Making Intelligent Walking Robots Accessible to Educators: A Brain and Sensor Pack for Legged Mobile Robots

Jerry B. Weinberg†, William Yu†, Kim Wheeler-Smith*, Robin Knight*, Ross Mead†, Ian Bernstein*, Jeff Croxell†, Doug Webster*

*RoadNarrows LLC
1151 Eagle Drive #140
Loveland, CO 80537

†Southern Illinois University Edwardsville
Department of Computer Science
Edwardsville, IL 62026
kim.wheeler@roadnarrows.com, jweinbe@siue.edu

Abstract

Wheeled robot platforms have reached an ease of use that allows AI educators to easily incorporate them into their courses to provide an opportunity for students to program physically embodied agents. This invites an opportunity for students to learn AI techniques for dealing with the physical environment and, just as importantly, brings an added interest and excitement to the course. Legged robot platforms have yet to reach the same ease of use. These robots add interesting problem dimensions and an additional level of inspiration to the AI class. This paper describes SkwelZone, a generalized system that brings affordable, programmable, legged robot platforms with reusable sensors, a microcontroller, and a common computing platform into AI curricula.

Introduction

Wheeled robot platforms have evolved to the point that instructors who have little or no background in robots can still use them in their AI courses. This gives students the opportunity to program embedded agents and learn AI techniques for dealing with the difficulties of the physical environment. It also brings an important level of interest and excitement to AI courses (Klassner 2206). Robot microcontrollers such as the Handy Board, the XBC, and the LEGO RCX (Martin, et al. 2000) provide users with simple techniques for connecting sensors and motors, as well as straightforward methods for programming in a variety of programming languages (Fagin 2003, Klassner 2002, Bagnall 2002). However wheeled platforms, which these controllers are primarily used for, require only a small number of motors (e.g. two or four). These controllers cannot be used for legged platforms that require a large number of servos for complex walking motions.

While there are wheeled robot platforms with sufficiently developed middleware products that make robotics accessible to undergraduate educators, there are yet platforms with sufficiently developed middleware to do the same with legged robots. One noted exception is the Sony quadruped AIBO, but this robot has been discontinued. Sony's target market was entertainment; however, educators had found AIBO with third party middleware useful in the classroom and research, e.g., (Drexel University, Chilton & Maria 2007).

The interest in legged robots is evident in the increase of legged robot platforms and competitions. For example, see Robot Magazine issue 5 (Robot Magzine 2007) and the Robo-One Tournament (Robo-One Tournament 2007). The discontinuation of the AIBO has left educators interested in walking robots a choice of platforms unfortunately without sufficient middleware tools to make them easy to use.

Carnegie Mellon University's (CMU) CMRoboBits course has shown that a robotics course can successfully introduce students to the complexities of legged locomotion. The course uses the Sony AIBO to introduce students to all the "bits" necessary for creating a complete intelligent robot (Veloso, et al. 2006, CMRboboBits 2008). The curriculum covers intelligent behaviors, motion, vision, localization, path planning, and multi-robot coordination. The course leverages the open-source development framework *Tekkotsu* developed by CMU for programming the Sony AIBO. A number of teachers have subsequently made use of Tekkotsu to incorporate AIBO's in courses, for robo soccer competitions, and for the Urban Search and Rescue competition (Tejada, et al. 2003).

More recently, Pyro libraries have been developed for the Sony AIBO that provide a programming environment that

is more accessible to students and educators (Blank et al. 2003, Blank et al. 2004, Pyro 2008). Pyro, which stands for Python Robotics, is a Python-based programming environment for easily exploring robotics without having to worry about the low-level details of the underlying hardware of different robot platforms. By creating a collection of Python object classes that abstract all of the underlying hardware details, Pyro provides a uniform software interface for the users to explore different types of mobile robots.

The goal of this project is to bring affordable, programmable, legged robot platforms with reusable sensors, a microcontroller, and a common computing platform into classroom curricula. The system, called SkewlZone, incorporates the Brain Pack (See Figure 1), which is a single processor pack with sensors that can be used on several commercial legged platforms. Investment in sensors, software and curricula is reduced by having the Brain Pack compatible with different robots. One of our objectives in this project is to develop advanced modules that support mobility, localization, navigation, path-planning, and intelligent behaviors. These software modules are being developed with the intent of supporting curriculum for courses in AI and robotics. We are leveraging the already well developed open source framework of Tekkotsu and the many developed modules for intelligent behavior for this purpose.

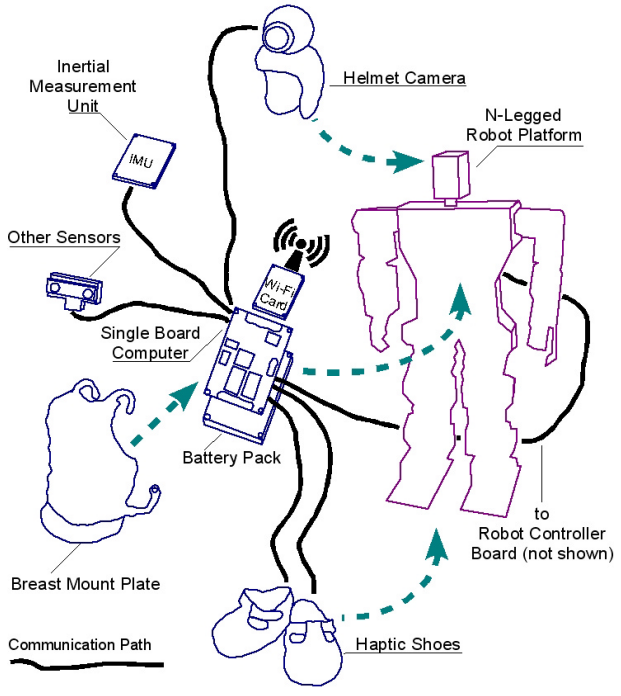

Figure 1: Exploded view of Brain Pack and sensor suite

SkewlZone: The Brain Pack and Sensors

SkewlZone is a generalized system that provides perception and cognitive ability to off-the-shelf legged robots such as the Kondo KHR-2HV. We have developed specialized haptic foot and hand sensors, and inner-ear balance (3-axis inertial measurement unit) using a standard I2C interface integrated with an on-board embedded XScale Linux board. The software architecture has been developed to allow real-time sensing and control on-board. Applications can be quickly developed using the SkewlZone open interface, which provides a layer of abstraction away from the details of the mechanical control of the legged system.

Many of the popular legged robots operate almost entirely in open-loop. That is, the robots have no or limited sensory input from the environment outside of servo position and speed data. Manufacturer-supplied robot controllers are usually dedicated for the real-time control of 17 or more servos. They have little access capacity to process more complex sensory input from the environment and integrate these input streams with the current set of desired robot behaviors and high-level goals. Augmenting a legged robot with the SkewlZone Brain Pack endows it with far greater potential. An easy-to-program single board Linux computer (SBC) connects to add-on sensors, such as hand and foot tactile sensors, an inertial measurement unit, and a USB color camera. The Linux SBC is also connected to the the manufacturer's servo controller to issue commands and receive timely servo state updates. Taken altogether, a robot with a Brain Pack provides a platform with capabilities of motion, touch, balance, eyesight. A Wi-Fi connection is an option for monitoring and for even more intense off-target AI applications.

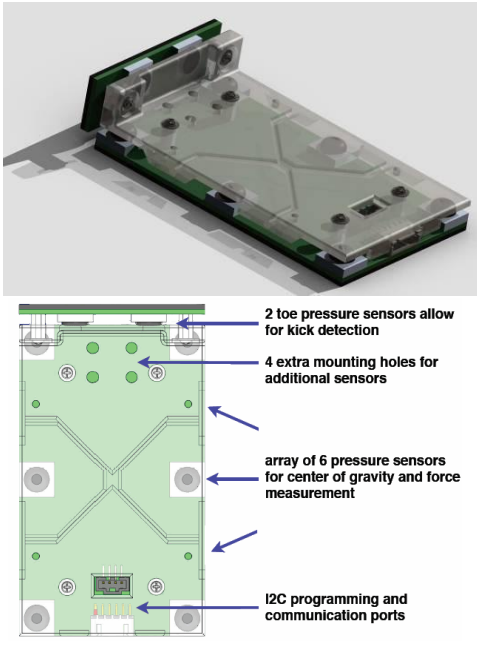

Figure 2: SkewlZone Humanoid Foot

The SkewlZone Humanoid Foot is one of the key sensors of the system (See Figure 2). It gives the roboticist the ability to continuously measure the location of the center of gravity of the robot, the center of pressure on each foot, and the magnitude of force on the bottom of the foot. The toe also gives force feedback making it ideal for soccer competitions. Its on board LEDs give the user instant feedback on the foot's current status and high visual appeal. An onboard Atmel® microcontroller comes preprogrammed with sensor calibration and operation controls, however, an I2C interface offers programmable control over all of the foot's functions for advanced users. When integrated with the Linux Brain Pack, on-board autonomous responses can be programmed for quick, low-level reaction to the environment.

The electrical and communication connections between the SkewlZone Brain Pack and the robots are provided by custom connectors made by RoadNarrows. The mechanical attachments for the Feet and Brain Pack are designed to easily connect to the supported robot platforms, and should take the user only a few minutes to assemble. All of the software and libraries developed for the Brain Pack hardware are provided and is open-source. RoadNarrows has developed a command-line shell that is generalized to very easily control different robot platforms with different sensor configurations. Specific robot controller interfaces, such as the RCB3 protocol, are contained in separate libraries. Only those libraries specific for the configuration in use are required to be loaded on the Brain Pack. The shell and TCP/IP interface developed for the Brain Pack make it very easy to integrate the SkewlZone system with a variety of higher-level applications, such as Webots™ by Cyberbotics, Microsoft Robot Studio, and Pyro Robotics (Pyro 2008). Simple platform-agnostic examples for controlling the robot and retrieving sensor data are provided in Python.

SkewlZone: AI Programming

As an open source application development framework for intelligent robots, CMU's Tekkotsu provides many modules, written as C++ classes, for controlling 4-legged robots, including movements of the four legs and head, as well as getting input from cameras, microphones, and touch sensors (Tekkotsu 2008). It has been successfully adopted in many AIBO based robotic courses (Turner 2008, Bowling 2008). Tekkotsu also provides a means for users (or developers) to define their own robot configurations. We are currently going through the process of defining new hardware configurations for legged robots such as the Kondo KHR-2HV. This includes creating a new namespace, defining the parameters in the newly created namespace, and providing kinematics information to allow modeling and manipulation of the robot's frame.

Defining New Behavior

As mentioned earlier, software modules are being developed to support curriculum of AI and robotics courses. In Tekkotsu, a user defines behaviors by creating C++ classes derived from existing Tekkotsu base classes. Given the large number of existing classes, it is relatively easy to create additional modules. The following is an outline of a simple obstacleAvoidance class which continues to walk while avoiding obstacles. It is a simple extension of the WalkForwardAndStop module (Tekkotsu 2008).

```
/* Robot will walk straight until an obstacle is detected by the IR
sensor.  It will turn until no obstacle is detected and then resume
walking. */

#include "Behaviors/BehaviorBase.h"
 // base for all; include other head files such as WalkMC.h

class obstacleAvoidance : public BehaviorBase {
public:

... // init various ids to invalid
virtual void DoStart() {
        BehaviorBase::DoStart();

        //define walk MotionCommand
        SharedObject<WalkMC> walk;
        walkID=motman->addPersistentMotion(walk);
        ...// add head MotionCommands

        // set up head position, this is hardware dependent
        ...

        // listen to all events
        erouter->addListener(this,
                                EventBase::sensorEGID);
}

virtual void DoStop() {
        // remove motCommands, stop listening, etc
        // call superclass DoStop
        BehaviorBase::DoStop();
}

virtual void processEvent(const EventBase& event) {
        if(event.getGeneratorID() ==
          EventBase::sensorEGID) {
            //access the walk motion command
            MMAccessor<WalkMC> walk(walkID);

            //get the current value for the IR sensor
            float distance =
               state->sensors[FarIRDistOffset];
            if(distance <= IRMinDist) {
               walk->walk.mc()->
                        setTargetVelocity(0.0, 0, 0.5f);
            }
            else
               walk->walk.mc()->
                        setTargetVelocity(150.0, 0, 0);
```

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

//class variables
protected:
        MotionManager::MC_ID walkID, headID;

};      // end of class
```

Creating new behavior in Tekkotsu involves defining three essential functions: DoStart() which does the initial set up, DoStop() which does the clean up, and processEvent(). In this example, DoStart() initializes the walk and head variable and set the EventRouter variable (erouter) to subscribe to all events. The function processEvent() is called when a registered event is caught, which, in this case, involves a distance reading from the IR sensor and setting the target (robot) velocity (x, y, ω), where x is the forward-reverse motion, y is the side-to-side strafing motion (orthogonal to x), and ω is the angular velocity. It sets the target velocity to (0, 0, 0.5f) (i.e., making a turn) if the distance is less than a pre-defined minimum threshold value. Otherwise, it maintains the target velocity at (150, 0, 0) (i.e., walk straight forward). As we can see from this example, Tekkotsu users are able to define behavior at a higher level of abstraction, leaving Tekkotsu to handle the hardware details.

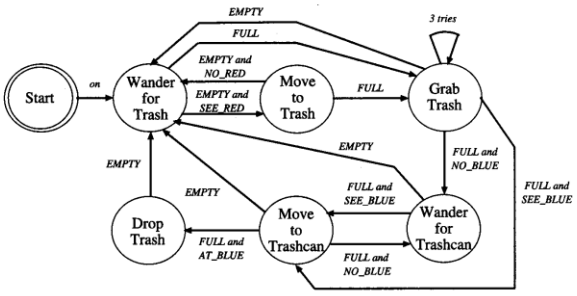

Figure 3: An FSM representing behavior of a trash pickup robot (reproduced from Murphy 2000, p. 181)

Defining Behavior using FSM

This technique of behavior definition follows the reactive/behavior-based approach (Murphy 2000), where behaviors are built by successive layers of lower level reactive control. A Finite State Machines (FSM) can be used to describe the overall behavior of a robot, in terms of a finite number of action states (or nodes) and events (or triggers) that cause the transition between these states. Being in a state means that the robot only responds to a specific subset of inputs and reacts in a specific way. FSMs provide an elegant and effective way of describing and analyzing robotic behavior. For example, the behavior of a trash pickup robot can be described by the FSM in Figure 3. As you can see from the FSM diagram, the robot begins in the Wander_for_Trash state; it then changes to the Move_to_Trash state when it sees the trash (i.e., SEE_RED) and its hands are EMPTY, and so on. Tekkotsu supports FSMs with a StateNode class. The obstacleAvoidance behavior discussed previously can be implemented as a simple FSM as illustrated in Figure 4.

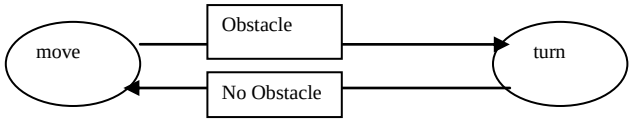

Figure 4: Obstacle Avoidance FSM

The following is an outline of its implementation in Tekkotsu demo behavior ExploreMachine (Tekkotsu 2008):

```
class ExploreMachine : public StateNode {
public:

... // constructor

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

virtual void setup() {
        SharedObject<WalkMC> walk;
        walkid=motman->addPersistentMotion(walk);
        WalkNode * move=NULL;
        addNode(move=new WalkNode(getName()
                                        +"::move",150,0,0));
        move->setMC(walkid);
        start=addNode(turn=new WalkNode(getName()
                                        +"::turn",0,0,0.5f));
        turn->setMC(walkid);

        // Add a transition from turn to move after 2 ms
        turn->addTransition(new  TimeOutTrans(move,2000));
        // add a transition from move to turn
        ...
        StateNode::setup();
}

void DoStart();
void ExploreMachine::DoStop();
virtual void processEvent(const EventBase& /*e*/);
virtual void teardown();

// called each time the turn node is activated, sets a new
// random turn direction and speed
virtual void processEvent(const EventBase& /*e*/);
```

```
protected:
    StateNode * start; // the node to begin with
    WalkNode * turn; // walk node to use when turning
    // we want to share a walk
    // between turning and walking nodes
    MotionManager::MC_ID walkid;

};
```

The setup() function creates the actual automaton by creating nodes and transitions using the addNode() and addTransition() methods. For example, *move* is defined as a walkNode and added with the statement

```
addNode(move=new WalkNode(getName()
                          +"::move",150,0,0));
```

which also specifies its name as ExploreMachine::move, and the velocity as (150, 0, 0) (walking straight at a speed of 150). DoStart() is run when a node is activated while DoStop() is run when a node stops. In other words, during a transition, DoStop() of the source node as well as DoStart() of the destination node are both executed.

Sample Projects

The nature of Tekkotsu's class structure allows more complex behaviors to be defined or expanded from simpler methods. Students will be able to focus on design of higher level behavior while leaving Tekkotsu to handle lower level hardware control. We illustrate that with a few sample projects

Project 1 – Basic Event-driven Motion Control

In this project, students will implement behavior for an *obstacle avoiding* robot, which is an extension of the *obstacleAvoidance* behavior outlined earlier. Specifically, the *obstacle avoiding* robot should act as follows:

- Start in a waiting state.
- Walk forward when the start button is pushed.
- Continue to walk in the same direction until either an obstacle is detected,
- When an obstacle is detected, repeatedly turn an angle and look for obstacle,
- Resume walking forward.
- Stop when the stop button is pushed.

The goal of this assignment is for students to gain understanding of sensor and motion control as well as Tekkotsu's event-driven programming.

Project 2 – Defining Behavior with FSM

In this project, students will implement the *obstacle avoiding* robot using a Finite State Machine (FSM). The emphasis of this assignment is on using appropriate Tekkotsu modules to construct automata that create the desired robotic behavior. Contrasting with project 1, students will develop an appreciation of FSMs, how they can describe generate overall intelligent behavior. For this assignment, students are encouraged first create diagrams of their FSMs. They will then use Tekkotsu Walking node to model walk and turn behavior.

Tekkotsu also includes a Storyboard Tool which allows the user to monitor the execution of a state machine and display a graphical execution trace called a "storyboard", as shown in figure 5 (Tekkotsu 2008). Created by Akkarit Sangpetch of CMU as a Senior honors project, the Storyboard tool provides detailed information on the nodes (name, type, etc) and the transitions (sources and destinations of each) as well as the execution of your FSM including state activation and deactivation events, transition firings, and sensor events. Students can use the Tekkotsu Storyboard Tool to monitor their FSMs.

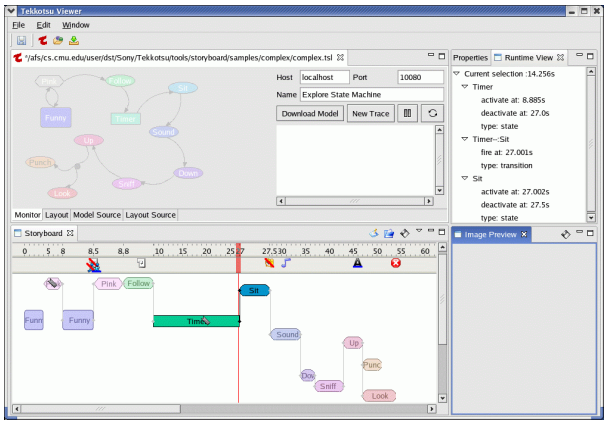

Figure 5: Tekkotsu Storyboard tool screen shots

Project 3 – Object Recognition and Subsumption Architecture

In this project, students will implement a *treasure hunting* behavior which extends from the *object avoiding* behavior created in previous projects. Specifically, the *treasure hunting* robot should act as follows:

- Start in a waiting state.
- Walk forward when the start button is pushed.
- Continue to walk green colored treasure is detected, at which point it stops.
- If an obstacle (i.e., the wall) is detected, the robot will repeat the turn and observe process until no obstacle is found, and then resume walking forward.
- If a treasure is detected, it will use its arm to pick up the treasure, play a sound and then stop.

The goal of this assignment is for students to develop a

reactive control architecture using a simple suppression network (subsumption) for a foraging robot which should exhibit obstacle avoidance as well as a searching behavior and a collect behavior. Additional behaviors such as fleecing can be introduced with the introduction of a predator recognized by another color such as red.

Another objective of this assignment is to introduce students to simple object recognition with a camera. Tekkotsu's DualCoding package provides support for in-depth image processing and has been used to successfully extract features from fair complex object such as a tic-tac-toe board (Touretsky & Tira-Thompson 2008). As a first assignment that uses the camera, students will only need to program the robot to recognized objects based on their colors.

Conclusions

SkewlZone is currently in prototype testing. The Brain Pack along with the device libraries and scripting language has been developed. Tekottsu definitions necessary to connect to the Brain Pack have also been developed. The Humanoid Foot sensors have been engineered and are currently being tested. Many of the other sensors are in various stages of creation. A full prototype implementation is expected by this summer and a production model will be designed from the results of the prototype testing.

For further information, images, and video for this project please see: http://wiki.roadnarrows.com/index.php?title=schoolZone_-_Main_Page . A video showing the foot haptics and adaptive response of a humanoid foot robot is available at: http://www.roadnarrows.com/customer/SkewlZone/latest/media/KondoFootSensors.MPG

Acknowledgements

This grant is supported by the National Science Foundation Small Business Technology Transfer Research (STTR) Program under Grant No. 0711909.

References

Bagnall, B., *Core LEGO MINDSTORMS Programming: Unleash the Power of the Java Platform*, Prentice Hall, 2002.

Blank, D., Meeden L., and Kumar, D., "Python Robotics: An Environment for Exploring Robotics Beyond LEGOs", *Proceedings of the Thirty-Fourth SIGCSE Technical Symposium on Computer Science Education*, Reno Nevada, ACM Press, February 2003.

Blank, D.S., Yanco, H., Kumar, D., and Meeden L. "The Karel-the-Robot Paradox: A framework for making sophisticated robotics accessible", *AAAI 2004 Spring Symposium on Accessible, Hands-on Artificial Intelligence and Robotics*, Stanford University, CA, March 2004. AAAI Press Technical report SS-04-01, 2004

Bowling, M. "Course Material for CMPUT412: Experimental Mobile Robots" , March 2008 http://www.cs.ualberta.ca/~bowling/classes/cmput412/

Chilton, J. and Maria, G., "Using AIBO's in a CS1 Course", in the Technical Report (SS-07-09) of the AAAI Spring Symposium on Robots and Robot Venues: Resources for AI Education, pp. 24 – 28.

CMRoboBits, March 2008, http://www-2.cs.cmu.edu/~robosoccer/cmrobobits/

Drexel University's CS 511, September 2007, http://www.cs.drexel.edu/~pmodi/cs511.html

Fagin, B, "Ada/Mindstorms 3.0: A Computational Environment for Introductory Robotics and Programming", IEEE Robotics and Automation, Spring 2003, pp 19-24.

Klassner, F, "A Case Study of LEGO Mindstorms Suitability for Artificial Intelligence and Robotics Courses at the College Level", *Proceeding of the 33rd SIGCSE Technical Symposium on Computer Science Education*, Northern Kentucky, February 2002, pp. 8-12.

Klassner, F., "Launching into AI's October Sky with Robotics and Lisp", AI Magazine, Vol. 27, No. 1, Spring 2006, pp. 51-65.

Martin, F., Mikhak, B., Resnick, M., Silverman, B., and Berg, R., "To Mindstorms and Beyond: Evolution of a Construction Kit for Magical Machines," *Robots for Kids: Exploring New Technologies for Learning*; A. Druin and J. Hendler, eds., Morgan Kaufmann, 2000, pp. 9-33.

Murphy, R , Introduction to AI Robotics, 2000, MIT Press, Cambridge, Massachusetts.

Pyro website, March 2008, http://www.pyrorobotics.org/

Robot Magazine issue 5, September 2007, http://www.botmag.com/articles/robo_one_ten.shtml

Robo-One Tournament, Septemper 2007, http://www.botmag.com/issue5/index.shtml

Tejada, S. Cristina, A., Goodwyne, P., Normand, E., O'Hara, R., and Tarapore, S., "Virtual Synergy: A Human-Robot Interface for Urban Search and Rescue", In the Proceedings of the AAAI 2003 Robot Competition, 2003.

Tekkotsu website, March 2008, http://www.cs.cmu.edu/~tekkotsu/

Touretzky, D. & Tira-Thompson, E. "The Vision Pipeline and Color Image Segmentation", March 2008, http://www.cs.cmu.edu/afs/cs/academic/class/15494-s08/lectures/vision_pipeline.pdf

Turner, S., "Introduction to Behavior using Tekkotsu", March 2008, http://www.tekkotsu.org/media

Veloso, M., Rybski P., Lenser S., Chernova S., and Vail D., "CMRoboBits: Creating an Intelligent AIBO Robot", *AI Magazine*, Vol. 27, No. 1, Spring 2006, pp. 67-82.