

System Vision Case Study: Robotic System Design

**Dan Block
Senior Project
Oregon State University**

Introduction

The TekBot™ is part of the Oregon State University (OSU) *Platforms for Learning*™ concept¹, created to teach students about analog circuitry, digital logic, and embedded systems – all in the context of robotic system design (see Figure 1). This paper describes an OSU senior project that focuses on system modeling techniques, using the SystemVision simulation environment produced by Mentor Graphics. Modeling with VHDL-AMS and SPICE models, simultaneously, is a significant benefit in exploring and understanding the interactions of the different hardware devices on the TekBot². In addition to modeling the basic TekBot, additional hardware was modeled that would allow the TekBot to find and sort 5 orange balls and 5 black balls into color-coded corners in a 10'x10' arena.

System modeling and simulation can be useful in projects such as this one, particularly when there are non-trivial interactions between subsystems. Co-verification of hardware and software was very beneficial. Finally, modeling of the system context (the arena) proved to be invaluable for this project.

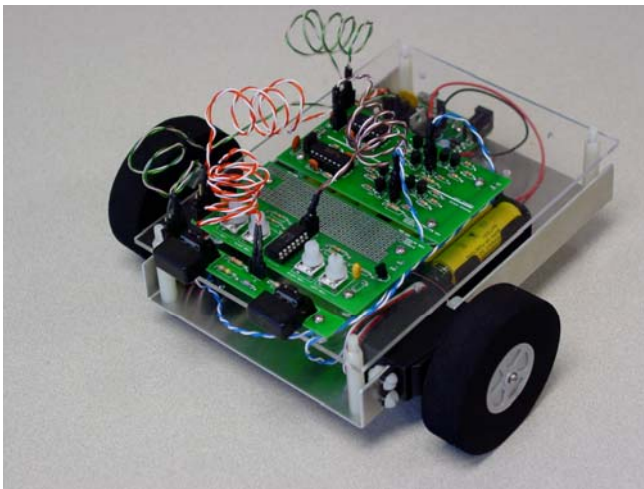


Figure 1. Basic Tekbot Platform

Methodology

To enable the TekBot to sort the randomly placed balls in the arena, hardware was needed that would not only meet the power requirements of

the battery-operated system, but would also allow the Tekbot to find and move all of the balls in the arena in a timely fashion. To find the balls quickly, a range finder was mounted on the front of the Tekbot to determine the approximate distance to an object. The Daventek SRF-08 ultrasonic range finder was chosen because it was fairly inexpensive and operated on the I2C interface, which is a 2-wire communications protocol supported by the microcontroller used on the TekBot. To read the color of an object, a simple analog circuit was developed that produces a voltage proportional to the intensity of light reflected from a surface. This output voltage is processed by the Tekbot's microcontroller through the on-board Analog-to-Digital converter. The source of the reflected light comes from 3 individual LEDs: red, blue and green. The readings taken from each source are then used to determine the surface color. This flexible solution provides recognition of a broad spectrum of colors and shades, even though the system specification required only recognition of orange, black, blue and white. The ball-capture subsystem was developed using two servos controlling gate arms that enclose a ball and allow the Tekbot and ball to roll together, maneuvering as necessary around the arena.

Preliminary design of these sensors and actuators included development of VHDL-AMS models. SystemVision was used to verify that these subsystems would work correctly in the context of the full system. This step involved creating schematics that included the individual sensor or actuator subsystem and the necessary power, stimulus, and loads for various test conditions. These schematic testbenches were then tested, via SystemVision simulation, to observe the circuit's behavior. The waveforms obtained by simulation provided the required visibility into both the models and the subsystem design – any deficiencies were corrected and verified through simulation.

Once the subsystem module designs were stabilized, they were integrated into the overall system design, where they were “attached” to the simulation model of the basic Tekbot. This allowed the microcontroller software to be developed and tested, including the sensor and actuator hardware driver code. The system model included a model of the arena, allowing the actual application software for sorting the balls into the corresponding arena corners to be tested and corrected, all in a virtual environment. This was particularly valuable for debugging software modules such as the line-following algorithm, since it provided simultaneous visibility of difficult-to-observe hardware and software states.

Simulation Models

The following paragraphs describe the basic Tekbot subsystems and the modules that were added for its ball-sorting mission. Modeling of each module is highlighted as appropriate.

Charger Board:

The analog electronics of the TekBot include the charger board, which takes a 9V input from a 110-volt power supply and uses it to charge onboard batteries on the Tekbot. This board is also responsible for delivering 6 separate power channels to the rest of the TekBot hardware.

A simple behavioral VHDL-AMS model of this subsystem was all that was necessary for this project.

Motor Controller Board:

This board is responsible for controlling the direction of the wheels that drive the Tekbot. It consists of two H-bridge analog electronic circuits that, depending on the inputs given, will switch the polarity of the voltage applied to the terminals of the modified servos that drive the TekBot forwards or backwards (see Figure 2). This circuit is modeled using traditional SPICE elements for the transistors and a VHDL-AMS model for the mixed-technology motor. The transistor device models were available from the

part manufacturer’s website and the motor model was constructed by making simple modifications to a VHDL-AMS model provided with the SystemVision software.

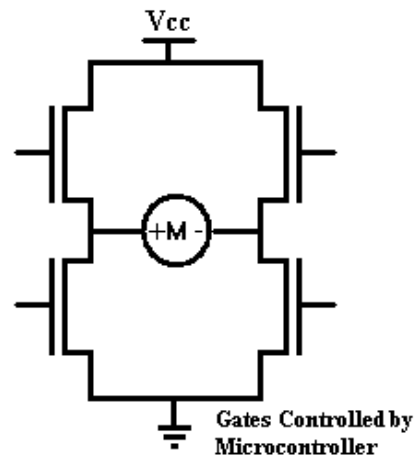


Figure 2. H-bridge motor controller

Digital Components:

The digital controller for the TekBot is the Atmel Atmega128 microcontroller. It has useful system-level features such as I2C Communication, Analog to Digital Conversion, and Extended SRAM. This processor is incorporated into a board that is produced by Oregon State University and supplied as part of the *Platforms for Learning* program. To implement a system-level model of this microcontroller in VHDL-AMS, an open source ATmega103 model (found online) was modified to model the Atmega128. The Atmega103 was the precursor to the Atmega128 and is backwards compatible. The Atmega103 VHDL model needed several modifications to provide the Atmega128 features: additional interface ports were added, and an 8-channel analog multiplexer feeding an 10-bit ADC unit. All of these additions were implemented in the VHDL-AMS language.

Color Sensors:

The three color sensors were implemented by first creating a model for an LED. A diode model found in *The System Designer’s Guide for VHDL-AMS*³ was used as a starting point. Additional equations were used to calculate a logarithmically

proportional light output based on the current in the device. The light output is modeled using an output “port” utilizing the VHDL-AMS Radiant energy “type”. The equations used to create this model are in Figure 3.

Area -> Area of silicon

Dn -> electron diffusion current

Dp -> hole diffusion current

np -> minority charge density

pn -> minority charge density

Ln -> diffusion length for electrons

Lp -> diffusion length for holes

Vt -> threshold voltage

$$vt = temp * \frac{K}{Q}$$

$$id = Q * area * \left[Dp \left(\frac{pn}{Lp} \right) + Dn \left(\frac{np}{Ln} \right) \right] * \left(e^{\frac{v}{vt}} - 1.0 \right)$$

$$power = v * id$$

$$Candescence = \log(1 + id)$$

Figure 3. LED equations

The same equation was modified to create the phototransistor model. Since the current output of a photoconductive material is proportional to the incident light on the material surface, the model was coded to output a minimal “dark current” if the radiance at the base terminal is less than a threshold value. If the radiant energy exceeds the threshold, a proportional output current is produced through the device.

Sonar:

The sonar model is a behavioral model that interacts with the model of the arena. It receives control commands from IO pins on the processor. When the model receives logic ‘1’ on the start pin, it records the current simulation time and sends out the sonar “ping” (in this case a std_logic signal) to the arena. The arena then calculates the appropriate delay for the sonar ping and sends an echo on the echo pin of the model after the calculated delay. When the sonar model receives the echo back, it calculates the difference in time between the “ping” and the “echo” and divides the

total delay by half, and then converts the time taken to travel to the object into a binary representation in inches. While the sonar is ranging, the output on the IO pins is a decimal 255 representing max range. This tells the microcontroller to keep reading the output until it is not 255. This corresponds to the actual behavior of the SRF-08 sonar system.

Grabber:

A detailed model of the grabber was not necessary for this project. The microcontroller uses a single IO pin to represent the state of the grabber hardware; 0 is open, 1 is closed. When the state of the pin switches, the arena calculates the position and direction of the TekBot. If it is close enough and facing the right direction, the arena will recognize the ball as being “picked up.” Otherwise it will do nothing.

Modified Servos:

The motors that drive the TekBot are servo motors modified to behave more like a DC motor than a positional servo (the servos are modified by bypassing the integrated PWM controller and removing a mechanical “stop” tooth on the axis of rotation). This approach was chosen because typical DC motors don’t produce adequate torque on their own to make the Tekbot move – a gear reduction system is needed. The modified servo approach leverages the existing gear structure commonly available in servo motors. The VHDL-AMS code implementation of the model can be seen in Figure 4.

```

01 library ieee;
02 use ieee.electrical_systems.all;
03 use ieee.mechanical_systems.all;
04
05 entity modified_servo is
06     generic(
07         r_int : resistance := 0.06;           -- Coil resistance (measured)
08         kt     : real := 2.44e-3;           -- Torque Constant [N*m/Amp] (VHDL-AMS book pg444)
09         l      : inductance := 2.03e-6;     -- Winding inductance [Henrys] (VHDL-AMS book pg444)
10         d      : real := 0.166e-6;         -- calculated from measurements
11                                             -- Damping coefficient [N*m/(rad/sec)]
12         j      : moment_inertia;           -- Estimate
13         kg     : real := 370.0;            -- gear ratio from servo cage
14     );
15     port(
16         terminal p : electrical;
17         terminal n : electrical;
18         terminal shaft : rotational_velocity);
19 end entity modified_servo;
20
21 architecture ideal of modified_servo is
22     quantity v across i through p to n;
23     quantity w_shaft across torq_shaft through shaft to rotational_velocity_ref;
24     quantity w : real;
25     quantity torq : real;
26 begin
27     torq == -1.0*kt*i + d*w + j*w'dot;
28     v == kt*w + i*r_int + l*i'dot;
29     w_shaft == w/kg;
30     torq_shaft == torq/kg;
31 end architecture ideal;

```

Figure 4. VHDL-AMS code for DC motor

Embedded Software Development Tools:

The ATmega family of processors is very easy to program. For nearly every model of the processor, a C cross compiler can be used to compile C source code into a native ATmega binary file. The next step involves running the binary output file through the *Hex2Jam* utility, which converts a binary file into a memory module that encapsulates the binary data into specific memory locations. With a few simple automated manipulations, the new memory module can be copied to the appropriate VHDL-AMS project location, effectively overwriting the old program code. The VHDL-AMS project must be re-compiled to include the new code. Once this is

completed, the project can be simulated to verify the program execution in the context of the entire system model!

Arena:

The arena model has many uses in this design. It uses VHDL-AMS processes to handle the different events occurring in the arena during the simulation. The wheel models output a velocity vector into the arena model. The arena model is responsible for taking the integral of these vectors to calculate the position of the Tekbot in the arena. If the TekBot hits a boundary, the arena will stimulate the sensors on the bumper board to steer the Tekbot away from the wall. Since the

bumper board was not used in our design, this process still stimulates the sensors on the Tekbot, but it does nothing to change its behavior.

The arena is also responsible for the sonar echo delay calculation. This is a lengthy calculation because it uses the position of the wheels to determine the bearing of the TekBot and calculates the distance from the left and right edges of the sonar cone. It then checks to see if any balls are in the cone and calculates the distance to them. The shortest distance of these calculations is multiplied by 2 to represent the echo traveling from the robot to the object and then back. Then the arena model sends out the echo signal after this calculated delay.

The arena uses the position of the Tekbot's color sensors and an IF/ELSE statement to create a continuous reflection response that is sent back to the color sensors.

The arena is also responsible for modeling the balls in the arena. When the TekBot grabs a ball, the arena updates the coordinates of the ball with respect to the TekBot's position. It also determines whether or not a ball has been captured and moved to one of the arena corners.

As can be seen, the arena model is a pivotal component of the overall system model, facilitating testing of the Tekbot operating in its real world environment.

System-Level Simulations

Once the individual Tekbot component models were available, they could be assembled into a virtual Tekbot system. A graphical symbol was created for each component model. These symbols were used to graphically assemble the system, using the schematic facility in the SystemVision environment (see Figure 5). For comparison, see the final Tekbot in Figure 6, including hardware modifications.

The system level schematic of the Tekbot system in Figure 5 was further encapsulated by creating a single symbol to represent this system. This symbol was placed in another (higher level) schematic along with symbols and models representing the arena and simulation stimuli, creating a hierarchical representation of the Tekbot and its test environment (see Figure 7). This hierarchical collection of symbols and associated models can then easily be simulated to explore the interaction of the system software with the sophisticated collection of system hardware. See the final working design sorting the black and orange balls into their respective corners in Figure 8.

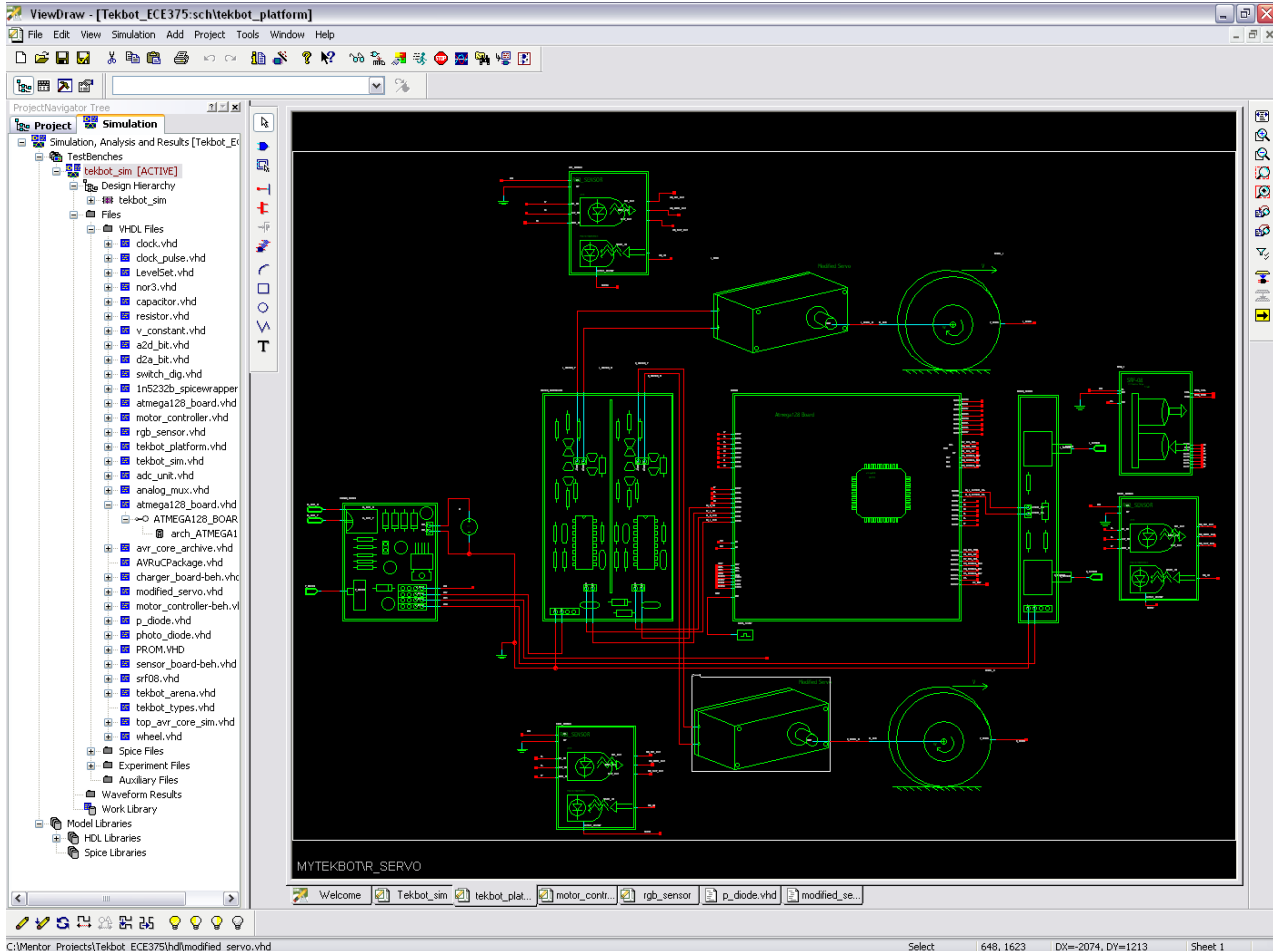


Figure 5. Tekbot system schematic

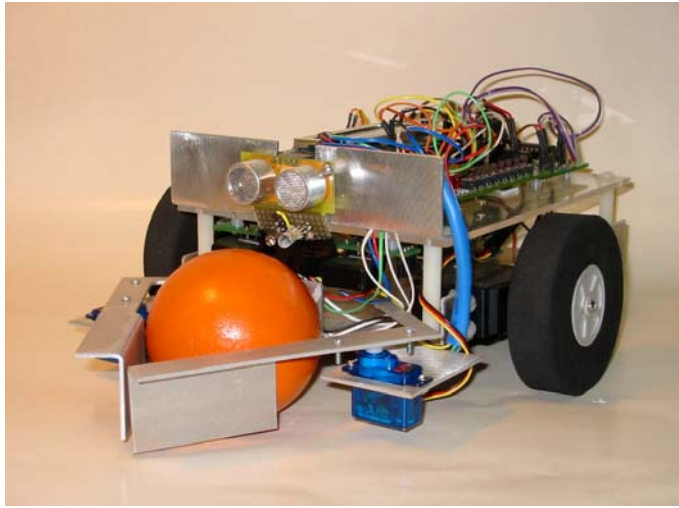


Figure 6. Final Tekbot, including hardware modifications

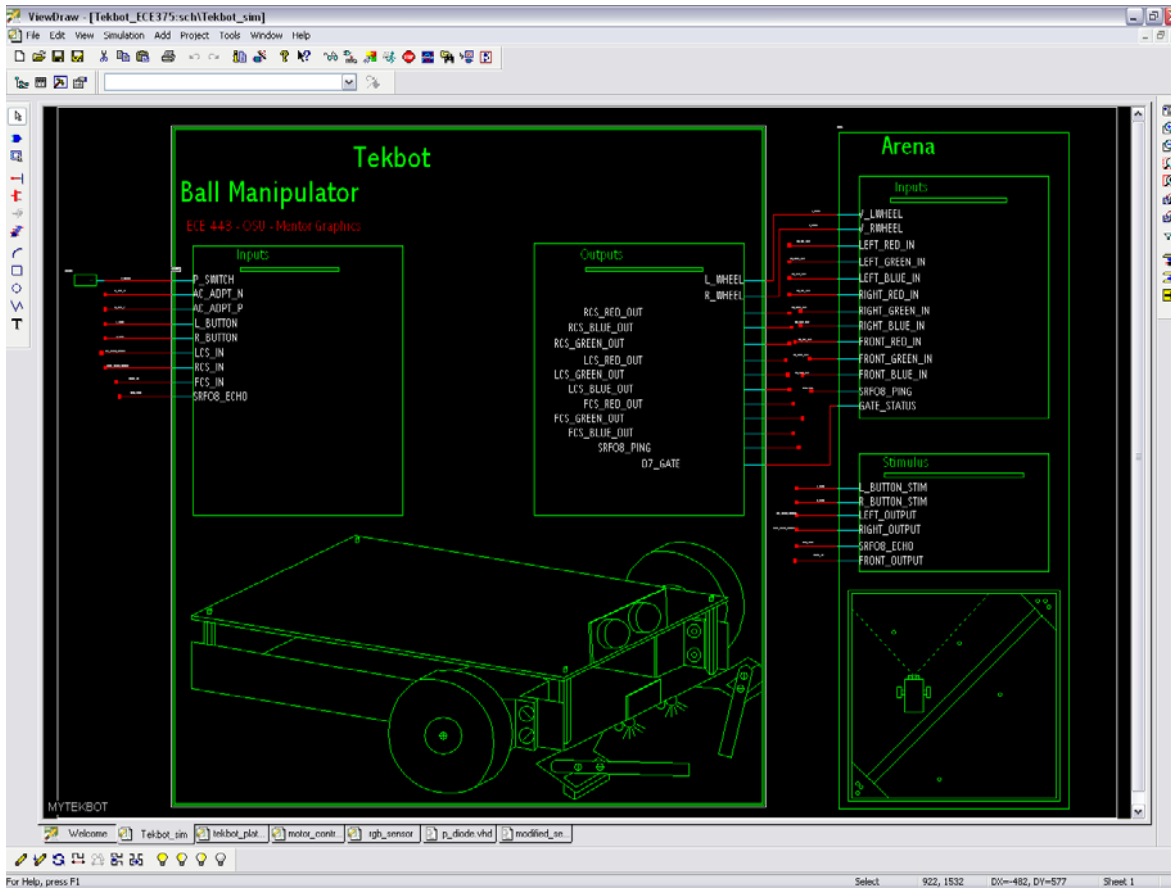


Figure 7. Tekbot in simulated arena

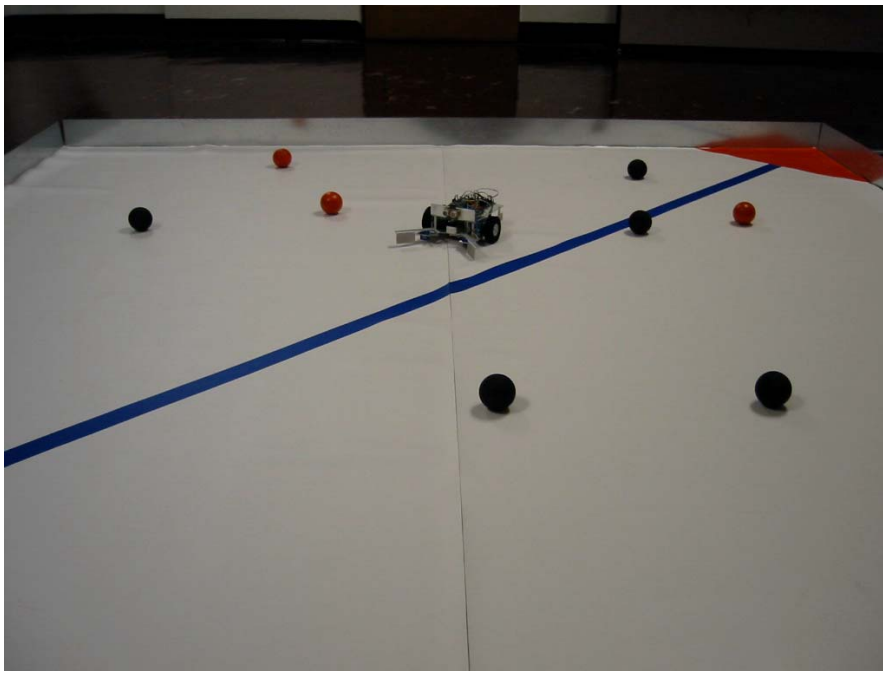


Figure 8. Tekbot sorting balls

Simulation Results

Once the Tekbot and arena models were available, the design process was primarily one of iteratively modifying the microcontroller software and testing its effectiveness using the system simulation models.

After SystemVision had completed each simulation run of the software and hardware, the simulation results were saved to a comma-separated-value (CSV) file and processed to visualize the Tekbot wheel positions in the arena. This verified that the behavior of the TekBot was correct. The result of this process can be seen in Figure 9

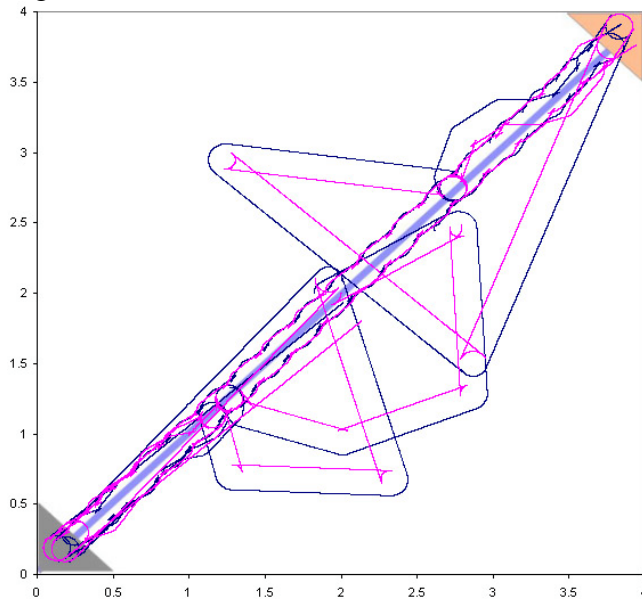


Figure 9. Trajectory of the Tekbot during a simulated run

Without these SystemVision hardware and embedded software simulations, this project would not have been completed. The simulations uncovered many flaws in the algorithms used for ball detection and line following. Traditional software debugging techniques could not be used without interfering with the operation of the Tekbot in the arena. The ability to use the SystemVision waveform viewer to pinpoint the cause of problems was particularly key in the

process of optimizing the ball detection and line following routines.

One unexpected benefit of doing the hardware simulation was the power consumption report that SystemVision produces while simulating a design. Battery life was calculated, given the simulation power dissipation, and compared to actual battery life. The difference between the calculated battery life and the observed battery life was very small. It was reassuring to see that the models of the hardware were reflecting reality, even in power consumption.

VHDL-AMS is a very powerful extension of the already comprehensive VHDL language. The analog electronics extensions and general purpose differential equation, s-domain, and z-domain extensions, combined with all things digital, make this a truly rich language for modeling a broad spectrum of real-world systems. It is difficult to visualize an example of a device covered in the electrical engineering curriculum that cannot be effectively modeled with VHDL-AMS.

The VHDL-AMS language, while powerful, is not trivial to learn. Using a graphical environment, such as SystemVision, significantly helps. The large library of models makes it easier to build systems from existing blocks or to customize existing models for specific needs. A comprehensive book, *The System Designer's Guide to VHDL-AMS*, provides an excellent reference for the VHDL-AMS language. In addition to VHDL-AMS, the capability in SystemVision to directly use SPICE models helps leverage a large body of existing device models and macromodels.

This project demonstrated to a development team the unequivocal benefit of using combined hardware/software modeling to develop a sophisticated, mobile, multi-technology system and get it to work, especially under time pressure.

For More Information

To learn more about this Oregon State University senior project, go to the Project Website
<http://classes.engr.oregonstate.edu/eecs/fall2003/ce441/groups/g20/>

To learn more about Mentor Graphics SystemVision software, go to
<http://www.mentor.com/systemvision>

End Notes

¹ See <http://eecs.oregonstate.edu/education/platforms.html> for more information on the Oregon State University *Platforms for Learning* concept.

² See <http://eecs.oregonstate.edu/education/about.html> for more information about the Tekbot system.

³ P. Ashenden, G. Peterson, D. Teegarden, *The System Designer's Guide to VHDL-AMS: Analog, Mixed-Signal and Mixed-Technology Modeling*. San Francisco: Morgan Kaufman Publishers, September 2002.
<http://www.mkp.com/vhdl-ams>.