider information coming from pressure sensors locateed in the paws and in the effectors to assess the presence of objects or obstacles under the chin.

The design used to implement the system showed good results. The robot had a quick response time and was performing the right action at the right time according to the requirements. Therefore, it appeared that finite state automata for hybrid systems were good way of design.

The Aibo robot seems to be a good support for testing theories about animal behaviours. Since Sony has open the access to programming the robot using the Open-r environment, the Aibo robot shows great potential for future use in several kinds of researchs like vision, object recognition, moving, or independent localisation. Furthermore, Tekkotsu provides good tools for handling high level features and leaves to the user the ability to improve them.

Thus, the Aibo did not show good ability for the pursuit of a moving prey, but this could be developed and improved making the robot a good subject for further work.

9 References

[4Legged, 2005] *Sony four legged robot league website*
Link: http://www.tzi.de/4legged/bin/view/Website/WebHome

[AIBO, 2005] *Sony AIBO Europe FAQs*
Link: http://www.eu.aibo.com/1_3_1_faq1.asp#q17

[Alur, 1993] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho, *Hybrid Systems I*, Lecture Notes in Computer Science 736, Springer-Verlag, 1993, pp. 209-229.
Link: http://www-cad.eecs.berkeley.edu/~tah/Publications/hybrid_automata.html

[Arkin, 1998], R. C. Arkin, *Behavior-Based Robotics*. MIT Press, 1998.

[Automata, 2005] Automata on Wikipedia
Link: http://en.wikipedia.org/wiki/Automata_theory

[Baillie, 2005] Jean Christophe Baillie, *Universal Robotic Body Interface official website*
Link: http://www.urbiforge.com/eng/index.html

[Bie, 2004] Stefan Bie, Johan Persson, *Behavior-based Control of the ERS-7 AIBO Robot.* Faculty of Science, Lund University, Sweden
Link: http://ai.cs.lth.se/xj/StefanBie/report.pdf

[Black, 2005] Paul E. Black, "finite state machine", from Dictionary of Algorithms and Data Structures, Paul E. Black, ed., NIST.
Link: http://www.nist.gov/dads/HTML/finiteStateMachine.html

[Breazeal, 2000] Cynthia L. Breazeal, *Sociable Machines: Expressive Social Exchange Between Humans and Robots* PHD Thesis, chapter 9. The MIT press, Cambridge
Link: http://groups.csail.mit.edu/lbr/mars/deliverables/deliverables9900.html

[Brooks, 1986] Brooks, R. A. "A Robust Layered Control System for a Mobile Robot", IEEE Journal of Robotics and Automation, Vol. 2, No. 1, March 1986.
Link: http://people.csail.mit.edu/brooks/papers/AIM-864.pdf

[Brooks, 1987] Brooks, Rodney Allen, *"Planning is Just a Way of Avoiding Figuring out What to Do Next"* inCambrian intelligence : the early history of the new AI_, Cambridge, Mass.: MIT Press, 1999. Chapter 6, , pp. 103-110.
Link: http://www.ece.osu.edu/~fasiha/Brooks_Planning.html

[Brooks, 1991] Brooks, R. A. *Intelligence without Representation. Artificial Intelligence* 47(1-3): 139-159.
Link : http://people.csail.mit.edu/brooks/papers/representation.pdf

[Bruemmer, 2005] Bruemmer Davir, *Behavior-based robotics.* Idaho National Laboratory

Link: http://www.inl.gov/adaptiverobotics/behaviorbasedrobotics/index.shtml

[Butoi, 2005],

Link: http://cs.brynmawr.edu/Theses/Butoi.pdf

[Capek, 1920] Capek, Karel *Biography*. Biography.ms

Link: http://karel-capek.biography.ms/

[Fred, 2004] Fred, *Un nouveau kit de développement*. vieartificielle.com

Link: http://www.vieartificielle.com/nouvelle/?id_nouvelle=637

[Frederic, 2004] Frédéric, Mutant, *L'ancêtre de l'AIBO*. vieartificielle.com

Link: http://www.vieartificielle.com/article/?id=199

[FSM, 2005] Finite State Machine on Wikipedia.
Link: http://en.wikipedia.org/wiki/Finite_state_machine

[Garabadu, 2003], Brijesh Garabadu, *Learning decision trees*. Machine learning lectures, School of computin, University of Utah.
Link: http://www.cs.utah.edu/classes/cs5350/slides/d-trees4.pdf

[Germanteam, 2005] *German team official website*
Link: http://www.germanteam.org/

[Globalspec, 2005] *The engering search engine*
Link: http://sensors-transducers.globalspec.com/LearnMore/Sensors_Transducers_Detectors/Acceleration_Vibration_Sensing/Vibration_Sensors

[Henzinger, 2000] Thomas A. Henzinger, *The theory of hybrid automata*. *Proceedings of the 11th Annual Symposium on Logic in Computer Science* (LICS), IEEE Computer Society Press, 1996, pp. 278-292.
Link: http://www-cad.eecs.berkeley.edu/~tah/Publications/the_theory_of_hybrid_automata.html

[Ihse, 2004] Christina Ihse Bursie *Dog-like behavior selection for an AIBO robot dog*. Royal institute of technology, Stockholm, Sweden
Link:
http://www.nada.kth.se/utbildning/grukth/exjobb/rapportlistor/2004/sammanf04/ihse_bursie_christina.html

[Ina, 2004]Goihman Ina, *Aura, autonomous robot architecture.* Haifa University
Link: http://math.haifa.ac.il/robotics/Projects/AuRA_Ina.pdf

[Jennings, 1995] Nick Jennings, *Intelligent agent, theory and practice.* Knowledge Engineering *Review* Volume 10 No 2, June 1995.
Link:
http://www.csc.liv.ac.uk/~mjw/pubs/ker95/subsubsectionstar3_3_2_3.html#SECTION00032300000000000000

[Jerome, 2004] Jerome, *L'histoire AIBO.* vieartificielle.com
Link: http://www.vieartificielle.com/article/?id=197

[Juengel, 2004] Matthias Juengel, Joscha Bach, *Using a flexible grid for image recognition.* Humboldt University of Berlin
Link: http://www.informatik.hu-berlin.de/~juengel/papers/

[Kaplan, 2001] Kaplan, Frédéric, *Un robot peut-il être notre ami ?* Orlarey, Y., editor, L'Art, la pensée, les émotions, pages 99-106, 2001. Grame.
Link : http://www.csl.sony.fr/downloads/ papers/2001/kaplan-rmpd2001.pdf

[Lamb, 1987] Lamb, J. R., *Computer simulation of biological systems. Molecular and Cellular Biochemistry,* 73:91-98

[Logan, 2004] Brian Logan, Deliberative *architecture.*University of Nottingham
Link: http://www.cs.nott.ac.uk/~bsl/G53DIA/

[Lötzsch, 2004] Martin Lötzsch, *XABSL, a behavior engineering system for autonomous agents.* Thesis report, Humboldt University of Berlin
Link: http://www.martinloetzsch.de/papers/diploma-thesis.pdf

[Maes, 1990] Maes, Pattie, *How to do the right thing.* Connection science journal, special issue on hybrid Systems, v1
Link: http://agents.www.media.mit.edu/groups/agents/publications/Pattie/consci/

[Maes, 1995] Maes, Pattie, *Artificial Life Meets Entertainment: Life like Autonomous Agents*, Communications of the ACM, 38, 11, p.108

[Mataric, 2004] Maja J. Mataric, *Behavior based control, a brief primer*
Link: http://www-robotics.usc.edu/~maja/bbs.html

[Minsky, 1986] Marvin Minsky, *The society of mind*
Link: http://web.media.mit.edu/~minsky/

[Moore, 2005] *Moore Machine on Wikipedia*
Link: http://en.wikipedia.org/wiki/Moore_machine

[Murphy, 2004] Dr. Robin R. Murphy, *Introduction to AI robotics: Reactive Paradigm*. University of South Florida
Link: http://www.csee.usf.edu/~murphy/book/slides/chapter4v2.ppt

[Mystic, 2004] Mystic Fortress, *Tamagotchi history, in an egg shell*. Tamagotchi Connection
Link: http://www.mimitchi.com/tamaplus/tama_history.shtml

[Newcastle, 2005] *Newcastle Robotics Laboratory web site.* University of Newcastle, Australia
Link: http://robots.newcastle.edu.au/robocup.html

[NUbot, 2004]*The 2004 NUbots team report*. University of Newcastle, Australia
Link: http://robots.newcastle.edu.au/publications/NUbotFinalReport2004.pdf

[Nwana, 1996] Hyacinth S. Nwana, *Software agents, an ovreview.* Knowledge Engineering Review, Vol. 11, No 3, pp.1-40, Sept 1996
Link: http://agents.umbc.edu/introduction/ao/

[Onrobo, 2003] *Onrobo product review and info*. Onrobo.com
Link: http://www.onrobo.com/reviews/AIBO/ERS_7_Series/on00ers700rosny/

[Open-r, 2005] *AIBO SDE official website*
Link: http://openr.aibo.com/

[Payen, 2005] Payen, Guillaume, *Technologie des robots autonomes.* CNRS Paris-Sud official website
Link : http://www.lri.fr/~felkin/agents/GuillaumePayen/sma-payen.html

[Phoebe, 1998] Phoebe, *Furby autopsy*. phoebe.com
Link: http://www.phobe.com/furby/

[Pylyshyn , 1988] Fodor, J.A., and Z.W. Pylyshyn. 1988. *Connectionism and Cognitive Architecture: A Critical Analysis*, Cognition. 28:3-71
Link: http://ruccs.rutgers.edu/ftp/pub/papers/jaf.pdf

[R-Code, 2004] *R-Code supplies website*
Link : http://dogsbodynet.com/rcode_supplies.html

[Rico, 2004] Francisco M. Rico, Rafaela G. Careaga, Jose M. C. Plaza, Vicente M. Olivera, *Programming model based on concurrent objects for the AIBO robot.* Universidad Rey Juan Carlos, Madrid, Spain
Link : http://gsyc.escet.urjc.es/robotica/publicaciones/concurrencia04.pdf

[Roberty, 2005] Roberty, Jérôme, *Le site francophone des fans d'AIBO*.
Link : http://www.premiumwanadoo.com/aibo-fr.com/menu.htm

[Robocup, 2005] *Robocup official website*
Link: http://www.robocup.org/overview/21.html

[Robocup2005, 2005] *Robocup 2005 Osaka official website*
Link: http://www.robocup2005.org/about/what.aspx

[RobocupJunior, 2005] *Robocup Junior official website*
Link: http://satchmo.cs.columbia.edu/rcj/

[RobocupRescue, 2005] *Robocup Rescue official website*
Link: http://kaspar.informatik.uni-freiburg.de/%7Ercr2005/

[RPI, 2005] *Rennsselaer Polytechnic Institute official website*
Link: http://www.rpi.edu/about/index.html

[Sony, 2005] *Sony official website*

Link: http://www.sony.net/Products/aibo/

[Sonystile, 2005] *SonyStyle official website*
Link: http://www.sonystyle.com/is-bin/INTERSHOP.enfinity/eCS/Store/en/-/USD/SY_DisplayProductInformation-Start?ProductSKU=ERS7M2/WKIT1

[Sorayama, 2005] Hajime Sorayama, *official website*
Link: http://www.sorayama.net/

[Sycara, 1998] Katia p. Sycara, *Multi agent systems.* AI magazine

Link: http://www.findarticles.com/p/articles/mi_m2483/is_n2_v19/ai_20914130

[Tan, 2004] Tan Hellen, *Quest for curious robot*. Computer Times 23rd June 2004

Link: http://computertimes.asia1.com.sg/ctkids/story/0,5104,2533,00.html

[Technostuff, 2005] *Techno-Stuff Robotics*

Link: http://www.techno-stuff.com/Accel.htm

[Tekkotsu, 2005] Tekkotsu official website

Link: http://www.cs.cmu.edu/~tekkotsu/index.html

[Tellez, 2004] Ricardo A Tellez. *R-Code SDK Tutorial*

Link : http://www.ouroboros.org/rcode_tutorial_1v2.pdf

[Top dog, 2005] Team "Top dog", Jack Liu, Bryan Knight, David Elrod, *AIBO cooperative soccer project.* Rennsselaer Polytechnic Institute
Link: http://www.cat.rpi.edu/~wen/aibo/

[Uncanny, 2005] *The Uncanny Valley study on Wikipedia*

Link: http://en.wikipedia.org/wiki/Uncanny_Valley

[Webb, 2001] Webb, Barbara, *Can robots make good models of biological behaviour?*
Link: http://www.bbsonline.org/Preprints/Webb/

[Wikipedia, 2005] *Behaviour on Wikipedia*
Link: http://en.wikipedia.org/wiki/Behavior

[Yamada, 2004] Yamada Seiji, Yamaguchi Tomohiro, *Training Aibo like a dog.* National Institute of Informatics, Tokyo, Japan

Link: http://research.nii.ac.jp/~seiji/publication/Conference/2004/ROMAN-2004-yamada.pdf

[Yart, 2005] *Yet Another R-Code Tool official website*

Link: http://www.aibohack.com/rcode/yart.htm

10 Credits

Figure 1 : http://www.firebox.com/?dir=firebox&action=product&pid=859

Figure 2 : pinkangel15.tripod.com/

Figure 3 : www.vieartificielle.com/article/?id=197

Figure 4 : http://www.premiumwanadoo.com/aibo-fr.com/ERS220.htm

Figure 5 : http://www.converj.com/blogs/converjed/archives/2004_10.html

Figure 6 : http://www.sony.net/Products/aibo/

Figure 7 : http://www.sony.net/Products/aibo/

Figure 8 : http://www.tzi.de/4legged/pub/Website/Downloads/Rules2005.pdf

Figure 9 : http://ai.cs.lth.se/xj/StefanBie/report.pdf

Figure 10 : http://www.cs.cmu.edu/~tekkotsu/

Figure 11 : Julien Roux

Figure 12 : http://en.wikipedia.org/wiki/Finite_state_machine

Figure 13 : Julien Roux

Figure 14 : Julien Roux

Figure 15 : Julien Roux

Figure 16 : Julien Roux

Figure 17 : Julien Roux

Figure 18 : Julien Roux

Figure 19 : Julien Roux

Figure 20 : Julien Roux

Figure 21 : Julien Roux

Figure 22 : Julien Roux

Figure 23 : Julien Roux

11 Appendix

11.1 PANode.h

```
// Julien ROUX
// Heriot Watt University
// File: PANode.h
// September 2005

#ifndef INCLUDED_PANODE_H_
#define INCLUDED_PANODE_H_

#include "Behaviors/StateNode.h"
#include "Shared/ProjectInterface.h"

//Class to simulate the Pursuit Ability (PA)
class PANode : public StateNode
{
  public:
    //Constructor
    PANode()
    : StateNode("PANode","PANode"),
      start(NULL)
    {}

    //Destructor
    ~PANode()
    {
        if (issetup)
            teardown();
    }

      virtual void setup();

      virtual void DoStart();

      virtual void teardown();

protected:
      StateNode* start;

private:
      PANode(const PANode&);
      PANode operator=(const PANode&);
};
#endif
```

11.2 PANode.cc

```
// Julien ROUX
// Heriot Watt University
// File: PANode.cc
// 06 September 2005

#include "PANode.h"
#include "PAExplore.h"
#include "PAWalkTo.h"

#include "Behaviors/Transition.h"
#include "PAWalk.h"
#include "Behaviors/Transitions/TimeOutTrans.h"
#include "Behaviors/Transitions/VisualTargetTrans.h"
#include "Behaviors/Nodes/OutputNode.h"
#include "Behaviors/Nodes/MotionSequenceNode.h"
#include "Behaviors/Nodes/GroupNode.h"
#include "Sound/SoundManager.h"

void PANode::setup()
{
      StateNode::setup();

    //**********  Variables **********//
    const float m_TurnSpeed = 1;
    const float m_Null = 0;
    const float m_Low = 50;
    const float m_Mid = 80;

    //************  Nodes ************//
    //Beginning of the behaviour. The robot don't move
    PAWalk * n_Start = new PAWalk(m_Null, m_Null, m_Null);
    n_Start->setName(getName()+"::n_Start");
    addNode(n_Start);

    //Wander around without purpose
    PAExplore * n_Wander = new PAExplore (getName()+"n_Wander", m_Low);
    n_Wander->setName(getName()+"::n_Wander");
    addNode(n_Wander);

    //Turn around
    PAWalk * n_Turn = new PAWalk (m_Null, m_Null, m_TurnSpeed);
    n_Turn->setName(getName()+"::n_Turn");
    addNode(n_Turn);

    //Look for the prey
    GroupNode * g_Search = new GroupNode (getName()+"::g_Search");
    addNode(g_Search);
    {
        PAExplore * n_Explore = new PAExplore(g_Search-
>getName()+"::n_Explore", m_Mid);
        g_Search->addNode(n_Explore);
        SmallMotionSequenceNode * n_Head = new
SmallMotionSequenceNode(g_Search->getName()+"::n_Head",
"pan_head.mot",true);
        g_Search->addNode(n_Head);
    }
```

```
    //Go to the prey
    PAWalkTo * n_GoTo = new PAWalkTo(ProjectInterface::visPinkBallSID);
    n_GoTo->setName(+"::n_GoTo");
    addNode(n_GoTo);

    //Beginning of the behaviour. The robot don't move
    PAWalk * n_Eat = new PAWalk(m_Null, m_Null, m_Null);
    n_Eat->setName(getName()+"::n_Eat");
    addNode(n_Eat);

      //**********  Transistions **********//
    Transition * t_Trans=NULL;

    //after 3 seconds, the robot starts
    n_Start->addTransition(t_Trans = new TimeOutTrans(n_Wander, 3000));
    t_Trans->setSound("howl.wav");

    //after 10 seconds, the dog is hungry
    n_Wander->addTransition(t_Trans = new TimeOutTrans(g_Search, 10000));
    t_Trans->setSound("sniff.wav");

    //the dog sees the prey and starts chasing it
    g_Search->addTransition(t_Trans = new VisualTargetTrans(n_GoTo,
ProjectInterface::visPinkBallSID));
    t_Trans->setSound("barkreal.wav");

    //the dog lose sight of the prey
    n_GoTo->addTransition(t_Trans = n_GoTo->newDefaultLostTrans(g_Search));
    t_Trans->setSound("whinmper.wav");

    //Once the dog reached the prey, it eats it
    n_GoTo->addTransition(t_Trans = n_GoTo->newDefaultCloseTrans(n_Eat));
    t_Trans->setSound("growl.wav");

    //after 10 seconds, the dog is hungry
    n_Eat->addTransition(t_Trans = new TimeOutTrans(n_Turn, 2000));
    t_Trans->setSound("barkmed.wav");

      //once you've turned around, wander for a while
      n_Turn->addTransition(t_Trans=new TimeOutTrans(n_Wander,(unsigned
int)(M_PI/m_TurnSpeed*1000))); //turn 180 degrees (aka PI radians)
      t_Trans->setSound("barkmed.wav");

      //preload the sounds so we don't pause on tranisitions
      sndman->LoadFile("howl.wav");
      sndman->LoadFile("sniff.wav");
      sndman->LoadFile("barkreal.wav");
      sndman->LoadFile("whimper.wav");
      sndman->LoadFile("growl.wav");
      sndman->LoadFile("barkmed.wav");

      //starts out exploring
      start=n_Start;
}

void PANode::DoStart() {
      StateNode::DoStart();
      start->DoStart();
}
```

```
void PANode::teardown() {
      //release the sounds
      sndman->ReleaseFile("howl.wav");
      sndman->ReleaseFile("sniff.wav");
      sndman->ReleaseFile("barkreal.wav");
      sndman->ReleaseFile("whimper.wav");
      sndman->ReleaseFile("growl.wav");
      sndman->ReleaseFile("barkmed.wav");
      StateNode::teardown();
}
```

11.3 PAExplore.h

```
// Julien ROUX
// Heriot Watt University
// File: PAExplore.h
// Base on ExploreMachine.h from Tekkotsu
// 06 September 2005

#ifndef INCLUDED_PAEXPLROE_H_
#define INCLUDED_PAEXPLORE_H_

#include "Behaviors/StateNode.h"
#include "Motion/MotionManager.h"

class PAExplore : public StateNode
{
  public:
      //!constructor
      PAExplore(const std::string& nm)
            : StateNode("PAExplore",nm),
          start(NULL),
          turn(NULL),
          walkid(MotionManager::invalid_MC_ID)
      {}

      //Constructor with a speed
      PAExplore(const std::string& nm, float speed)
            : StateNode("PAExplore",nm),
          start(NULL),
          turn(NULL),
          m_Speed(0),
          walkid(MotionManager::invalid_MC_ID)
      { m_Speed = speed;}

      //!destructor
      ~PAExplore()
    {
            if(issetup)
                  teardown();
      }

      virtual void setup();
      virtual void DoStart();
      virtual void DoStop();
      virtual void teardown();

      virtual void processEvent(const EventBase&);
```

```
protected:
      StateNode * start;
      class PAWalk * turn;
      MotionManager::MC_ID walkid;
      float m_Speed;

private:
      PAExplore(const PAExplore&);
      PAExplore operator=(const PAExplore&);
};
#endif
```

11.4 PAExplore.cc

```
// Julien ROUX
// Heriot Watt University
// File: PAExplore.cc
// 06 September 2005

#include "PAWalk.h"
#include "Behaviors/Transitions/SmoothCompareTrans.h"
#include "Behaviors/Transitions/TimeOutTrans.h"
#include "Shared/ERS7Info.h"
#include "Wireless/Socket.h"
#include "Shared/WorldState.h"

#include "PAExplore.h"

void PAExplore::setup() {

    //**********  Variables **********//
      unsigned int IRDistOffset;
      SharedObject<WalkMC> walk;
    const float m_Null = 0;

    if(state->robotDesign&WorldState::ERS7Mask)
            IRDistOffset=ERS7Info::NearIRDistOffset;
      else
    {
            serr->printf("ExploreMachine: Unsupported model!\n");
            return;
      }

      walkid=motman->addPersistentMotion(walk);

    //************  Nodes ************//
    //Walk
    PAWalk * move=NULL;
      addNode(move=new PAWalk(getName()+"::move",m_Speed,m_Null,m_Null));
      move->setWalkID(walkid);

      //turn
      turn=new PAWalk(getName()+"::turn",m_Null,m_Null,0.5f);
      addNode(turn);
      turn->setWalkID(walkid);
```

```
    start = turn;

      //**********  Transistions **********//
      move->addTransition(new SmoothCompareTrans<float>(turn,
                        &state->sensors[IRDistOffset],
                        CompareTrans<float>::LT,
                        350,
                        EventBase(EventBase::sensorEGID,
                                  SensorSourceID::UpdatedSID,
                                  EventBase::statusETID),
                                  .7));
      turn->addTransition(new TimeOutTrans(move,3000));

      StateNode::setup();
}

void PAExplore::DoStart()
{
      StateNode::DoStart();
      start->DoStart();
      erouter->addListener(this,
                         EventBase::stateMachineEGID,
                         (unsigned int)turn,
                         EventBase::activateETID);
}

void PAExplore::DoStop()
{
      erouter->removeListener(this);
      StateNode::DoStop();
}

void PAExplore::teardown()
{
      motman->removeMotion(walkid);
      StateNode::teardown();
}

void PAExplore::processEvent(const EventBase&)
{
    cout << "IRD: " << &state->sensors[ERS7Info::NearIRDistOffset] << endl;
      float vel=rand()/(float)RAND_MAX*2.0f-1;
      if(vel<0)
            vel-=.25;
      if(vel>0)
            vel+=.25;
      turn->setAVelocity(vel);
}
```

11.5 PAWalkTo.h

```
// Julien ROUX
// Heriot Watt University
// File: PAWalkTo.h
// Adatation of WalkToTargetNode.h from Tekkotsu
// 07 September 2005

#ifndef INCLUDED_PAWALKTO_H_
#define INCLUDED_PAWALKTO_H_

#include "Behaviors/StateNode.h"
#include "Motion/MotionManager.h"

class PAWalkTo : public StateNode
{
  public:
    //Constructor
      PAWalkTo(unsigned int obj)
            : StateNode("PAWalkTo","PAWalkTo"),
          tracking(obj),
              walker_id(MotionManager::invalid_MC_ID),
          headpointer_id(MotionManager::invalid_MC_ID)
      {}

      //Constructor
      PAWalkTo(const std::string& nodename, unsigned int obj)
            : StateNode("PAWalkTo",nodename),
          tracking(obj),
              walker_id(MotionManager::invalid_MC_ID),
          headpointer_id(MotionManager::invalid_MC_ID)
      {}

      virtual void DoStart();
      virtual void DoStop();

      static std::string getClassDescription() { return "go to a target"; }
      virtual std::string getDescription() const { return
getClassDescription(); }

      virtual void processEvent(const EventBase& event);

      virtual Transition* newDefaultLostTrans(StateNode* dest);
      virtual Transition* newDefaultCloseTrans(StateNode* dest);


protected:
      PAWalkTo(const std::string& classname, const std::string& nodename,
unsigned int obj)
            : StateNode(classname,nodename),tracking(obj),
                  walker_id(MotionManager::invalid_MC_ID),
headpointer_id(MotionManager::invalid_MC_ID)
      {}

      unsigned int tracking;
      MotionManager::MC_ID walker_id;
      MotionManager::MC_ID headpointer_id;
```

```
private:
      PAWalkTo(const PAWalkTo&);
      PAWalkTo operator=(const PAWalkTo&);
};

#endif
```

11.6 PAWalkTo.cc

```
// Julien ROUX
// Heriot Watt University
// File: PAWalkTo.h
// Adatation of WalkToTargetNode.h from Tekkotsu
// 07 September 2005

#include "PAWalkTo.h"

#include "Motion/HeadPointerMC.h"
#include "Motion/WalkMC.h"
#include "Motion/MMAccessor.h"
#include "Events/VisionObjectEvent.h"
#include "Shared/WorldState.h"
#include "Behaviors/Transitions/TimeOutTrans.h"
#include "Behaviors/Transitions/VisualTargetCloseTrans.h"

void PAWalkTo::DoStart()
{
      StateNode::DoStart();
      cout << "PAWalkTo starts" << endl;
      headpointer_id = motman-
>addPersistentMotion(SharedObject<HeadPointerMC>());
      walker_id = motman->addPersistentMotion(SharedObject<WalkMC>());
      erouter->addListener(this,EventBase::visObjEGID,tracking);
}

void PAWalkTo::DoStop()
{
      erouter->removeListener(this);
      motman->removeMotion(headpointer_id);
      headpointer_id=MotionManager::invalid_MC_ID;
      motman->removeMotion(walker_id);
      walker_id=MotionManager::invalid_MC_ID;
      StateNode::DoStop();
}

void PAWalkTo::processEvent(const EventBase& event)
{
      static float horiz=0,vert=0;
      const VisionObjectEvent *ve = dynamic_cast<const
VisionObjectEvent*>(&event);

      if(ve!=NULL && event.getTypeID()==EventBase::statusETID)
    {
            horiz=ve->getCenterX();
            vert=ve->getCenterY();
            cout << horiz << "x" << vert << endl;
```

```
      } else
      {
        cout << "-----------return" << endl;
            return;
    }

      double tilt=state->outputs[HeadOffset+TiltOffset]-vert*M_PI/6;
      double pan=state->outputs[HeadOffset+PanOffset]-horiz*M_PI/7.5;
      if(tilt>outputRanges[HeadOffset+TiltOffset][MaxRange])
            tilt=outputRanges[HeadOffset+TiltOffset][MaxRange];
      if(tilt<outputRanges[HeadOffset+TiltOffset][MinRange]*3/4)
            tilt=outputRanges[HeadOffset+TiltOffset][MinRange]*3/4;
      if(pan>outputRanges[HeadOffset+PanOffset][MaxRange]*2/3)
            pan=outputRanges[HeadOffset+PanOffset][MaxRange]*2/3;
      if(pan<outputRanges[HeadOffset+PanOffset][MinRange]*2/3)
            pan=outputRanges[HeadOffset+PanOffset][MinRange]*2/3;
      {MMAccessor<HeadPointerMC>(headpointer_id)->setJoints(tilt,pan,0);}

      {
            MMAccessor<WalkMC> walker(walker_id);
            if(pan<-.05 || pan>.05)
                  walker->setTargetVelocity(80,0,pan);
            else
                  walker->setTargetVelocity(130,0,0);
      }
}

Transition* PAWalkTo::newDefaultLostTrans(StateNode* dest) {
      return new TimeOutTrans(dest,1500,EventBase::visObjEGID,tracking);
}

Transition* PAWalkTo::newDefaultCloseTrans(StateNode* dest) {
      return new VisualTargetCloseTrans(dest,tracking, 120);
}
```

11.7 PAWalk.h

```
// Julien ROUX
// Heriot Watt University
// File: PAWalk.h
// Adatation of WalkNode.h from Tekkotsu
// 07 September 2005

#ifndef INCLUDED_PAWalk_h_
#define INCLUDED_PAWalk_h_

#include "Behaviors/StateNode.h"
#include "Motion/MotionManager.h"
#include "Motion/WalkMC.h"
#include "Motion/MMAccessor.h"
#include "Events/LocomotionEvent.h"
#include "Events/EventRouter.h"

class PAWalk : public StateNode
{
  public:
    enum WalkMode_t { VelocityWalkMode, DistanceWalkMode };

  public:
      //!constructor
      PAWalk()
        : StateNode("PAWalk"),
        walkid(MotionManager::invalid_MC_ID),
        walkidIsMine(true),
        x(0), y(0), a(0), n(-1),
        walkMode()
      {}

      //!constructor, positive @a yvel is counter-clockwise from above (to
match coordinate system), assumes velocity
      PAWalk(float xvel, float yvel, float avel)
        : StateNode("PAWalk"),
        walkid(MotionManager::invalid_MC_ID),
        walkidIsMine(true),
        x(xvel), y(yvel), a(avel), n(-1),
        walkMode(VelocityWalkMode)
      {}

      //!constructor, positive @a yvel is counter-clockwise from above (to
match coordinate system), assumes distance
      PAWalk(float xvel, float yvel, float avel, int steps)
        : StateNode("PAWalk"),
        walkid(MotionManager::invalid_MC_ID),
        walkidIsMine(true),
        x(xvel), y(yvel), a(avel), n(steps),
        walkMode(DistanceWalkMode)
      {}

      //!constructor, positive @a yvel is counter-clockwise from above (to
match coordinate system), assumes velocity
      PAWalk(const std::string& name, float xvel, float yvel, float avel)
        : StateNode("PAWalk",name),
        walkid(MotionManager::invalid_MC_ID),
```

```
        walkidIsMine(true),
        x(xvel), y(yvel), a(avel), n(-1),
        walkMode(VelocityWalkMode)
      {}

      //!destructor, check if we need to call our teardown
      ~PAWalk() {
        if(issetup)
          teardown();
      }

      //! sets the velocity of the walk
      void setDisplacement(float xd, float yd, float ad, int np = -1) {
        storeValues(xd, yd, ad, np, DistanceWalkMode);
      }

      //! sets the velocity of the walk
      void setVelocity(float xvel, float yvel, float avel, int np = -1) {
        storeValues(xvel, yvel, avel, np, VelocityWalkMode);
      }

      //! sets the velocity in x direction (positive is forward)
      void setXVelocity(float xvel) { x=xvel; storeValues(xvel, y, a, n,
VelocityWalkMode); }

      //! returns the velocity in x direction (positive is forward)
      float getXVelocity() { return x; }

      //! sets the velocity in y direction (positive is forward)
      void setYVelocity(float yvel) { y=yvel; storeValues(x, yvel, a, n,
VelocityWalkMode); }

      //! returns the velocity in y direction (positive is forward)
      float getYVelocity() { return y; }

      //! sets the velocity of the turn, positive is counter-clockwise from
above (to match coordinate system)
      void setAVelocity(float avel) { a=avel; storeValues(x, y, avel, n,
VelocityWalkMode); }

      //! returns the velocity of the turn, positive is counter-clockwise
from above (to match coordinate system)
      float getAVelocity() { return a; }

      virtual void DoStart()
    {
            StateNode::DoStart();
            if (walkid != MotionManager::invalid_MC_ID)
                erouter->addListener(this, EventBase::locomotionEGID,
walkid, EventBase::statusETID);
            cout << "PAWalk starts" << endl;
          updateWMC();
      }

    virtual void DoStop()
    {
          erouter->removeListener(this);
          if(walkid!=MotionManager::invalid_MC_ID)
        {
              MMAccessor<WalkMC> walk(walkid);
              walk->setTargetVelocity(0,0,0,0);
```

```
        }
          StateNode::DoStop();
      }

        //! receive locomotionEGID status event and throw stateMachineEGID
status event, ie a completion event
    virtual void processEvent(const EventBase& e)
    {
        cout << "PAWalk event" << endl;
          if (e.getGeneratorID() == EventBase::locomotionEGID)
        {
              const LocomotionEvent le = *reinterpret_cast<const
LocomotionEvent*>(&e);
              if (le.x == 0 && le.y == 0 && le.a == 0)
                  postCompletionEvent();
          }
    }


      //! removes #walkid if #walkidIsMine
      virtual void teardown()
    {
            if(walkidIsMine) {
                  motman->removeMotion(walkid);
                  walkid=MotionManager::invalid_MC_ID;
            }
            StateNode::teardown();
      }

      //! use this to force the WalkNode to use a shared WalkMC - set to
MotionManager::invalid_MC_ID to reset to internally generated walk
      virtual void setWalkID(MotionManager::MC_ID id) {
            if(walkidIsMine) {
                  motman->removeMotion(walkid);
                  walkid=MotionManager::invalid_MC_ID;
            }
            erouter->removeListener(this, EventBase::locomotionEGID);
            walkid=id;
            walkidIsMine=(id==MotionManager::invalid_MC_ID);
            erouter->addListener(this, EventBase::locomotionEGID, walkid,
EventBase::statusETID);
      }

      //! use this to access the WalkMC that the WalkNode is using
      virtual MotionManager::MC_ID getWalkID() { return walkid; }

      //! returns true if #walkid was created (and will be destroyed) by
this WalkNode - false if assigned by setWalkID()
      virtual bool ownsWalkID() { return walkidIsMine; }

protected:
      //! stores the values and if active, calls updateWMC()
  void storeValues(float xp, float yp, float ap, int np, WalkMode_t wmode)
{
    x = xp;
    y = yp;
    a = ap;
    n = np;
    walkMode = wmode;

    if (isActive()) {
```

```
      updateWMC();
    }
  }

      //! makes the appropriate calls on the WalkMC
  void updateWMC() {
    if(walkid==MotionManager::invalid_MC_ID) {
      SharedObject<WalkMC> walk;
      MotionManager::MC_ID id = motman->addPersistentMotion(walk);
      setWalkID(id);
      walkidIsMine=true;
    }
    MMAccessor<WalkMC> walk(walkid);
    switch(walkMode) {
    case VelocityWalkMode:
      walk->setTargetVelocity(x,y,a,n);
      break;
    case DistanceWalkMode:
      walk->setTargetDisplacement(x,y,a,n); // WalkMC will calculate
velocities.
      break;
    default:
      std::cout << "Unknown Walk Mode" << std::endl;
      break;
    }
  }

      MotionManager::MC_ID walkid; //!< the current WalkMC
      bool walkidIsMine; //!< true if the walk was created in updateWalk
(instead of assigned externally)
      float x; //!< velocity in x direction (positive is forward), or
distance if #walkMode is DistanceWalkMode
      float y; //!< velocity in y direction (positive is dog's left), or
distance if #walkMode is DistanceWalkMode
      float a; //!< velocity of the turn, positive is counter-clockwise
from above (to match coordinate system), or distance if #walkMode is
DistanceWalkMode
      int n; //!< number of steps (-1 means walk forever)

  WalkMode_t walkMode; //!< the current interpretation of #x, #y, and #a
};

#endif
```

11.8 Pictures

The robot we used during the development was called Biscuit. Following is a small portrait gallery.

Figure 20 'Biscuit' and his pink ball

Figure 21 'Biscuit' "eating" the prey

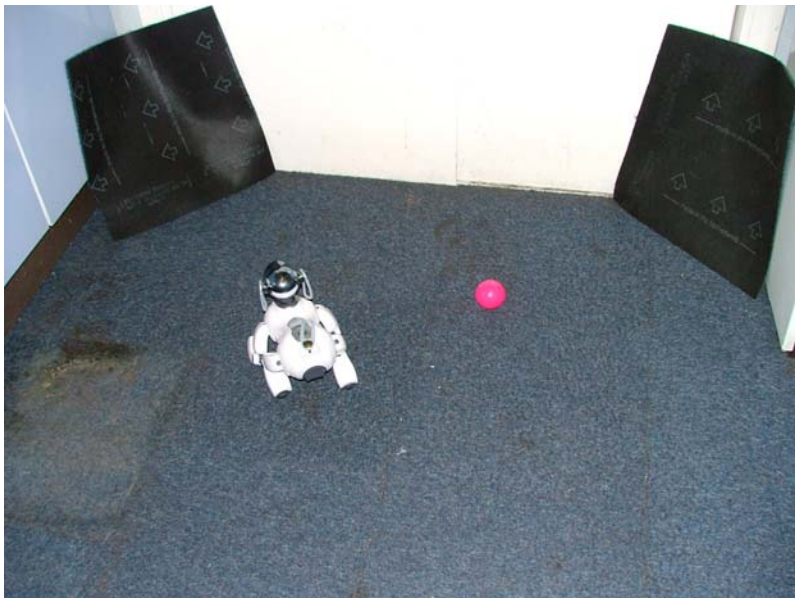


Figure 22 'Biscuit' looking for the prey

Figure 23 The development "environment"

11.9 Poster

HERIOT WATT UNIVERSITY

The pursuit ability of a robot 'pet'

By Julien ROUX
Supervisor: Patrick Greene
Dog: Biscuit

The context

Nowadays, robots are not only considered as slaves primarily intended to dangerous, dirty, repetitive, or difficult scenario. They can now be used for entertainment with no other specific tasks allocated. They simply mimic humans or animals, attempting to entertain or provide company to human beings.

This new generation of robots has generated a new market. The most popular of all these robots is the AIBO, launched by Sony in 1999. This robot pet, resembling a dog, was created by the Digital Creature Laboratory for Sony in Japan after six years of research lead by Doctor Toshitada Doi. This robot basically fulfils the most current constraints arising when we consider autonomous robots. It can work without being linked to a computer and it does not need a permanent plug to a power supply other than its battery.

Many researches about artificial intelligence and robotics have been done using AIBO and one of the main outcome is the Robocup. It is a tournament organized every year gathering impassioned of programming and robotics from all over the world and team from universities. The aim of this cup is to have robots playing a full game autonomously. Each team uses the same physical platform, the AIBO for the Sony Four-Legged Robot League. There are two models currently used, the ERS-7 and the older ERS-210.

For this project, the mimic a dog chasing a prey. It will first wander around, looking for it, and once the prey has been localised, the AIBO will chase it and eat it. If it loose it, it will go back to search for it. The behaviour has been implemented using finite state automata for hybrid architecture.

3 strategies for behaviour selection

Three kinds of architectures have been used since the beginning of artificial intelligence design. They are defined by the behaviour selection system used to make the robot behave without following a precise and pre-programmed order. The hierachical architecture also called deliberative architecture has been used since the late 1960 with Shakey the robot which was the first agent to demonstrate reasoning about its action. It was based on a 'sense then think then act' ideology. The idea was to plan the next action by generating alternatives before to choose between them. After this, the reactive architecture appeared in the 1980. This new architecture removed the planning part considered as too consumming, either in terms of ressources or computation. Sensors readings are directly translated into actions. Finally, another type of architecture appeared in the 1990 with a merge of the previous ones. Hybrid architectures decompose tasks into subtasks and then decide what the suitable behaviour for each subtask should be.

Also, having multiple agents performing a task together whilst showing signs of collective behaviour is a big challenge. They have to avoid unpredictable or harmful behaviour but they lack a global point of view of the problem. This criteria means that when several agent have to cooperate like during the Robocup, the design become more complex and the agents have to be organised.

What is a behaviour?

Concerning the AIBO, or any kind of autonomous agent, a behaviour refers to something meaningful. It is in general an action that accomplishes something or a sequence of actions. The agent has to decide which action to perform, when and how. It can be seen as what should be done under given circumstances or under which circumstances a given behaviour should be activated. Therefore, as it is impossible to prevent all that can happen in an open environment, the second solution will be more suitable. Many solutions have been designed to implement decision making for autonomous agents. Behaviours, be it in robotics or zoology, can be divided into three categories. Reflective behaviours are direct reactions to the input without the control from a third party. For instance, eyes will instinctively maintain a level of moisture supply to keep from drying by means of blinking. Reactive behaviours are also used without real conscience wareness of it but unlike reflexive behaviours, these are learned. Humans learn how to ride a bike or how to swim but once learned they will perform such activities without thinking about how to do it. The last type of behaviour is the one most complex. Conscious behaviours imply that the agent performing them has to be conscious of it. It is the case when cooking or playing a game.

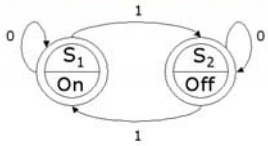
Decision trees

A decision tree is a hierarchical structure traversed from the top and downwards until a leaf is reach. It takes an object or a situation for input and returns a decision as a result. Each internal node of the tree is a test and the branches correspond to the possible values of the test. Each leaf node is a value to be returned if reached.

In the tree, a leaf corresponds to a simple behaviour while an higher node is more complex and often composed of the ones bellow.

Finite State Machine and Hybrid Automata

A Finite State Machine is a model of computation consisting of a set of states, start state, an input alphabet, and a transition function that maps input symbols and current states to a next state. In the example below, the machine will remain on the same state as long as the condition will be 0 and change state when the condition is 1



Hybrid automata are generalyzed finite-state machines for modelling hybrid systemms. Automata can be ritten using Tekkotsu with a class inheriting from the basic node called *StateNode*. This node is defined as either a state machine controller as well as a node within a state machine itself. This way, a node belongs to a hierarchical structure similar to the decision trees. Therefore, this is one of the reasons for using automata on this project.