AIBO Gestures

Liesbeth van der Feen Eelke van Foeken

31st August 2004

1 Introduction

Humans do not have many problems reaching for an object. The way they do this is with internally calculating what movement they will make. To have robots make the same movements, they will also have to calculate the angles in which they have to move their limbs. Our project is part of another project on imitation. The goal of this project is to have two AIBOs imitate each other. One of the AIBOs makes a gesture with his front leg, the other robot sees this and will also make this movement. Our part in this project is having the AIBO make gestures. There already is a lot of literature on movement of limbs, as well as in humans as in robots. The main difference an AIBO has compared to a human is that a human has many of degrees of freedom to move his arms. The AIBO on the contrary, has just as many degrees of freedom as there are dimensions in the space it moves in. This makes the mathematics being used quite a bit simpler as when there are only three degrees of freedom in a 3D space, there is only one way a movement can be made. When there are more degrees of freedom, there are several ways in which a movement can be made. There are many more differences. The question is to what degree it is possible to have the AIBO make a natural movement. There are several ways to move an arm. As we want the arm to move as realistic as possible, we will use the mass-spring model. This models the movement of an arm in a sinus like way. This is of course just one part of natural movement. You will see that the physical and mental analog between human or animal arm movement and AIBO physics is low. We will try to have the end-effector (or terminal device, the end of the lower leg which is the point directly related to the goal of the performed action) move in a straight line from the begin location to the end location. In between we will simulate, as mentioned before, a mass-spring system to calculate the points on that virtual line. The distance between these points is constant in time, but not in space. As the only way to move the front leg of an AIBO is to give it the number of degrees it should move a limb, we will use inverse kinematics to calculate these angles. Inverse kinematics is the mathematics to calculate, given a point in space, what the angles of the different joints are. First there will be a paragraph of associated literature, where there are resemblances and differences between the different methods and our methods and between the AIBO and natural movement. Then the simulation and the theory will be
explained. After this the implementation, problems and results are covered. At last there will be a conclusion and discussion.

2 Literature

In an article of animating human motion [3], Seyoon Tak and Hyeong-seok Ko have tried to do this by using inverse kinematics and imitating an example motion. This example motion consists of pre-measured data. In our project we use inverse kinematics to calculate the angle positions of every joint at every frame given the end-effector positions. This gives at most two solutions. A human is far more complex and has many more degrees of freedom than only three and has an infinite number of solutions. The conventional inverse kinematics would produce infinite solutions, which is not desirable. It could produce jerky movements. In [3] they try to imitate an example motion with the goal to produce a unique solution. They try to minimize the gap between the goal and the current end-effector position and imitate the original motion at the same time. Motion capture is a technique to measure and copy complex motion. The problem with this technique is that the motion data that is captured is a specific motion of a specific subject. The anthropometric scale (the size of the limbs) between the measured subject and the animated object will be different. The anthropometric scale of a real dog is very different from that of an AIBO. When two AIBOs copy each others motion this is no problem, because the links are the same, you just have to copy the angles to satisfy the end-effector position. Also, the target motion to be animated might be different from the measured motion. In [3] they combine inverse kinematics with motion capture to make the movement as optimal as possible, minimizing the shortcomings of both by combining. A set of motion data is used as an example to imitate the example configuration at the moment. First an objective function is used to drive the inverse kinematics to imitate the example configuration at the moment. This objective function will be minimized in order to minimize the gap between the goal and current end-effector position and imitate the original motion mentioned earlier. Thus there will be two soft constraints: The end-effector constraint and the joint angle imitation goal constraint. A discrepancy in either will be penalized. The ratio between them can be controlled by giving different weights. The results are that the end-effector error can be reduced to a negligible level by controlling the weights.

In [2] Koichi Kondo uses another way of calculating the orientation of a human arm, focusing mainly on human factors of products. Computer-aided design and concurrent engineering are applications for which their model is useful. It could be used for evaluating the design of a product in its usability and maintainability. Or it could be useful in simulating the use, by a human, of a product. Therefore there has to be a natural movement of the arm. The movements in the tasks are mostly specified in object trajectories. Kondo has made a model that produces postures and motions of arms by looking at the description of the product, for
example the manipulation of the dials of an operation panel. By looking at the geometry, positions and orientations of the objects an appropriate grasp of the hand will be selected. Next the position and orientation of the hand will be selected, hence the corresponding arm posture will be selected. Because there are more than three degrees of freedom there is no unique solution. The only thing we use is kinematics. In the medical world it is widely agreed that arm movement is represented kinematically and that dynamics is a matter of post-processing. The interesting part of the model is the inverse kinematics algorithm that is based on a sensory transformation model developed recently in neurophysiologic experiments. This model describes how visual information of a target location is transformed into the appropriate arm posture in the human brain. The brain determines the arm posture almost independently of the wrist posture. First the arm posture is determined by the object location, and then the wrist posture is determined by the object orientation and arm posture. The visual information about the target is first transformed into some spherical coordinates centered about the shoulder, hence the arm posture is roughly determined. This is only for static postures but they have also verified that it is also applicable to motions.

In our project the focus is on making gestures by robots. Nowadays, robotic research in modeling animal movements has progressed slowly. In [1] Brian Adams theorizes about the discrepancy between animal movement (muscles and a chemical energy source) and robotic movement (motors and electricity) and comes forward with biological-inspired ideas on how to simulate real muscle control and movement. A reason why this discrepancy is so great might be the functionalist approach most motion control systems take. This approach studies the physical system of a robot and creates a control structure that manipulates the properties of the system. So this approach doesn’t take direct notice of biological properties, hence it doesn’t make it a primary goal to copy the working of real muscles and energy consumption. To make the biologically inspired approach work, there must be complete understanding of how the animal motion control system works. Then the question arises of how to model this in robots or simulations. Adams tries, despite the incomplete model of the motion control system, to expand the robotic motion control system by use of those things that are known. What we really implement in our project onto the AIBO is focused on manipulating the environment, the desired state of the earlier mentioned end-effector. By using transforms and inverse kinematics we can calculate the orientation and angles of the limbs and joints given a desired position of the end-effector. The little biological influence expresses itself in that human or animal motor planning is done in part by minimizing the cost associated with certain uncomfortable joint angles during a trajectory. This method is for example used for the humanoid robot DB in the Kawato Dynamic Brain Project [1] that successfully produced drumming and dancing. Our method which is used in more projects is successful from a task-execution standpoint, but has probably a low analog with the biological motion structure. The biggest
difference is that movement doesn’t come from simple joints but from muscles that are attached to the limbs. Those muscles sometimes span across several joints, thus causing multiple joint movement by one muscle. What we also use in our project, to make the movement natural, is a simple spring-based control architecture just like the Humanoid Robotics Group at MIT, but not in all the joints but only in the abstracted movement of the end-effector from one point to another point. This approach makes the movement a little bit natural but the sensory information from the arm is less complex (only corners) compared to human muscles. There is also no memory of the past. To learn novel, humanoid gestures this is necessary. Adams mentions more shortcomings which we will not mention. We will now focus on the solutions which are mentioned. In [1] first a model is made for the energy consumption of muscles. This model will help in two ways. First, there is greater equivalency between the robot and those interacting with it. When learning a task a robot has to know why it went wrong, and in case of mounting to heavy objects the robot should know it is because of too much energy consumption, or fatigue. Second, limitations that are imposed on this model will help the robot develop along human lines. The energy metabolism model will prevent virtual muscles to develop in a superhuman way (no energy loss) and thus provides a way to develop human boundaries on learning new gestures. They have implemented such a system on Cog called meso. It simulates the behavior of the major organs involved in energy production. The more torque, the more energy the metabolic system uses from the organs. Chemical levels of, for example, glucose and also different levels at different points in the human energy metabolism, such as glycogen and fatty tissue which is more general, are simulated. This system displays the differing level of chemical levels during a gesture, which is different in a short strong movement than in a slower and lengthier movement, and can also select one gesture being more efficient than the other. As mentioned earlier it prevents the robot from making superhuman gestures, and thus learning the humanoid limits on making gestures.

3 Methods

To compile and test programs directly on an AIBO is very slow. To test our calculations we will write a simulation first. For this simulation we will write three programs. The first program will calculate the inverse kinematics, the second program will calculate the forward kinematics and the last program will calculate the spring movement. After writing this simulation, we will convert the algorithms to the AIBO. When everything is installed on the AIBO we will first have the AIBO make a gesture with one of its paws. When this works, we will try to have the AIBO make a gesture with both of its paws. Once this all works, we will implement some standard gestures and write an interface for easy usage of the gesture class.
4 Simulation

As mentioned in the previous section, the simulation consists of three separate programs. The first programs calculate the forward kinematics given three angles. The second program calculates the inverse kinematics given three coordinates. The third program calculates the different positions the leg has to go per 16 frames, because in the AIBO-program there is a constant called CommandVectorFrames which is 16. The AIBO has a standard for the number of frames it executes per second. This number is adaptable, but we will first use standard of 16. In the following paragraphs we will give a description of the three different programs and the mathematical origin of the algorithms used in the program. First we will describe the program which calculates all the points the paw has to pass to get to its destination. Then we will give a description of the forward kinematics and last we will give a description of the inverse kinematics.

Spring  We want the end-effector to move in a straight line from begin to endpoint. Also, to make it a natural movement we will use the spring-theory. A spring can be pushed in and stretched out. The force associated with the pushing or stretching the spring is dependent off the mass of the object, the stiffness and damping of the spring. In simple form the formula is:

\[ F = kx \]

where \( k \) is the spring-constant or damping, and \( x \) is the distance from equilibrium. We want to work with velocity and acceleration as well. So we make it a differential equation.

\[ F = ma = m \frac{d^2x}{dt^2} = -kx - c \frac{dx}{dt} \]

\( m \) for mass, \( k \) for damping, \( c \) for stiffness. For convenience we say that \( k = 2 \times mass \times y \), \( y \) is the damping. In the beginning we have an \( x \) that is the same as the distance to cross and a \( v \) of 0. We know everything except the acceleration. So to calculate the acceleration at one time we have the equation

\[ a(t) = \frac{(-kx(t) - cv(t))}{m} \]

We want to calculate the velocity at any time step within the movement. So we have to make it discrete. The only thing we have to do is calculate the \( x \) on every time step and the velocity at every time step, given the acceleration at the last time step until the end position has been reached. In formula it looks like this:

\[ x(t) = x(t - dt) + dtv(t - dt) \]

to calculate the next position. \( x(t-dt) \) is the last position plus the last speed times the delta \( t \). The tricky bit (relatively, because it is very simple) is the last
speed. Because off the acceleration the next speed will be different than the last speed. So to only use the last speed would not be sufficient. At \( t = 1 \), the speed at \( t = 0 \), the time step before, will be 0, so the position at \( t = 1 \) consequently will also be 0. The arm is accelerating, so the speed is not constant over a given time step. So we have to calculate the average speed over the time step. We do this by simply adding the speed to the current time with the speed one time step before, hence dividing this by two to middle:

\[
\frac{(v(t) + v(t - dt))}{2}
\]

To calculate the speed we use this equation:

\[
v(t) = v(t - dt) + dt \cdot a(t - dt)
\]

With these equations we iterate over every time step until the end-effector has approached the target close enough. Then we know the movements it must take, so the AIBO knows the moves (positions) it must make to reach its target. The movement from one point to another is covered in the same time, so the larger the difference the larger the speed. We want the simulated spring of the arm to make a weak damping. This means that the spring damps not all the way but it slightly oscillates around the zero-position, like a sinc function, to zero. This seems natural, because if you move your arm to a certain position you will notice that you will not reach that position directly but that you will pass that point a little and correct accordingly. We think this also corresponds with a ballistic movement of a human arm, hence a correction to the position. The ballistic movement is the gross movement. Also we want the gesture to first accelerate and then decrease its speed towards the end. By testing we will find out which values for mass, stiffness and damping are right.

We have made a simulation of this formula and we printed out some results of the positions on the line. The first one shows weak damping with large am-
plitude, which is not good because the movement would be very jerky. The last one has low weak damping causing a little movement around the equilibrium. That is what we want. The pink one doesn’t oscillate around the end position at all, but moves slowly to the end position.

**Forward kinematics** Forward kinematics calculates the (x,y,z)-coordinates given the angles. The z-coordinate is the distance to the side. This can be calculated with a simple sine function, where the length of one side is \((l_2\cos\gamma + l_1)\) and the angle is \(\beta\). You get the following function:

\[
z = \sin\beta (l_2\cos\gamma + l_1)
\]

Coordinates x and y are dependant on all three angles. For these coordinates you get the following functions:

\[
x = -l_2\cos\alpha\sin\gamma + \sin\alpha\cos\beta (l_2\cos\gamma + l_1)
\]

\[
y = \cos\alpha\cos\beta (l_2\cos\gamma + l_1) + l_2\sin\alpha\sin\gamma
\]

![Figure 1: Direction of the axis](image)

**Inverse kinematics** To calculate the inverse kinematics we used the formulas from the forward kinematics. As all the formulas contain two or more unknown variables, we needed more than just these formulas. The angle \(\gamma\) can easily be calculated with the law of cosines.

\[
\cos C = \frac{a^2 + b^2 - c^2}{2ab}
\]

Once \(\gamma\) is known, \(\beta\) can easily be calculated with the following formula:

\[
\beta = \arcsin\left(\frac{z}{l_2\cos\gamma + l_1}\right)
\]

Then \(\alpha\) can be calculated with
\[ \alpha = \text{atan} \frac{a}{b} + \text{atan} \sqrt{\frac{a^2 + b^2 - c^2}{c}} \]

where \( a = l_1 \sin \gamma \), \( b = \cos \beta (l_2 \cos \gamma + l_1) \) and \( c = y \).

5 Implementatie

The last stage of our research is to implement the algorithms from the simulation onto the AIBO. We will first give a short explanation of the AIBO software. The AIBO is programmed in the OPEN-R environment. We used a standard object to program our algorithms in. We first wrote a function that made the AIBO sit on its hind legs, so it could easily make gestures. If you don’t want to make any more gestures the AIBO neatly sits down. Then we implemented our simulation. The first problem we encountered was that the AIBO was not able to sit on its hind legs. It did not seem to have enough strength. Therefore we changed the gain values. The second problem was, that the OPEN-R environment does look like a C++ environment, but is not exactly the same. When a program works well as a C++ program, it can still give errors when transferred to an OPEN-R program. Unfortunately it takes a lot of time to debug an OPEN-R program. In the next sections we will describe some of the problems we have encountered while transferring to OPEN-R. First we will give a short explanation of the different functions we have used.

5.1 Function

**Interface** The interface is located in the state GESTURE. When the AIBO sits in a position where he can move his arm most freely, the user firstly is asked what leg must be moved. Then the user can change the parameters mass, stiffness and damping of the virtual mass-spring, in order to check different configurations. Then the question is asked whether the user wants to set the position to be reached. If these last two questions are answered negatively, default values are used. Then the associated function Gesture is called, until it
returns false, which implicates the gesture has finished. Next the state is being changed to FORE_STRAIGHT and the question will be asked if the user would like to make another gesture. If so, it moves to its initial position ready to make another GESTURE.

The programming we have done uses state-transitions. This means the AIBO is in a state and the state will change if certain conditions are met. The function Ready is called in a loop, and depending on what state it is in, accordant functions are called. An example is given in the last part. The states that we used are:

- IDLE,
- START,
- ADJUST_DIFF_JOINT_VALUE,
- HEAD_UP,
- FORE_KNEEL,
- FORE_STRETCH,
- RETREAT_LEGS,
- FORE_STRAIGHT,
- GESTURE,
- SIT_DOWN1,
- SIT_DOWN2,
- SIT_DOWN3,
- DONE

At first the IDLE, START and ADJUSTDIFFJOINTVALUE, where the initialization happens, is executed. The next five states are states in which the AIBO moves to its initial position. GESTURE is the main state where the chosen position will be reached.

**getTraject, trajectReady, doTraject** The three functions getTraject, doTraject and trajectReady are respectively to: get the trajec between the current positions (angles) and the given positions(angles) of the joints, first get the current position then calculate the difference between this and the end position and then calculate the number of steps to be taken and how big one step is. The distance of one step is dependant on the MAXSPEED and the frame. One frame is 0.4 degrees per 8 ms. doTraject does one step for all joints. TrajectReady checks if all steps are taken, so whether the movement is finished or not.
getTrajectKinematics, trajectReadyKinematics, doTrajectKinematics

These functions are somewhat different from the previous functions and are meant for the movement of the robot arm with the mass-spring. GetTrajectKinematics does not get the current joint value in order to make the movement faster. To get the joint value at every step takes a lot of time. The angles to cover are calculated beforehand, hence the steps in between and the delta to cover in one step are calculated. The other two do not differ a lot, the only thing to mention is that an index is an extra parameter to indicate a number in a specific array.

Gesture

To avoid confusion, gesture means the whole movement, movement is the movement from one calculated position to another which is composed of one or more steps. These steps have a constant angle per move Gesture is the spine function of the program. The gesture is made here. Here the parameters are the positions to reach and a boolean to decide if the left or right legs will be moved. At the beginning of a gesture the begin position is calculated first with the forward kinematics, given the angles the joints make. After this the angles, which are the angles per joint per step, are calculated. More of this function later. If angles return NULL this indicates an impossible angle has been calculated. As a consequence false is returned causing the gesture to end. After this the current angles of the joints are calculated. This is done with getTraject, which holds the extra ability to get the current angles of the joints. Then with getTrajectKinematics, which does not get the current joint values, the delta and steps are calculated per move. So now there are two arrays which contain the number of steps and the deltas per move. With these arrays the gesture will be made. DoTraject executes one step of a move at a time for all the given joints, this with an angle delta. TrajectReadyKinematics checks if this move is finished. If so it increases index, and the next move will be taken. There is a check whether or not all the moves are made, so if the gesture is finished or not. If so, the state will be changed to FORE_STRAIGHT where the question will be asked if you want to make another gesture or not. Also the boolean first is set on true and the index is set to zero.

Angles

This function is used to create an array with all the angles the arm joints have to pass before they reach their end position. First a list of all the positions is being created and for each of these positions, using inverse kinematics, the corresponding angles are being calculated. If one of the positions turns out to be an impossible position, in which case inverse kinematics returns null, then the array with the angles so far calculated is being deleted and the function returns null.

Positions

The distance between the begin and end location is being calculated using the magnitude. Then the subpositions on this track the AIBO arm has to pass are being calculated with the function PositionsOnLine. These line segments are not all of equal length due to the natural movement we are try-
ing to accomplish. Therefore we needed a separate function to calculate these segments. The array with the subpositions is returned.

**LineSegments** This function is the implementation of the earlier discussed mass-spring model. Given the magnitude it calculates the positions on this virtual line. These positions will be different positions with different values of the mass, stiffness and damping. The loop is ended when the last step is smaller than 1/100 of the total distance.

**PositionsOnLine** To convert the 1D positions on the line to 3D positions, which can be used to calculate angles, this function is used. First the vector of the line in the 3D plane is calculated. Then per x, y and z the positions are calculated with this function

$$P[j][k] = bP[k] + (m - lS[j])/m * v[k])$$

Where $P[j][k]$ is the 3D position at move $j$ and $k$ is for x, y or z. $bP[k]$ is the beginPosition at x, y, or z. $m$ is the magnitude or distance between begin and endpoint. $lS[j]$ is the 1D position at move $j$. $v[k]$ is the vector of x, y, z.

**ForwardKinematics** As mentioned before this function calculates the coordinates given the angles. This function receives a list of used joint values, depending on if the left leg or the right leg has to be moved. The function also receives an empty list to write the results to. The joint values are read using a standard OPENR function, GetJointValue. GetJointValue returns microradials which are then converted to normal radials. Then the positions are calculated and written to the empty list.

**InverseKinematics** This function calculates the angles given a set of coordinates. Like the ForwardKinematics function this function receives a list with coordinates and an empty list. Using the formulas given earlier, this function calculates the angles and writes them to the empty list.

### 5.2 Problems

While writing our program, we have encountered several problems. We started writing our simulation in Open Dynamics Engine. With this program you can simulate bodies of an object and their dynamics and kinematics in a 3D environment. It took a lot of time to understand this program and then we found out it was not suitable for what we wanted. The reason for this was that in our project we concentrated mainly on the kinematics while in ODE it was necessary to also simulate the dynamics such as friction and torque. So, instead of using ODE, we decided to write a simple text-based C++ program with which we could check our algorithms.
When we implemented our algorithms onto the AIBO we found out that the OPENR environment is not exactly the same as an C++ environment. When compiling our C++ program in OPENR we got several errors, mainly with our array usage and pointers. Unfortunately OPENR programs compile very slowly, which made it very hard to debug our program. Also when reading values with GetJointValue, we found that apparently these values are not very exact. We had used a very broad range to catch all the values, but still some values went out of bound. We could not find out why. Another problem was that there is a maximum speed the joints can handle. There is also a frame with a period of 8 ms. Because of the mass-spring model the arm was to accelerate and decelerate. Because in these 8 ms the speed is constant, you can not accelerate in a continuous way. Accordingly we have made a spring-mass model which can handle discretion. Because the transition from one move to the next is not completely smooth, the movement becomes less smooth as a whole, certainly when the movements are very small. This forced us to make the moves larger (make less steps to finish the gesture), especially in the middle of the gesture and with large differences to travel. This causes the arm to move at its maximum speed at the larger part of the gesture. Is this a bad or a good consequence? We do not know. In our literature not much was said about how a human arm moves. It could be that an arm continuously accelerates until it almost reaches the end location, or that, what would be good for us, the human arm has a maximum speed in the middle of the movement. Secondly, the larger the movements the larger the differences in speed between the movements. This causes a movement less smooth which is not what you want. Here is a graph of one of those moves. You can see that it quite nicely accelerates in the beginning and decelerates later. We made weak damping here but cut it off when the end-effector came near the end-point. This because the number of steps would be to great then. But you can also see that the velocity increases greatly in the beginning causing the shocky transitions from one move to another and the not so smooth acceleration.
Thirdly, because of the discretion the moves in between points are not in a straight line. If the distance between points is larger the line will be less straight.

Because the transition from one movement to the other is not very smooth this way, we left the idea of weak damping and implemented critical damping. The consequence of this is that the gesture is without oscillation at the end location. We thought that it would be good to speed up the movement, by calculating less per move. It seemed to difficult to calculate all the angles to be used beforehand and put them in some sort of array. The main reason why this is difficult is because of the different sizes of the angles to be used in one gesture. There are scenarios thinkable where one angle is much larger than the other, hence the number of moves to make is much larger. Then this angle will be finished earlier than the other. This is not what we want. The angles must be covered in the same amount of time. It was too difficult to remember all the angles per step. What we did was remember the steps and delta per move. Then per move the next angle is the previous angle plus the given delta. This way the move was reasonably fast.

The leg is moved by choosing a position to reach. The problem is, how can you show which positions are reachable? The way we implemented it you can choose a position and there will be a warning if you cannot reach the target. It is very difficult and maybe impossible to show in a neat way which targets you can reach or not. Another problem was the time. We just did not have enough time, mainly because of the endless debug-phase, to make good tests.

6 Results

When testing everything on the AIBO it worked reasonably well. When entering some coordinates, the AIBO could move its paw to the right position. However when entering certain coordinates, the hind legs could be in the way of the movement. Also on some occasions the paw and the AIBO itself as well trembled, which is quite unnatural. Another problem was that the AIBO didn’t seem to move very natural, it still looked like a robot movement. The acceleration is not visible enough.

The basic values we have used for damping, stiffness and mass in the mass-spring system were respectively 0.7, 10 and 10. When changing the damping value to some lower value, for example 0.1, the movement was faster. This causes weak damping which should entail the end-effector oscillating vigorously around equilibrium. Because of this when nearing the end-position we forced it to stop. Here the graph of the velocity at these values:

When changing it to a higher value, the movement was very slow. Changing the mass value had pretty much the same effects as changing the damping value. When giving the stiffness a low value, the movement becomes very slow and not smooth. With these numbers and especially the one with damping 0.1 , the
velocity in the beginning (see the last graph in the paragraph: Simulation) is high. Therefore we tried some other values where this would neater. The best values we could find were causing low weak damping and made the limb slow down when nearing the end position. These values were: damping 0.4, stiffness 0.05, mass 0.1. See the graph 'Speed movement'. However, when giving the stiffness a high value, the movement becomes fast, but with distinct steps, where the movement within the steps is very smooth. The whole movement looks a bit jumpy.

7 Conclusion

Our original question was to what degree it is possible have let the AIBO make a natural movement. We have found that it is very hard to have the AIBO make a natural movement. Firstly, the AIBO is made from a task-execution viewpoint, so the builders never tried to make the physics of the AIBO natural. No muscles or nerve fibres. At this point we can conclude that the AIBO fails miserable. Then what remains is the question in what degree is it possible to make it look like a natural movement? Our final movement is not as natural as we would have wanted it to be. Several reason for this can be given. First the movement the AIBO makes is very small. Humans have long arms where it is easier to see an acceleration. AIBOs however have a very small reach and it is therefore harder to see an acceleration in the movement. This could also be the reason why the movement did not look very natural. Another reason why the
movement does not look natural is the appearance of the AIBO. It looks very much like a robot, so we expect it to move in a robotic way, especially as the sounds it makes are very mechanical as well.

8 Discussion

While we have managed to have the AIBO make a gesture, there is still a lot of improvement possible. On some movements the AIBO tends to be in its own way. Humans can easily solve this problem by recalculating the path to the end location or by moving the other limb that is in the way, out of the way.

This is probably possible when using an AIBO, because the sizes of the parts of the AIBO are known. With this knowledge and the angles the different parts make, it is possible (maybe not in the way humans do) to avoid hitting itself. When calculating the positions there can be a check if something is hit. If so, the track can be recalculating or a different gesture could be made. Another option that we have not explored, but that we would have wanted to explore is to move both arms at the same time.

We used a spring-mass system on the virtual straight line in between begin and end position. What might have been more natural, as mentioned in [1], is simulating a spring-mass system in every joint. Muscles seem to have elasticity, so this way would have been better. As suggested in the literature the AIBO physically, recall the functionalist approach, does not look like a real animal at all. From a task-execution viewpoint the theory is good but it is a long shot to make the movement physically natural. What could be simulated in the AIBO is an earlier mentioned energy metabolism system. There is still a lot to explore and the question whether it is possible to have the AIBO make a natural movement is still not completely answered. However, we tend to say no.

References

