Programming tools for Aibo

Sven Westermark

April 7, 2005
Abstract

In 1999 the Sony© Corporation released its first version of its entertainment robot, the four-legged doglike Aibo. Although intended primarily for private customers, the robot has been warmly embraced by the academic society, and is frequently used in beginner courses covering robotics.

The goal of this project has been to develop laboratory instructions and assignments, that can be used in such a course. The idea is that using this set of assignments, a genuine understanding of how the robot works will come quickly, and thus give the students more time on their hands for advanced experimenting and programming.

Secondly, this report contains a general overview of robotics. It gives an introduction as to how robots work, what the key features of autonomous robots are, as well as it introduces the jargon used in the robotics area. There is also a thorough investigation of the robot at hand, the Sony© Aibo.
Sammanfattning på svenska

Robotar har länge varit viktiga element i tillverkningsindustrin, och mer avancerade robotar har använts för att utföra arbete i olika typer av miljöer som är farliga för människor, exempelvis på havsbottnar, i eldvågor, och ute i rymden.

Dessa mer avancerade robotar erbjuder stora utmaningar för flera grupper av forskare och tekniker. Avancerad fysik och matematik krävs för att räkna ut hur robotarna ska kunna gå utan att falla. Det krävs mycket databehandling av sensordata, och såväl signalbehandling som reglerteknik kan komma ifråga, både för att förstå vad de insamlade datavärdena betyder, som att använda dem till att styra robotens uppförande på ett bra sätt.

Det krävs också mycket datavetenskap, då robotarnas program måste vara tillförlitliga och robusta. Speciellt viktigt är detta för autonoma (självständiga/självstyrande) robotar, som måste operera helt utan mänsklig styrning. Ett exempel på detta är månlandarrobotar, som kanske hamnar i ”radioskugga” eller skadar sin antenn, och måste klara sig själva.

Sedan något decennium finns det också en liten marknad för konsumentrobotar, och den tros bli större inom en snar framtid. I takt med att datorerna har blivit mer avancerade, mindre och billigare, så har det också blivit vanligare att sätta in mikrodatorer i allt från kylskåp till dockor. Detta har också möjliggjort framkomsten av de avancerade konsumentrobotarna, som till exempel Sonys™ Aibo och Hondas™ Asimo.


Fast den var tänkt som ett elektroniskt husdjur, en leksak, så har Aibo använts flitigt vid olika tekniska högskolor runt om i världen. Det främsta skälet till detta är förmodligen att Sony tillhandahåller programmeringsplattformen Open-R gratis, vilket gör det lätt för studenter att utveckla egna program och prova dem i Aibo.
Med en Aibo och Open-R installerat på en dator, så kan man utforska flera tekniska, matematiska och datavetenskapliga områden.

- Man kan arbeta med olika rörelsemönster för att se vilka gångstilar som lämpar sig bäst på olika underlag.
- Man kan skriva program för bildanalys, och på olika sätt analysera de filmer och stillbilder som Aibo kan ta med sin kamera.
- Man kan experimentera med röststyrning och -igenkänning, genom att ett program får analysera det som kommer in via Aibos mikrofon.

1 Introduction

1.1 Background

In the early spring 2004, the Uppsala University gave the first robot course using the Sony® Aibo. The course was project oriented, and had high ambitions as to what goals the project groups would achieve. However, the task proved to be harder than intended. A lot of time was spent just getting the basic features working, and it was awkward to learn how to use the Open-R kit, which is Sony’s programming environment for the Aibo. The need for introductory assignments was clear, and it was decided that developing those assignments was to be part of a degree project.

1.2 An overview of the 2004 course

The course given was named ”Embedded control systems”, and the way it worked was that the students were given a few project ideas to pick from. The different projects were:

- ”Identify Aibo”: Determine (via system identification) what transfer functions are operating in the Aibo.
- ”Go Aibo”: Write an algorithm that makes the Aibo adapt its walking style, depending on the terrain.
- ”Seeing while walking”: The Aibo has a camera in its nose, that is used for object identification. However, due to the rocking of the head, this can only be used while the Aibo is still or crawling. This should be overcome, by somehow compensating for the movement of the head, so there would be good pictures anyway.
• ”SimAibo”: The purpose of this project was to build a computer model of the Aibo, so that all its properties could be simulated. This would be done in Matlab/Simulink. Of course, to simulate the Aibo, it would have to be identified as well, so a close cooperation with the ”Identify” group would be necessary.

There were only three groups, so the ”SimAibo” project was left out.

1.3 Robotics background

Robots have been an important part of factoring industries for a long time, and more advanced robots have been used for performing work in human-unfriendly places, such as the bottom of the sea, fires, and in space.

The advanced robots offer great challenges for several groups of researchers and technicians. Advanced physics and mathematics are necessary to calculate how the robots should walk without falling. There is a considerable need for data processing. Both signal processing and automatic control are necessary, in order to both understand the collected data, and to use the data to control the robot’s behavior.

Computer science is also important. The programs that run in the robots must be reliable and robust. This is especially crucial in autonomous robots, which may be without human contact for long periods of time.

For the last decade or so, there has been a small market for commercial, consumer-dedicated robots. This market is predicted to grow in the near future. As computers have become more advanced, smaller and cheaper at the same time, it has become more and more common to put microprocessors into everything from refrigerators to dolls. This has also enabled the spawning of advanced commercial robots, such as the Sony© Aibo and the Honda© Asimo.

1.4 Objectives

1.4.1 Problem formulation

When you get your hands on a fresh Aibo, you have a huge task in front of you. You have to learn about all its mechanical and electronical features, such as its limbs, its sensors, camera and so on. There is an entirely unique software platform, the Open-R environment, possibly with the Tekkotsu framework on top of it. Learning these can be tricky, as their documentation
leave a lot to be desired. Just downloading all the files, and searching the
documentation for how to do, can be frustrating and time-consuming.

1.4.2 Purpose and aim

The purpose of this project is to develop introductory assignments that will
aid students toward a better understanding about how to utilize the Aibo,
so that they do not waste time on learning basic things the hard way.

1.5 Disposition

This report is structured in the following way: Sections 2 and 3 give a general
overview of robotics, and a thorough investigation of the Aibo, respectively.
Section 4 describes the problems experienced by the former project groups,
and section 5 contains the assignments themselves. Section 6 summarizes
and discusses the results of this project. The appendixes consist of example
code for the Aibo, followed by some configuration files, and also some screen
shots from the Tekkotsu user interface.
2 Robotics - a general overview

2.1 Definitions

There are two especially important topics when it comes to walking robots. One is forward kinematics, and the other is inverse (or backward) kinematics. The latter is usually the most important, and it can be derived from the former. For more information on kinematics, [3] is a very good source. In this chapter, the following terms will be used.

- **Chain** - A set of linked rigid bodies, sequentially connected by joints. Each link can bend and/or rotate relative to the previous one.

- **Degree of freedom (DOF)** - A ball in a room can go in any direction it likes to, so it has 3 DOF:s (x,y,z). On the other hand, the gas pedal of a car can move in one dimension only, so we say it has 1 DOF. It is common when you describe robots, that you mention how many DOF:s it has.

2.2 Kinematics

2.2.1 Forward kinematics

Many mechanical objects in the real world consist of solid sections connected by joints. Examples of these are:

- Creatures such as humans and animals
- Car suspension
- Robot arms
- Ropes, string and chains

A good example of a chain is a robot arm, where each section of the arm can pivot, bend and/or rotate from the previous stage. In biological terms, say when the object at hand is an animal or human, a movable section can be referred to as a bone, and the attachments between each pair of bones as joints. Motions of chains can be specified in terms of translations and rotations.

An example of forward kinematics is the following: From the amounts of rotation and bending of each joint in an arm the position of the hand can be calculated.
Chains are usually represented with no information about their actual shape, because the only things that are interesting are the relative positions of all joints. The first link is called the base, or "anchor point" in nautical terms. The base is the end of the chain, and it always has a fixed or known position. From the base and on, each link can be rotated, possibly in different dimensions depending on the degree of freedom of the joint. The other important factor is the distance between the joints, which can be represented as a translation.

So forward kinematics allows one to find the position of the endpoint (and all the points in-between) relative to the base. This is rather straightforward, it is just a matter of matrix multiplications.

2.2.2 Inverse kinematics

For inverse kinematics (IK), the position of the end point is known, and the problem is to find the angles of the joints. This is a much harder problem as there may be many different solutions, or there may not be a set of angles that would reach to that point. When dealing with IK problems, it can be handy to build a graphical, interactive computer model of the system at hand. This can give the human user an intuitive feel of what the limitations are, and how to perform best under those conditions.

There are two approaches to solving inverse kinematics problems:

- Analytical - requires a lot of trigonometry or matrix algebra
- Iterative - works better if there are lots of links and degrees of freedom

IK - Analytical approach

If there are only two or three links then it may be possible to solve the IK problem analytically. One possibility is to draw the arm with the angles shown on it, then solve for the angles using geometry. The problem is that this approach does not generalize to more advanced cases. Another analytical approach is to represent the impact that each single joint has on the chain, by a matrix that varies depending on the angle(s) and relative distance (to the anchor) that the joint is set to. With this approach, the total impact of the chain, ie the position of the end point, can be found by multiplying all the joint matrices together.
However, this does not always work. There may be many solutions or there may be no solutions. In other words there are lots of ways to reach to a given point, but it may also be out of reach.

If there are many solutions, additional constraints may be applied, sorting out the best candidate for the solution. Examples of constraints can be bounding of one angle, eg human joints can only bend within certain limits, or some cost function such as energy consumption.

**IK - Iterative approach**

This is a more general approach for programming complex chains.

Start off with the joints in any position, then move each of the joints in turn, so that each movement takes the endpoint toward the target.

Starting with the joint nearest the end point, rotate the joint so that the current end point moves toward the required end point. Then do the same with the next joint toward the base and so on until the base is rotated. Then keep repeating this, until the end point is close enough to the required end point or if further iterations are not moving it closer to the required point. It may be possible to have a more realistic strategy than this, for instance, if I am using my arm to pick up an object then, if the object is far away, I will move the bigger joints in the arm, then as the hand gets closer the smaller joints of the hand are used for the fine adjustments.

The angle of rotation for each joint is found by first calculating the dot (or scalar) product between the two links attached to that joint, and thereafter using the arcsin() function to obtain the angle.

### 2.2.3 Solving an IK problem on the Aibo

The problem to be solved is: What is the relationship between the paw (P=(x,y)), and the two joint angles $\alpha$ and $\beta$?

In this example, the z-component has been neglected, thereby reducing the problem to solving two, rather than three angles.

The desired joint angles (see Figure 2.2.2) are calculated using the following derivation. We begin by expressing the known parameters $x$ and $y$ as a system of function of the unknown angles.

\[
x = By \cdot \sin \beta + Ay \cdot \sin(\alpha - \beta) \tag{1}
\]

\[
y = By \cdot \cos \beta + Ay \cdot \cos(\alpha - \beta) \tag{2}
\]

Where:
Figure 1: The knee joint of an Aibo
\( \alpha = \) Elbow or knee angle  
\( \beta = \) Shoulder angle  
\( A_y = \) Length of lower limb = 66.71 mm for the front leg and 76.49 mm for the rear leg.  
\( B_y = \) Length of upper limb = 61.85 mm.

Squaring equations 1 and 2 gives the following system:

\[ x^2 = B_y^2 \cdot \sin^2 \beta + A_y^2 \cdot \sin^2(\alpha - \beta) + 2 \cdot A_y \cdot B_y \cdot \sin \beta \cdot \sin(\alpha - \beta) \]  \( (3) \)

\[ y^2 = B_y^2 \cdot \cos^2 \beta + A_y^2 \cdot \cos^2(\alpha - \beta) + 2 \cdot A_y \cdot B_y \cdot \cos \beta \cdot \cos(\alpha - \beta) \]  \( (4) \)

Adding 3 and 4 gives:

\[ x^2 + y^2 = A_y^2 + B_y^2 + 2 \cdot A_y \cdot B_y \cdot (\cos \beta \cdot \cos(\alpha - \beta) + \sin \beta \cdot \sin(\alpha - \beta)) \]  \( (5) \)

Using trigonometric identities, 5 reduces to:

\[ x^2 + y^2 = A_y^2 + B_y^2 + 2 \cdot A_y \cdot B_y \cdot \cos \alpha \]  \( (6) \)

Hence:

\[ \alpha = \arccos \frac{x^2 + y^2 - A_y^2 - B_y^2}{2 \cdot A_y \cdot B_y} \]

Similar calculations lead to the following expression for the shoulder angle, \( \beta \).

\[ \beta = \arcsin \frac{(B_y + A_y \cdot \cos \alpha) \cdot x - A_y \cdot \sin \alpha \cdot y}{B_y^2 - 2 \cdot A_y^2} \]

In this example, it is possible to determine an analytical solution to the IK problem. As stated previously, in a system with more DOF:s and more links, this is not generally the case.
3 Equipment

3.1 The Aibo

The silver-colored entertainment dog Aibo, produced by Sony©, is one of the most well-known robots to the general public. It is used both in soccer competitions and as an electronic (but expensive) pet. The robot is handy for educational purposes, because the programming platform gives total freedom to the programmer. It is an open-source program, which you can examine, modify and extend if you wish.

The Aibo is a four-legged robot, designed to look like a dog (see Figure 3.1). It has twenty internal actuators, that enable it to move in a natural way [1]. It is equipped with a camera and an IR sensor in the nose, pressure-sensitive buttons on the head and the back, as well as sensors in the paws which sense when they hit the ground. It also has sensors for temperature and vibrations. In addition to this, the Aibo can express itself via LEDs in the face (“eyes”) and the tail, which it can also wag. A speaker enables it to bark and emit other sounds, and a microphone enables it to hear in stereo. The Aibo has a 64 bit RISC processor, with a clock frequency of 166 MHz. Its main memory is 32 MB big, and Sony supplies Flash memory sticks of size either 8 or 16 MB. Also optional is a WLAN card especially designed for the Aibo, which is used for communication with a personal computer. The memory stick is necessary to put your own programs into the Aibo. In order to use Tekkotsu, you need the network card as well.

3.2 Open-R

The Open-R Software Development Kit (SDK) is the official programming platform for the Aibo, provided for free by Sony©. The programming language used to develop programs with the SDK is C++. Aibo uses a proprietary real-time operating system called AperiOS. AperiOS fully implements
a subset of the standard C++ API functions, and provides slightly modified versions of other basic system calls [2].

The SDK mandates that the programs written for the Aibo conform to a certain object-oriented style. Every process executing on the Aibo takes the form of an object which inherits attributes from a standard object provided by the SDK. Not only can learning this programming style be difficult, Open-R also demands from the programmer to change the settings in a couple of configuration files, object.cfg and connect.cfg. object.cfg defines what objects (compiled binaries) that AperiOS should run, and connect.cfg gives AperiOS information about what objects communicate between each other, so that it can control the message-passing. Examples of object.cfg and connect.cfg can be seen in Appendix D and Appendix E, respectively.

Unlike ordinary programs which have only one starting point, namely when the user launches them, an Open-R Object can be invoked whenever one of the following occurs:

- Initialization in functions DoInit() and DoStart().
- Receipt of a message by the function designated to handle messages for this observer. The observer is the recipient of a message for a communication channel. The communication ends for each object is defined in the file 'stub.cfg'.
- Receipt of an Assert Ready signal as a confirmation that an observer has received a message from a subject of this object is ready to receive new messages. The subject is the sender end of a communication channel.
- Destruction in functions DoStop() and DoDestroy().

3.2.1 WLAN

The SDK provides wireless LAN facilities for communication between the robot and the development machine. This is implemented by the memory stick holding a special file called wlanconf.txt, that contains information about the wireless network the Aibo should join. This file can be seen in Appendix C.

The network connection gives the programmer a good opportunity to monitor the sensor values of the Aibo, passing it commands, and also keeping track of how the main program’s execution is proceeding. However, due to process scheduling and message pipelining, there is a risk that if the Aibo crashes
(software-speaking) the final messages that could explain what caused the crash might be lost.

3.2.2 emon.log

When an exception error (i.e crash) occurs on the Aibo, the content of the processors, registers etc. is written to the file ‘emon.log’ on the memory stick. The information in this file can be processed by a Perl-script, which outputs the object and function that was the cause of the crash. These functions are also defined in ’stub.cfg’.

3.2.3 Debug

When a lot of functionality is added to an application, it is advantageous to develop the program for the personal computer first if possible, and later translate the program back into an Open-R object. This makes it possible to debug the program with a debugger like the Gnu Debugger (GDB). Another option is to make use of the Remote Processing Open-R which enables execution of AIBO objects locally on the development machine. The disadvantage of this method is that the other object still has to run on the robot at the same time.

3.3 The Tekkotsu framework

The Tekkotsu is an application framework designed for robotic platforms. It has been developed at the Carnegie Mellon University, and is continuously updated. Based and derived on Sonys OPEN-R framework, it offers the user high level programming functionality without neglecting performance aspects. Tekkotsu itself is an object oriented and event casting architecture, with its strength lying in its lower level programming encapsulation. With the Tekkotsu platform, the user does not have to know the details and specifics of OPEN-R programming.

3.3.1 Behaviors

Creating an application in the Tekkotsu environment is straight forward and will be explained in more detail further on. However, since the applications are run on a robot designed to look like “man’s best friend”, the Tekkotsu applications are always referred to as Behaviors, thus giving the whole system a hint of personality. The behaviors are implemented through a standard C++ class interface (example see Appendix A), and play the role of executing decisions placed there by the programmer, due to sensor readings or other
circumstances. A behavior contains a default and/or a custom constructor and if needed a destructor. For initiating the behavior a special method named `doStart` is called at runtime start-up, and equally an ending method, `doStop`, is invoked at runtime shutdown. At the heart of the behavior is the method `processEvent`, which handles all events.

The system waits idle and "listens" for different events (events are described below) to occur. Once occurred, the `processEvent()` is invoked and whatever commands written there are then executed. After completion the procedure `doStop()` is invoked, terminating the application/behavior.

### 3.3.2 Event casting

Adding new functions to a behavior is done by adding event listeners, that will take the appropriate action when the desired event occurs. In the code, it will look something like this:

```plaintext
class addListener(this,EventBase::sensorEGID,0);
```

E.g the behavior listens to the thermal heat sensor, and performs some task if the sensor value reaches a certain threshold.

The system awaits for the task to perform and immediately afterwards employs the `processEvent()` method, executing whatever code is written there. Events can either be periodic, or occur as a consequence due to some external action.

### 3.3.3 Motion commands

Motion commands are collections of classes handling various kinds of kinetics for the Aibo. The programmer does not need to know how they work, or other specific details about them. He merely needs to know what they do. After a listened event has occurred and the method `processEvent()` has been invoked, code in that function is executed sequentially along with any employed motion command. First off though you need to declare a motion command object to your behavior in the `doStart()` procedure invoked during start-up. The declaration should also specify which type of motion should take place, e.g. should a walk pattern be loaded. With this declared, Tekkotsu handles all lower level function calls and executes whatever motion is stated.

### 3.3.4 Tekkotsu GUI

One of the main reasons for choosing Tekkotsu as a programming platform for your applications is its easy-to-use graphical user interface (see Figure 3).
Using the GUI enables you to easily start and stop different behaviors, read and write to input/output, and also see what the Aibo’s camera is seeing. The ability to alternate between different behaviors is dependent on the fact that a correctly configured `Startup_SetupBackgroundBehaviors.cc` (see Appendix B) has been compiled onto the Aibo’s memory stick.

4 The problems encountered by the project groups

4.1 Identify Aibo

The group started on trying to identify one of the Aibo’s joints, namely one of the knees. In order to do a proper system identification, they had to be able to store important data. Although this was straightforward, since the Aibo measures all the desired data, the group still had to write their own Open-R application to store all values in files.

This took them quite a while, due to the poor documentation of Open-R. When they managed to output this into files, there was a clash between the format of those files, and the format accepted by Matlab. So the group built their own parser, to convert the file from the Aibo, into a file readable my Matlab.

When they were finally in the stage where they could process their data in Matlab, the results were not satisfactory at all. Whereas the Aibo’s documentation claimed that the joint was controlled by a PID-regulator, the group had a very hard time trying to identify any of the PID-parameters. However, this thesis will not delve into the group’s bad results, but rather puts the focus on why the first steps took so long, leaving the project group with precious little time to do their real work.

4.2 Go Aibo

This project group had a task that sounds a lot less difficult than the previous one. Merely create a program for the Aibo, that somehow measures what the current ground is like, and adapts its walking style (aka ”gait”) thereafter. That is, it should take higher steps on rough ground, so it does not trip, and lower steps on flat ground. The group took the approach that the Aibo should measure its own speed at all times, and change its gait if it noticed that the speed was decreasing. The basic assumption was that an ”offroad gait” will be slower on a flat floor, and the other way around. Hence, just
taking decisions based on the development of speed would give satisfactory results. It appears that this group had a fairly easy time writing their Open-R applications, namely one program that measured and timestamped the necessary data, one command interface server, and a program that controls the motions of the Aibo. This group instead ran into problems calculating the speed of the dog. The Aibo lacks a speedometer, but has an accelerometer that gives the acceleration in three directions (x,y and z). Although speed could be obtained from the acceleration sensor (via integration), the group had to calculate the orientation of the acceleration, because this sensor was very shaky, and moved sort of like the head does (see below). This altogether proved to be too hard, so the group's work failed on that they could not calculate the speed or travelled distance of the dog.

4.3 Seeing while walking

This group was probably the most successful. That is, they did not reach their goal completely, but their failure was mostly due to built-in limitations of the Aibo. Unlike the other two groups, this one early on embraced Tekkotsu. As stated above, Tekkotsu is a framework that works on top of Open-R, and simplifies Aibo programming considerably. This probably saved this group a lot of time, enabling them to test a lot of different strategies. The main thing in this problem was to determine how the legs movements affected those of the head, in both the x- and the y-coordinate, i.e. right-left and up-down. The group tested several ways to find a connection, using sampled data from the Aibo to do their calculations. However, it turned out that the head’s movements were too high-frequent to be eliminated. A solution similar to Tekkotsu’s sample program *HeadLeveller* only took away the low-frequent parts of the shakes, actually making it worse in the higher frequency band.

As the group instead tried to use system identification algorithms with the acceleration sensors as input and camera movement as output, it became clear that the model used was erroneous. The assumption that the head could move rather freely was wrong, since there were PID-regulators controlling the head. There was also a backlash in the head joint motor, which made it impossible to do what the group intended. This backlash was not a malfunction on that particular Aibo, but rather something all Aibos suffered from.
5 Introductory assignments

5.1 Lab 0-Optional

Installation and configuration of the platform for Windows

This assignment is voluntary, and interesting only for those students who wish to use the Aibo platform in their home or portable PC. Although you may not be permitted to take the Aibo home with you, it may still be advantageous to have the Open-R platform and such at home. After this assignment, you should have a ready and working installation of Cygwin, Sun JDK, Open-R, Tekkotsu, and TekkotsuMon in your PC.

5.1.1 Notes

Important

Cygwin is an absolute requirement for running Open-R on a Windows machine. If you already have Cygwin installed, skip step 2.1.

Line breaks

You will encounter numerous text files, readme files, C++ programs etc, which all have UNIX-style linebreaks. This means that Notepad and other Windows programs will not break the lines in those documents correctly. There are other editors that do, the author recommends XEmacs. XEmacs also has features for syntax fortification of C++ programs, as well as other features.

JDK

The JDK (Java Development Kit) is necessary in order to run the TekkotsuMon ControllerGUI, which is a very good user interface. If you already have it installed, just set the CLASSPATH and skip the rest of step 2.2. Despite the look of it, you can access all of your hard drive from inside Cygwin. For instance, if you already have JDK installed in c:/program/jdk, the Cygwin command will be: "export CLASSPATH=$CLASSPATH:/cygdrive/c/program/jdk/"

Cygwin

If you’re new to Cygwin: Cygwin is a UNIX-like environment for Windows. Most UNIX commands are valid inside Cygwin, but some have been omitted.
If you’re new to UNIX as well, here is a small list of commands:

ls: list the current directory
pwd: tells you what is your current directory
cd: ‘dir’ changes current directory to dir. ‘cd ..’ takes you up one level, ie to the parent directory of the one you’re in.
cp: ‘file’ ‘dest’ copies file to destination dest
mv: ‘file’ ‘dest’ moves file to destination dest
rm: ‘file’ removes file. to remove directory, use rm -r ‘dir’. warning, removing is permanent.

Cygwin has environment variables. (As does Windows, but you rarely see them.) The ones you may need to know are PATH and CLASSPATH. To see the value of PATH, type ‘echo $PATH’. To extend PATH, type ‘export PATH=$PATH:/my addition/’. Don’t forget the $PATH, otherwise you clear the PATH entirely. The same goes for CLASSPATH.

.tar

Although the ‘tar’ program is included in Cygwin, the author of this guide had problems unpacking the Open-R and Tekkotsu source files from Cygwin. This is probably due to some file system clash, or other collision of formats. If you have this problem, or just want a good graphical view of what you’re doing, there are lots of compression utilities to download for free. One example is ‘Hard Compress’, but there are lots of others as well.

5.1.2 Getting the software


- Download JDK version 1.4 or later, from http://java.sun.com. Install in /usr/local, under the Cygwin directory. This will make the pathing easier. Set your Cygwin CLASSPATH, with the following command:

  'export CLASSPATH=.:.../usr/local/JDK/...:/home/nisse/myfiles/:/otherpaths'

  If you want to extend the Classpath later on, just type :

  'export CLASSPATH=$CLASSPATH:/more_locations/'

  You’ll also need to add jdk/bin to the environment variable PATH, or Cygwin won’t find javac and java. This is accomplished through:

  'export PATH=$PATH:/usr/local/jdk/bin'
In case you want to use Java in Windows as well, you need to set CLASSPATH from the Command Interpreter. The syntax is: 'set CLASSPATH=my_classpath', where my_classpath is the same one as above. This Windows environment variable is not shared by Cygwin, so it needs to be set indeed.

- Download the Open-R SDK, simply go to http://www.openr.org and register an account. Membership is free, but for some reason, they want you to be a member.

  The files you need are:
  The Open-R SDK itself. MIPS cross-development tools for Cygwin. jpegsrc.v6b source files possibly the gcc source files, but you can try if the gcc in Cygwin is enough.

  What you should also download, but don’t ‘need’:
  The Open-R SDK documentation, which you should read.
  Open-R sample programs, these are good for understanding Open-R. 3D model for your Aibo. This is really cool.

- Download Tekkotsu.

  The URL is http://www.tekkotsu.org. The files are found in the download area, but it is also advisable that you surf the page for a while. It contains demo videos, screenshots, tutorials and such.

  The files you need are:
  Tekkotsu
  Tekkotsu documentation
  TekkotsuMon
  precompiled Memory stick

  Unpack Tekkotsu and its documentation into separate folders in /usr/local. The TekkotsuMon should be installed under Tekkotsu/tools/, and there are instructions for this where you downloaded it. The Tekkotsu-memstick you can unpack wherever you wish, but it’s handy to have everything under /usr/local.

Network settings

Copy the folder Tekkotsu_Memstick_ers_xxx to the memory stick. In the folder open_r/system/conf, there is a file called wlandflt.txt. Don’t change this, but open it in an editor.
On your PC, go to Control Panel/System/Device Manager/Network cards, and select your WLAN card. You need to set the mode to Ad-hoc, the channel and SSID to the same as on the AIBO (see the text file). After this is done, your connection should work. Confirm this by typing (from Cygwin) 'ping 10.0.1.100', where 10.0.1.100 is the default IP of the AIBO. Further instructions can also be found in the "Installation guide" of the Open-R documentation. Now you’re ready to start working!

5.2 Lab 1 - Getting acquainted with the Aibo

5.2.1 Introduction

The goal of this assignment is to familiarize yourself with the Aibo. By trying it out with the default program, you will get a basic idea of how the Aibo moves, sounds, where the LED’s are situated, and what simple means of interaction it has. Then by running the Aibo under Tekkotsu, you will see some of the more advanced things the Aibo can do. Hopefully, seeing the provided behaviors should give you ideas of your own. Finally, you should try and modify one of the behaviors yourself. This should make you confident that you can be this dog’s master! :)

5.2.2 What to do

Test Aibo with the built-in program

The user’s guide tells you how the Aibo acts when this program is active. Verify that it works.

Try out Tekkotsu

You should copy the Tekkotsu-Memstick directory onto an otherwise empty memory stick. Set up the network settings on the Aibo, ie copy wlanfll.txt to wlanconf.txt and make the appropriate changes. Information on how to do this is in the installation guide, which can be found in the Open-R documentation. Verify that you have a connection, by typing 'ping AIBO-ip-address'. The default value of the AIBO’s IP address is 10.0.1.100, but you can set this as you wish, as long as it is in the same range as the computer it is communicating with.
Using TekkotsuMon

Change directory to `/usr/local/Tekkotsu/tools/mon`, and type `./ControllerGui AIBO-ip-address`. Now you should see a GUI which enables you to switch between behaviors. Select a Behavior by double-clicking it and press 'Go!'. An especially funny Behavior is the Walk Remote Control, which gives you full freedom to steer the Aibo around.

Look up the Tekkotsu tutorial

Make SampleBehavior in the way described in ”Creating your first Behavior”. Add an entry in StartupBehavior_SetupModeSwitch.h in order to be able to select it. Congratulations to making your first Behavior!

Modify SampleBehavior

Program the Aibo to make a sound when patted, instead of blinking LEDs. Note! When adding a new Behavior, don’t forget to add an entry in StartupBehavior_SetupModeSwitch.h.

Your idea

Come up with an own idea of a Behavior.

5.3 Lab 2 - Data processing

5.3.1 Introduction

There are uncountable applications where it is necessary to record or sample the values of a set of given variables, merely automatic control and signal processing contain lots of such scenarios. It is often the case that one is trying to find a connection between one or more 'input' variables and ditto output. Robotics, and especially autonomous robots, are no exception from this. Since they’re out in the real world, and depend on themselves to perform their task, they really need to try and acquire as much data as they can.

This assignment will teach you how to access sensor data from the Aibo, and you will also learn how to store these in files for later processing.
5.3.2 What to do

Collecting data

Create a Behavior that collects data and saves it into a file. One example of
data that can be collected is the values of the acceleration sensors. Search
through the Doxygen documentation of Tekkotsu, to find out how to access
sensors, as well as how to manage files. Note: examine how Matlab stores
values (in .mat files), and make sure your output file matches that format.

Using the data

Use the collected data to create a data sequence in Matlab. Now you could
plot different sensor values against time, and make FFT:s to find significant
frequencies.

Note: when sampling values in the Tekkotsu framework, you should make a
record of what time each value was sampled. Tekkotsu is a high-level, multi-
tasking environment, which cannot guarantee that values are sampled at a
given, constant frequency.

5.4 Lab 3 - Using Open-R

5.4.1 Introduction

The default platform for the Aibo is Open-R, provided freely by Sony. How-
ever, the Tekkotsu framework, developed at the Carnegie Mellon University
is also free, and a lot more user-friendly than the former. That is the reason
these assignments focus mainly on Tekkotsu.

Despite this, there are reasons to try using Open-R in itself. For one,
Tekkotsu comes with no warranties, and if you discover a bug in it, you’ll
probably have to fix it yourself. So if Tekkotsu cannot achieve what you want
it to do, or if you’re really into low-level C++ programming of robots, you
should take a deeper look upon the Open-R platform.

After this assignment: You should be able to write simple programs using
pure Open-R. You should understand why the directory structure of the
memory stick looks the way it does, and what the ’object.cfg’ and ’con-
nect.cfg’ files do.

5.4.2 What to do

We have been using the Tekkotsu framework in the previous assignments, and
hopefully this has been satisfactory. However, it can be useful to know a bit
of Open-R as well. A good idea is to study the Programmers Guide, which is located in the Open-R documentation folder. This holds useful information about how to make your programs in Open-R.

A good rule of thumb in making an Open-R program is this:
Go to OPEN_R_SDK/OPEN_R/MS_X/memprot, where X is the model of your Aibo. Copy the entire OPEN-R folder which you find here onto your memory stick. This will give you the Aperios and other necessary files, but not the actual program ("Behavior").
Create your program by typing 'make install' in the directory where your program is located, eg MYDIR/sample/HelloWorld/, which will generate the .bin files into the subfolder MS. From the MS/OPEN-R/MW/OBJJS folder, copy the .bin files to the same path on the memory stick. Also copy MS/OPEN-R/MW/CONF/object.cfg to the memory stick, which will overwrite the default object.cfg.
After this, insert the memory stick into the Aibo, and just turn it on. More detailed instructions as to how to write your own Open-R programs can be found in the Programmers guide. Looking at the sample programs can also be helpful.

5.5 Lab 4 - Advanced features

5.5.1 Introduction

A walking robot yields rather complex equations regarding how the limbs and joints affect each other. A solution can usually be found solving a set of coupled differential equations, while also making some more or less realistic assumptions. Although this may have to be done at the pioneer stage, ie at the factory, solving this kind of problem is not something a robot user can easily do himself.

A good idea is therefore to experimentally determine fair approximations of unit step responses, linking one variable to another at a time. This will give the user something to work with, eg s/he can know what effect a change in variable x1 has in x2.

After this assignment: you should know how to sample values in the Aibo, as well as how to store them for later usage.

5.5.2 What to do

Study the sample file SampleBehavior2.cc. What this behavior does is that it records the values of the acceleration sensors in a text file. Notice how the sensor values are obtained, merely by calling 'state->sensors[desired_offset]'.
To see what sensors have what offsets, look through the Doxygen documentation of the \textit{WorldState} class. Modify the sample behavior by changing what values to record. For instance, log the velocity (several components) and the position of one (or more) of the legs over a period of time.

5.6 Lab 5 - Debugging programs on the Aibo

5.6.1 Introduction

For a robot, especially an autonomous one, it is crucial that the main program never hangs up, or that the robot crashes. In order to make your programs that robust, you will probably have to rewrite, retry, and debug your program over and over again.

So how do you debug? If you only loaded a program in your robot, and observed it until it didn’t do the right thing anymore, you would be clueless as to what really happened. Therefore, you must have some kind of logging. What should be logged, how, and where it should be stored, is dependant both on your robot and your application.

\textit{This assignment:} gives you the background information on why debugging can be necessary.

5.6.2 How to do

In case you want to write your behaviors in pure Open-R, it is advisable that you read all of chapter 5 in \textit{Programmers guide for Aibo}. It contains detailed information on how to interpret the 'emon.log' file, as well as a description on how the MIPS processor works.

For those of you who work solely in Tekkotsu, or write bug-free programs from start ( :-) ), the debugging part might not be necessary at all.

6 Discussion

The Aibo robot is a rather intricate thing. It is not only a machine with physical properties and parameters that can be set, such as the pace speed, the PID controllers, and the angles of the knees. It is also a full-fledged computer with a CPU, memory, inter-communicating processes, and lots of inputs and outputs. Last but not least the Aibo has a special interface to the PC, enabling a real-time communication and controlling over a wireless network connection.
Although engineering students know about all of these things, it takes time to get acquainted with a complex machine covering all the aspects. Since there is a lot of programming involved, and the programming syntax is specialized on the platform in question, writing your first program unaided will probably produce something that does not work at all. As most people who have learnt different kinds of programming would agree, seeing a working example is better than hours of theoretical lectures about how to write programs. Therefore a big part of the introduction, aside from the assignments themselves, will be to study the example behaviors that are provided, which have a hopefully comprehensible documentation as well.
The assignments can be found in the Appendix.

References


[3] Euclidean space http://www.euclideanspace.com Euclidean space is a web site dedicated on teaching the mathematics and physics concerning 3D-modeling and kinematics.

A Example code

This is an example of what a Tekkotsu Behavior might look like. This particular Behavior makes the dog emit a barking sound when it is patted on the back. Note that there are many headers included, so not all of the code necessary for this is visible in this file.

```c++
// -*-c++-*-
/*
Litet kul Behavior, skiller nr man trycker p knappen.
*/

#ifndef INCLUDED_ExBehavior2_h_
#define INCLUDED_ExBehavior2_h_
#include "Behaviors/BehaviorBase.h"
#include "Motion/MotionManager.h"
#include "SoundPlay/SoundManager.h"
#include "SampleMC.h"

class ExBehavior2 : public BehaviorBase{

public:
 ExBehavior2() : BehaviorBase("ExBehavior2"){}

 virtual void DoStart(){
 BehaviorBase::DoStart();
 //my code.
 mirror_id=motman->addPersistentMotion(SharedObject<SampleMC>());
 erouter->addListener(this,EventBase::buttonEGID);
 }

 virtual void DoStop(){
 //my code.
 motman->removeMotion(mirror_id);
 erouter->removeListener(this);

 BehaviorBase::DoStop();
 }

 virtual void processEvent(const EventBase& event){
 if (event.getGeneratorID()==EventBase::buttonEGID){
```
woof=SoundManager->playFile("3barks.wav");
}

else {
    cout<<"Bad Event:"<<event.getName()<<endl;
}
}

protected:
    MotionManager::MC_ID mirror_id;
    SoundManager::Play_ID woof;
};
#endif

B StartupBehavior_SetupBackgroundBehaviors.cc

This file defines the list of Behaviors that you can choose between from within the TekkotsuMon. Notice that several Behaviors can run concurrently, eg the Aibo can both walk and move its head at the same time.

#include "StartupBehavior.h"
#include "Behaviors/Controls/ControlBase.h"
#include "Behaviors/Controls/BehaviorSwitchControl.h"
#include "Behaviors/Demos/SimpleChaseBallBehavior.h"
#include "Behaviors/Demos/StareAtBallBehavior.h"
#include "Behaviors/Demos/AutoGetupBehavior.h"
#include "Behaviors/Demos/BatteryMonitorBehavior.h"
#include "Behaviors/Demos/HeadLevelBehavior.h"
#include "Behaviors/Demos/ToggleHeadLightBehavior.h"
#include "Behaviors/Demos/ CrashTestBehavior.h"
#include "Behaviors/Demos/FreezeTestBehavior.h"
#include "Behaviors/Demos/RelaxBehavior.h"
#include "Behaviors/Demos/WorldStateVelDaemon.h"
#include "Behaviors/Demos/CameraBehavior.h"
#include "Behaviors/Demos/MotionStressTestBehavior.h"
#include "Behaviors/Demos/ASCIIVisionBehavior.h"
#include "Behaviors/Nodes/TailWagNode.h"
#include "Shared/WorldState.h"
#include "Shared/ERS210Info.h"

ControlBase*
StartupBehavior::SetupBackgroundBehaviors() {
    addItem(new ControlBase("Background Behaviors","Background daemons and monitors"));
    startSubMenu();
    {
        addItem(new BehaviorSwitchControl<SimpleChaseBallBehavior>("Simple Chase Ball",false));
        addItem(new BehaviorSwitchControl<StareAtBallBehavior>("Stare at Ball",false));
        addItem(new BehaviorSwitchControl<HeadLevelBehavior>("Head Level",false));
        if(state->robotDesign & WorldState::ERS220Mask)
            addItem(new BehaviorSwitchControl<ToggleHeadLightBehavior>("Toggle Head Light",false));
        addItem(new BehaviorSwitchControl<TailWagNode>("Wag Tail",false));
        addItem(new BehaviorSwitchControl<RelaxBehavior>("Relax",false));
        addItem(new BehaviorSwitchControl<CameraBehavior>("Camera",false));
        addItem(new BehaviorSwitchControl<ASCIIVisionBehavior>("ASCIIVision",false));
        addItem(new ControlBase("Debugging Tests","Stress tests"));
        startSubMenu();
        {
            addItem(new BehaviorSwitchControl<MotionStressTestBehavior>("Motion Stress Test",false));
            addItem(new BehaviorSwitchControl<CrashTestBehavior>("Crash Test",false));
            addItem(new BehaviorSwitchControl<FreezeTestBehavior>("Freeze Test",false));
        }
        endSubMenu();
    }
    addItem(new ControlBase("System Daemons","Provide some common sensor or event processing"));
    startSubMenu();
    {

addItem((new BehaviorSwitchControl<AutoGetupBehavior>
("Auto Getup",false)));
addItem((new BehaviorSwitchControl<WorldStateVelDaemon>
("World State Vel Daemon",false))->start());
addItem((new BehaviorSwitchControl<BatteryMonitorBehavior>
("Battery Monitor",false))->start());
}
endSubMenu();
}
return endSubMenu();
}

C wlanconf.txt

This file defines the wireless network settings for the Aibo. The file seen here is the default setup, but it can of course be modified, as long as the computer’s settings are updated.

# Make this your "human readable" name
# This only really works if you have a DNS server also configured with this name
HOSTNAME=AIBO

# The name of the Wireless network to join
ESSID=AIBONET

# Do you want encryption?
WEPENABLE=0
WEPKEY=AIBO2

# APMODE=0 - "ad-hoc demo mode" (peer-to-peer)
#     =1 - "infrastructure mode" (access point)
#     =2 - "auto-detect" (defaults to infrastructure if both are found)
APMODE=2

# Channel of wireless network - is only needed with APMODE=0
CHANNEL=3

# Dynamic Host Configuration Protocol - automatically gets IP address
# from DHCP server. Note that you will need either a DNS
# server or a
# static IP from the DHCP server or you won’t know what IP
# address the
# Aibo is running under
USE_DHCP=0

# If you don’t have DHCP, you’ll need to set these manually
ETHER_IP=10.0.1.100
ETHER_NETMASK=255.255.255.0
IP_GATEWAY=10.0.1.1

D  object.cfg
This tells AperiOS what objects it should run.

/MS/OPEN-R/MW/OBJS/MAINOBJ.BIN
/MS/OPEN-R/MW/OBJS/MOTOOBJ.BIN
/MS/OPEN-R/MW/OBJS/TINYFTPD.BIN
/MS/OPEN-R/MW/OBJS/SNDPLAY.BIN
/MS/OPEN-R/SYSTEM/OBJS/TCPGW.BIN

E  connect.cfg
This tells AperiOS which Open-R objects that intercommunicate, so it can
pass the messages back and forth.

#System->Main
OVirtualRobotComm.Sensor.OSensorFrameVectorData.S
MainObj.SensorFrame.OSensorFrameVectorData.O
MainObj.Image.OFbkImageVectorData.O
OVirtualRobotAudioComm.Mic.OSoundVectorData.S
MainObj.Mic.OSoundVectorData.O

#Main<->Motion
MainObj.RegisterWorldState.WorldState.S
MotoObj.ReceiveWorldState.WorldState.O
MotoObj.RegisterMotionManager.MotionManager.S
MainObj.ReceiveMotionManager.MotionManager.O
#MotionManagerMsgs
MainObj.MotionManagerComm.MotionManagerMsg.S
MotoObj.MotionManagerComm.MotionManagerMsg.O
MotoObj.MotionManagerComm.MotionManagerMsg.S
MainObj.MotionManagerComm.MotionManagerMsg.O

#Motion (+Main’s Ears) -> System
MainObj.MoveJoint.OCommandVectorData.S
OVirtualRobotComm.Effector.OCommandVectorData.O
MotoObj.MoveJoint.OCommandVectorData.S
OVirtualRobotComm.Effector.OCommandVectorData.O

#SoundPlay -> System
SoundPlay.Speaker.OSoundVectorData.S
OVirtualRobotAudioComm.Speaker.OSoundVectorData.O

#SoundPlay -> Main, Motion
SoundPlay.RegisterSoundManager.SoundManager.S
MainObj.ReceiveSoundManager.SoundManager.O
SoundPlay.RegisterSoundManager.SoundManager.S
MotoObj.ReceiveSoundManager.SoundManager.O

#SoundManagerMsgs
MainObj.SoundManagerComm.SoundManagerMsg.S
SoundPlay.SoundManagerComm.SoundManagerMsg.O
MotoObj.SoundManagerComm.SoundManagerMsg.S
SoundPlay.SoundManagerComm.SoundManagerMsg.O

#EventTranslatorQueue (SoundPlay, Motion -> Main)
MainObj.RegisterEventTranslatorQueue.EventTranslatorQueue.S
MainObj.RegisterEventTranslatorQueue.EventTranslatorQueue.S

#RemoteOPENR
MainObj.ROPENRSendString.char.S
TCPGateway.ROPENRSendString.char.O
TCPGateway.ROPENRRReceiveString.char.S
MainObj.ROPENRRReceiveString.char.O
The Tekkotsu is a framework for easier application development on the Aibo, which also comes with a graphical user interface looking like this.