2nd International Conference on Autonomous Robots and Agents
December 13-15, 2004 Palmerston North, New Zealand

Evolving Autonomous Biped Control from Simulation to Reality

Adrian Boeing, Stephen Hanham, Thomas Bräunl
Mobile Robot Lab - CIIPS
The University of Western Australia, Perth
http://robotics.ee.uwa.edu.au {boeing-aj, tb}@ee.uwa.edu.au

Abstract

Transferring an evolved control system from a simulated environment to the physical world poses a number of challenges. The difficulty in accomplishing such a task increases the complexity of the system that is being simulated and controlled increases. One of the most challenging control tasks is to generate a stable walking gait for a bipedal robot. This article describes a method in which a simulated control system for a small humanoid robot is evolved and transferred to robot hardware.

Keywords: evolve, dynamic, simulation, biped, spline, servo, reality gap

1 Introduction

Evolving control systems for robot locomotion is becoming a standard approach for the generation of improved or newer control systems for robots [1, 2, 3, 4, 5]. There have been a number of successful demonstrations of legged robot control being transferred from a simulated environment to a physical environment. Satisfactory results for quadruped**,** hexapod and octopod robots have been obtained [1, 2, 6], however results for bipedal robots have not been generally satisfactory [3], often resulting in shuffling movements rather than walking motions. This article presents a simple control system and describes a method to overcome some of the difficulties encountered when making the transition from a simulated world into the real world.

If a physically accurate simulation model can be constructed for a robot, then a number of advantages for robot development present themselves. Physical simulation of the robot allows a robot designer to prototype and visualize a robots design without requiring physical construction. The simulation also simplifies the task of evolving a control system for the robot. Thus, the workload on the designer is reduced, and robot walking motions can be tested and optimized at an early design stage allowing the designer to modify the design, if necessary.

2 Target Hardware

The target hardware for the controller is a small humanoid robot called Andy (see Figure 1) [7]. Cost and weight were important design considerations in Andy's development. As a result, Andy stands approximately 350mm tall, and weighs around 1400g. Andy has 10 degrees of freedom in his legs, and each joint is powered by a Hi-Tec 945 MG servo [8]. The Hi-Tec servo specifications are listed in Table 1. Links are made from 3mm thick aluminum flat plate and are used to connect the plastic shafts of the servos directly to the next link. These connections result in a substantial amount of inherent flexibility.

Andy can be equipped with a number of sensors, including a color camera, PSDs, inclinometers, gyroscopes and pressure sensors. The pressure sensors are permanently mounted as Andy's feet, which are constructed from three metal "toes". Each toe has two strain gauges that are used to produce a voltage in proportion to the applied force.

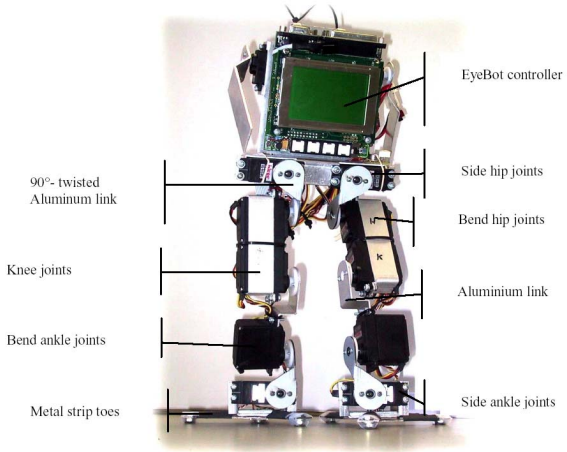

Figure 1: Andy Droid Robot

The biped's processing requirements are provided by an EyeBot MK3 controller, an inexpensive but powerful platform [7]. The controller is based on a 25 MHz 32 bit Motorola 68332 chip, has an LCD display, four buttons, parallel and serial ports, as well as 8 digital inputs and outputs and 8 additional analog inputs.

Parameter	Specification
Operating Voltage	4.8 V
Stall Torque	11 kg*cm
Deadband Width	4 µs
Operating Speed	0.16 s / 60° (no load)
Operating Angle	45° / 400 µs

Table 1: Servo Specifications

3 Simulation

3.1 Mechanical Simulation

There are a number of advantages in using a simulated environment to evolve robot controllers. Simulations eliminate the risk of damaging robot hardware and other hardware related concerns, such as battery power and temperature effects. Simulation also provides the added convenience and freedom to manipulate any force or environment variable to suit the situation. This aids greatly in general experimentation, and in resetting the robot to an identical initial position for each evolution trial. Simulations typically execute faster than control programs executed on robot hardware, and information from the motions is easier to extract. These factors make simulations an attractive option.

The simulation tool used to simulate the robot was the freely available Dynamechs library [9]. Dynamechs is an efficient rigid-body dynamic simulation library that is based on the Articulated Body Algorithm (ABA) developed by Featherstone [10]. The robot's structure is defined using multiple chains, starting with a mobile base with each link described in terms of the previous link using modified Denavit-Hartenberg parameters [11].

The ABA is based on the observation that the accelerations of bodies in a rigid-body system are always linear functions of the applied forces [10]. Initially the velocities of each joint are calculated by working from the base link to the terminal links. The Articulated Body Inertia (ABI) matrix can then be calculated by traversing back from the terminal links to the base link. Finally the accelerations of the bodies are calculated using Equation 1.

$$f = I^A a + p^A \quad (1)$$

Equation 1 - ABA [10]

Where:

f - Force
I^A - ABI
p^A -Bias Force (force required to bring acceleration to zero)

Extensions to the simulation package were made to include the fitness evaluation functions required to operate the genetic algorithm, as well as the simulated servo and sensor models. The simulated control system was also added to the package.

3.2 Robot Model

A schematic of the robots legs are illustrated in Figure 2. The robot model required by Dynamechs was constructed using the RobotBuilder [12] package. For each link, Dynamechs requires information including its relative position and orientation, mass, center of gravity and inertia matrix.

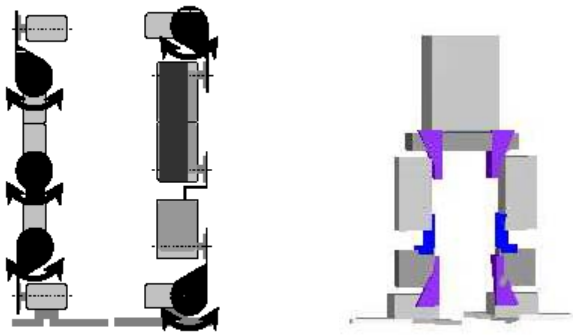

Figure 2: Leg Schematics and Simulator Model

RobotModeler (part of the RobotBuilder package) allows the use of primitive shapes such as cubes and spheres to approximate the physical shape of each link and subsequently allows calculation of the inertia matrix of each link. The center of gravity was estimated using a similar method.

To model the inherent flexibility in Andy's toes, an extra joint was added to each toe. The flexibility in Andy's toes is a result of the steel springs that are used for pressure sensing. The flexibility was replicated using a rotational joint with very small joint limits and a large friction value. Once the joint moves outside its limits, a spring restoring force is applied, mimicking the memory effect of the steel spring.

3.3 Servo Model

An accurate model of the torque produced by the servomotors is critical if the walking gaits produced by the mechanical simulator are to provide a reasonable approximation to the gaits produced in the real world. A servomotor comprises essentially two components, a control system, and a DC motor. The DC motor torque can be mathematically modeled using the standard armature controlled DC motor model [13], represented by (2). The control system for a servo is generally a Proportional-Integral-Derivative (PID) controller. The controller can be mathematically simplified by ignoring the integral and derivative terms, since the proportional term dominates its behavior. Incorporating the DC motor model into the servo P-controller and using $\theta_\varepsilon = \theta_{output} - \theta_{input}$ for the error signal gives (3).

Considering the case where the armature is stationary ($\omega_n = 0$) and the maximum supply voltage is applied to the armature ($V_a = 4.8V$), allows us to determine:

$$T_A(t) = K_T \frac{V_a(t) - K_b \omega_n(t)}{R_a} \quad (2)$$

Equation 2 - Armature Controlled DC Motor Torque Equation [13]

$$T_A = \frac{N.K_T}{R_a}(K_a K_\varepsilon \theta_\varepsilon(t) - K_b \omega_n(t)) \quad (3)$$

Equation 3 - Torque Equation for Servo Control System [14]

Where:

N – Gear reduction
R_a – Armature resistance (ohms)
K_T – Motor torque constant (Nm/A)
K_a – Power amplifier gain
K_b – Motor back EMF constant (Vs/rad)
K_ε – Proportional gain for error signal
ω_n – Angular velocity of motor (rad/s)
V_a – Applied armature voltage (Volts)
θ_ε – Angular error signal ($= \theta_{output} - \theta_{input}$)

$$\frac{N.K_T}{R_A} = \frac{T_{A,stall}}{V_{a,\max}} = 0.180 \text{ N.m/V} \quad (4)$$

Equation 4 - Stall Torque Test

When the motor is at top speed the applied armature voltage equals the back EMF, i.e. $V_a(t) = K_e * w_{n,max}(t)$. From the servo specifications we know the maximum angular velocity, allowing us to solve the motor back EMF constant:

$$K_e = \frac{V_{a,\max}}{w_{n,\max}} = 0.733 V.s/rad \quad (5)$$

Equation 5 - Maximum Speed Test

This gives:

$$T_A = 0.180(V_a(t) - 0.733.w_n(t)) \quad (6)$$

Equation 6 - Servo Torque Equation

The proportional component of the controller was modelled with (7). The model assumes that the maximum supply voltage is applied to the motor until it gets within a tolerance of the desired angle. The voltage applied to the motor is then linearly decreased until the servo reaches its final destination.

$$V_a = \begin{cases} K_\varepsilon \theta_\varepsilon(t) & , K_\varepsilon \theta_\varepsilon(t) < V_{a,\max} \\ V_{a,\max} & , \text{otherwise} \end{cases} \quad (7)$$

Equation 7 - Armature Voltage P-Controller Model

This model performed adequately for large movements, however, it was found that for small angle movements, where the maximum armature voltage was not achieved, the servo model was not accurate since the full stall torque is not applied. To overcome this, it is assumed that the maximum supply voltage is always applied to the armature. This is a reasonable assumption since the slowing down of the servo has only a minor effect on its time response.

The deadband specification of the servo was used to decide when the servo model had reached its target angle. Once the servo is decreed to have reached its destination, a torque is no longer applied to the joint. This is shown in (8).

$$T = \begin{cases} 0, & |\theta_{\text{current}} - \theta_{\text{target}}| < \frac{\theta_{deadband}}{2} \\ T_A, & otherwise \end{cases} \quad (8)$$

Equation 8 – Servo Deadband Model

4 Control System

A spline based control system is responsible for manipulating the servo inputs [4]. The spline controller comprises of a set of connected Hermite splines. Each spline can be defined by a variable number of control points allowing variable degrees of freedom. The function used to interpolate the control points, given starting point p1, ending point p2, tangent values t1 and t2, and interpolation point s, is shown below:

$$f(s) = h1.p1 + h2.p2 + h3.t1 + h4.t2 \quad (9)$$

Equation 9 – Hermite Spline

Where:

$h1 = 2s3 - 3s2 + 1$
$h2 = -2s3 + 3s2$
$h3 = s3 - 2s2 + s$
$h4 = s3 - s2$

Three connected splines are combined to form the overall control structure for one servo. The three splines are responsible for three different phases of the robot's walk. The initial phase of the walk is considered to be responsible for moving the robot from a stationary position into the walking motion. The servo inputs for this phase are represented by the start spline. The repeated motions that sustain the walk correspond to the cyclic spline. And finally, the end spline is used to moving the robot safely back to a stationary position.

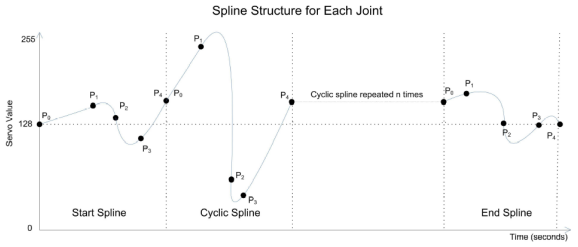

Figure 3: Spline Controller

The advantages of a Hermite spline controller are:

- It can be represented by a compact chromosome which aids the genetic algorithm convergence speed [4]
- Relatively computationally inexpensive and hence can execute comfortably on the EyeBot platform
- Joint positions and velocities are always continuous

The spline controller used to control Andy's movements contained 8 control points for each cycle, and

had a cycle time of three seconds. The end section of the spline controller was discarded, and the start spline contained only 2 control points. Each spline had an associated hardcoded offset value and amplitude modifier. This allowed the masking of the effects of motor wear and enabled some compensation between the small differences between the individual servos mounted on Andy.

5 The Reality Gap

Brooks [15] states that one of the fundamental reasons for avoiding robot simulations is that there is a great danger that the simulations will not match the real world. This difference between the simulated and real world is sometimes referred to as the reality gap. Despite this, there are a number of examples of successful evolution of controllers using simulations.

Various approaches have been tried to enable the seamless transition from simulated controllers to the physical world. These approaches range from generating lookup tables from sensor data obtained in the real world [5], to accurate factory-built simulations supplied with the robot [16]. Jakobi [1] argues that such approaches will always fail to transfer from simulation to reality, as the evolved controllers come to depend on certain aspects that are only present in the simulation, and are not reflected in the physical world. Jakobi's solution to this problem is to evolve controllers with very minimalist assumptions concerning the robot simulation. A similar approach is proposed in this article, in that the genetic algorithm is limited to producing approximate control solutions, as opposed to being allowed to refine solutions to a globally optimized form. This limits the controller from evolving to a form that relies on specific responses only available in the simulation.

6 Genetic Algorithm

6.1 Encoding

Success of evolved spline control systems has already been demonstrated for various legged robot configurations for a wide range of activities from walking to jumping [4]. The spline controller can be directly encoded with each joint's control point parameters encoded using 8 bit fixed-point values.

Typically, all of the control point's parameters are encoded (position in time and output value, and tangent) to enable the complete description of the spline. However, to enforce the approximate solution required to overcome the reality gap, the control point locations are limited. Each control point is forced to be an equal distance from all other control points, and the tangents are forced to be zero (see Figures 3 and 4). Thus only the servo input value at any specific time position is evolved.

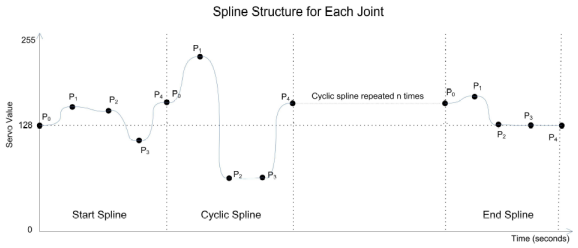

Figure 4: Limited Spline Controller

Limiting the controller's output in this way stops the GA from over optimizing the controller so as to take advantage of specific features only present in the simulation. Essentially it is this aspect that enables the robot to cross the reality gap. This approach also requires less evolution time as the resultant controller chromosome is more compact.

6.2 Genetic Algorithm Configuration

The genetic algorithm employed a simple fitness proportionate selection scheme. The operators implemented were a bitwise mutate, a bitwise crossover, a byte-wise mutate, average, creep. The byte-wise mutate replaced a randomly selected byte with a randomly generated byte value. The byte-wise creep randomly incremented or decremented a byte by 1, and the average operator generated a new chromosome from the byte-wise average of each byte in the two parent chromosomes. The parameter configurations are given in Table 2.

Operator Name	Selection Chance
Bitwise Mutate	10 %
Bitwise Crossover	30 %
Bytewise Mutate	25 %
Bytewise Average	30 %
Bytewise Creep	5 %

Table 2: GA Parameters

6.3 Fitness Function

In order to evaluate the appropriateness of each gait a fitness function is employed, which returns information to the genetic algorithm about the performance of each gait. To evaluate the fitness of each robot, the function takes into consideration the forward distance of the walk, and the average velocity at which the robot's center is lowering [4].

$$fitness = 5 * forwards\_distance - 50 * ave\_vel\_lowering \quad (10)$$

Equation 10 – Fitness Function

In order to decrease the evolution time, a terminating condition was included to the fitness function. Termination would occur if the torso (main reference point of the robot) touched the ground, i.e. the robot fell over.

7 Results

7.1 Servo Model

The simulated servo model was verified against the physical servo responses by attaching an inclinometer to the servos and recording the data generated during movement. Figure 5 and 6 show the outputs of both the simulated servos and the actual servos.

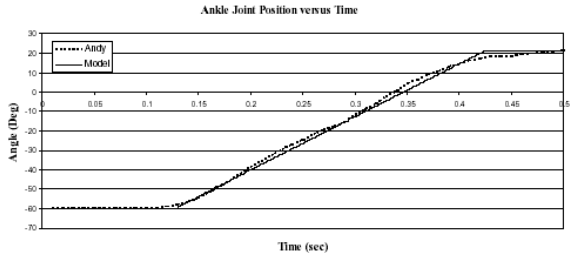

Figure 5: Ankle Joint Response

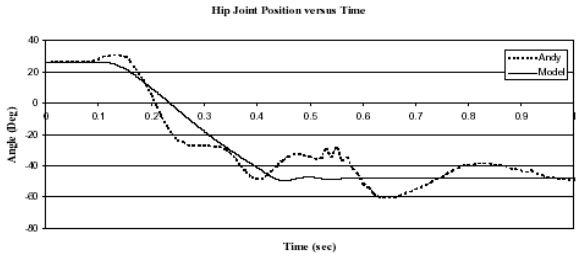

Figure 6: Hip Joint Response

Figure 5 shows a close approximation between the software simulation and Andy for the ankle joint. The ankle joint has only a small amount of mass, mainly the feet, as a load. However, in Figure 6, a larger discrepancy can be seen between the simulated model and the physical response. This is due to the extra load on these joints. The hip joint, in particular, has the entire leg as load. The observed overshoot in these cases is due to several factors. The first factor is due to the overshoot of the PID controller in the servo motor. The second factor is due to the flexibility of the plastic shafts of the servos. Another factor is the reaction torque of the servo inducing vibrations in the robot that tend to affect the inclinometer readings.

7.2 Simulation Results

The parameters of the control system can be extracted from the gene pool in a device independent format. This allows the parameters to be transparently imported into either the simulated program via a file, or downloaded to Andy's control program over a standard RS232 cable. Figure 7 and 8 illustrate the same evolved control program executing on the real and simulated Andy robot.

Figure 7 depicts the simulated robot movements. The robot achieves locomotion by initially pressing downward with its left toes, causing the robot to tilt to its right (Figure 7.2). The robot then drags its left foot along the ground in front of it. The robot then presses downwards with its right toes and lifts its right foot off the ground and places it in front of it. This cycle repeats itself to produce a slow forwards walk.

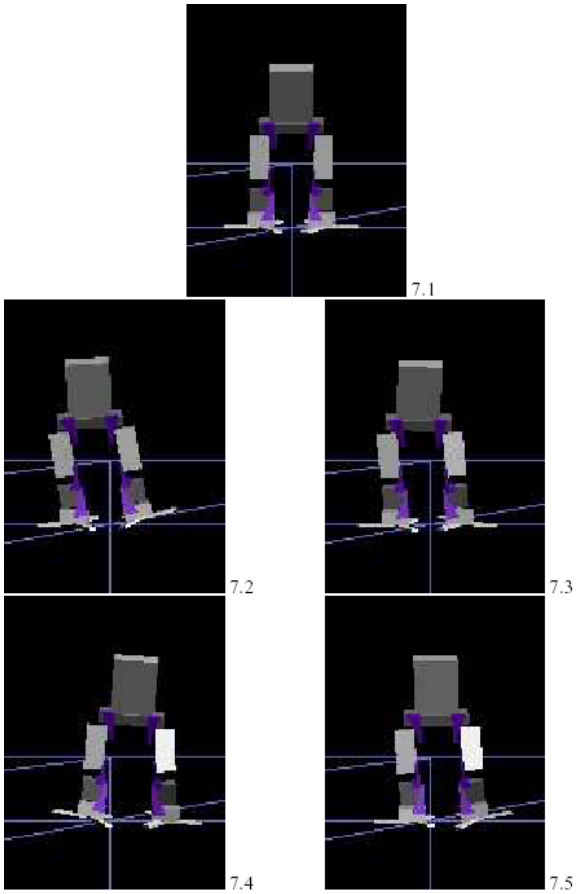

Figure 7: Simulated Walk

7.3 Real Robot Results

The figures show a close mapping between the simulated robot and the physical robot. One significant discrepancy between the simulated and physical walks is illustrated in Figures 7.4 and 8.4. This difference between the simulator and Andy is probably due to worn motors, whose behavior has changed over time.

Whilst the method did result in transferable walking patterns between the simulated robot and the physical robot, the resulting locomotion still performed worse than a good manually designed gait. Furthermore, whilst almost all transferred walks allowed Andy to sustain motion for three or more full walk cycles, few actually resulted in satisfactory forwards motion. A number of factors hampered the performance of Andy's gait, including battery power, servo jitter, the flexibility in the plastic joints, and motor wear.

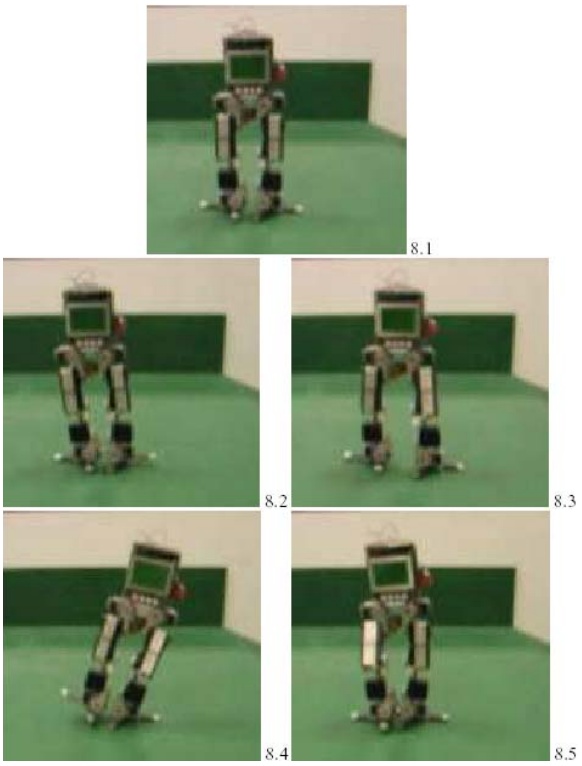

Figure 8: Andy Droid Robot Walking

8 Outlook and Conclusions

The problems observed in this system are believed to arise from the incomplete servo model and the shortcomings of the robot hardware. The evolved walking patterns could potentially improve if the restrictions on the evolved controller were loosened. The servo model could be improved if the internal parameters of the servos PID controller were fully known. This could be achieved by additional in-depth testing of the servo motor. Alternatively, the robot hardware could be redesigned to operate with DC motors with separately designed PID controllers.

Further investigations into the encoding of the spline controller could reveal an optimal configuration enabling further optimizations of the controller, yet still limiting its complexity such as to avoid the problems related with the reality gap. Incorporating sensor feedback into the controller could also increase the robustness of the gait. This should then assist the controller in bridging the reality gap.

Our system has successfully managed to evolve controllers in a simulated environment that then transfer to the physical robot. However, research is still at an early stage and has potential for significant improvement.

9 Acknowledgements

We thank J. Zimmermann at FH Koblenz for using his images and his assistance in software development.

10 References

[1] N. Jakobi, “Minimal Simulations For Evolutionary Robotics” Ph.D. dissertation, Univeristy of Sussex, Brighton , UK 1998.

[2] J. Ziegler and W. Banzhaf, “Evolution of Robot Leg Movements in a Physical Simulation” in *Proceedings of the Fourth International Conference on Climbing and Walking Robots*, CLAWAR , 2001

[3] J. Ziegler, J. Barnholt, J. Bush and W. Banzhaf, “Automatic Evolution of Control Programs for a Small Humanoid Walking Robot”, in *5th International Conference on Climbing and Walking Robots (CLAWAR)*, 2002

[4] A. Boeing and T Bräunl, “Evolving Splines: An alternative locomotion controller for a bipedal robot” in *Proceedings of the Seventh International Conference on Control Automation, Robotics and Vision* (ICARCV 2002), 2002.

[5] O. Miglino, H. H. Lund, and S. Nolfi. “Evolving Mobile Robots in Simulated and Real Environments.”, in *Artificial Life*, vol. 2, pp. 417--434, 1996.

[6] G. S. Hornby, S.Takamura, J. Yokono, O. Hanagata, M. Fujita and J. Pollack, “Evolution of Controllers from a High-Level Simulator to a High DOF Robot.” in *Evolvable Systems: from biology to hardware; proceedings of the third international conference* (ICES 2000), 2000.

[7] T. Bräunl *Embedded Robotics*, Berlin: Springer-Verlag, 2003

[8] Hitec RCD, “Announced specification of HS-945MG Standard Coreless Motor High Torque Servo”, 6th November 2003. Available: http://www.hitecrcd.com/Servos/hs945.pdf

[9] S. McMillan, “DynaMechs: A multibody dynamic simulation library.”, 6th November 2003, Available: http://dynamechs.sourceforge.net/

[10] R. Featherstone, “The Calculation of Robot Dynamics using Articulated-Body Inertias”, *Int. J. Robotics Research*, vol.2, no.1, pp. 13-30, 1983.

[11] S. McMillan, D. E. Orin, and R. B. McGhee, “DynaMechs: An Object Oriented Software Package for Efficient Dynamic simulation of Underwater Robotic Vehicles.” in *Underwater Robotic Vehicles: Design and Control*, pp. 73-98, 1995.

[12] S. Rodenbaugh, and D. E. Orin, “RobotBuilder”, 6th November 2003, http://www.eleceng.ohiostate.edu/~orin/RobotBuilder/RobotBuilder.html

[13] R.C. Dorf, and R.H. Bishop, *Modern Control Systems*, Prentice-Hall, 2001

[14] R. W. Landee, D. C. Davis, and A. P. Albrecht, *Electronics Designers' Handbook* McGraw-Hill, 1977

[15] R.A. Brooks, “Artificial Life and Real Robots,” in *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, , pp. 3-10. 1992.

[16] B. Yamauchi, R. Beer, “Spatial Learning for Navigation in Dynamic Environments”, *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics, Special Issue on Learning Autonomous Robots*, vol.26, no.3, pp.496-505 1996