Dancing Robots!

CSCE 489 Final Report

Adam Eck, Bradly Paul, and Daniel Podany
University of Nebraska-Lincoln
May 4, 2008

Table of Contents

Abstract

In this final report, we describe the design and implementation of a senior design project involving the creation of dancing robots. The goal of this project was to inspire young children to become more interested in both music education and technology by creating a system where several robots dance interactively based on both synchronous and asynchronous (i.e., stored) music sources. Using a software bridge between music and hardware, our system visualizes MIDI data, helps users choose segments for robots to dance to, assists with the assignment of various robotic actions to those segments, incorporates a *Simon Says* type game for musical instruction, and commands the robots over Bluetooth wireless connections. The main contributions of this report include a description and analysis of the various design decisions made during the course of the project, as well as information about the project's primary components and how their interactions enable dancing robots to form an educational tool.

Section 1: Introduction

1.1. Motivation

Throughout the musical education literature, one common theme stands out: children who listen to and learn about music have increased skills in learning, reasoning, and retention of information (e.g., [1]-[3]). In order to help children become more interested in learning about music both in schools and at home, we have designed an educational toy which combines the playfulness of robots with music education. This toy, Dancing Robots!, enables instructors to use robotic dances to emphasize musical structures visually, providing an extra type of feedback to help children understand concepts at a faster rate. Dancing Robots! also provides a playful response to a child's learning, rewarding children when they repeat melodies performed by an instructor.

1.2. Project Goals

Our goals in creating Dancing Robots! were 1) to build an educational toy that emphasizes entertainment while learning, 2) help children become interested in both music and technology, 3) provide a relatively cheap and affordable product, and 4) use anthropomorphic, programmable robots with many available actions across several degrees of freedom to create our dance performances. These goals were drafted after surveying existing products on the market, and after discussions with both the Computer Science and Engineering Department and the Department of Music at the University of Nebraska-Lincoln.

1.3. Related Work

Looking at the current market, we found that there are existing robots which already dance to music. For instance, Tiger Electronics has created the I-DOG and several other cheap ($20) dancing animal music players [4]. These devices can play music through internal speakers but only dance randomly to the sounds and have a limited number of available actions. Another interesting type of robot popularized in Japan is called BeatBots [5]. Their prominent robot is named Keepon, which looks like a yellow snowman made of two tennis balls. This robot has four degrees of motion and reacts directly to the beat of music, but has a limited number of available actions and is non-programmable. Its price is currently unknown and the robot is not available for sale. Finally, Sony has created the QRIO (formally known as Sony's Dream Robot) which is able to dance and perform to music, as well as sing and conduct dialogs with humans [6]. This robot has even been used to test how children react to small robots with varying degrees of success by introducing the robot as a classmate to a room full of toddlers [7]. However, although the QRIO is probably ideal for our project, its cost is extremely prohibitive: before the project was discontinued, Sony announced the robot would be priced between $60,000 and $80,000 [8].

1.4. Paper Outline

The rest of this paper is organized as follows: in Section 2, we provide an overview of our project's design and introduce the three primary components. In Section 3, we describe how we implemented these components and illustrate how the system works. Finally, in Section 4, we provide a summary of our project and detail our vision for future extension of this toy. Additional information, including a cost analysis, is provided in an appendix.

Section 2: Design Overview

In order to build robots that can perform to music and serve as an educational toy, we have created a design utilizing both software and hardware with three main modules, as shown in Figure 2.1. These modules are explored more in-depth in the following subsections and include 1) musical input devices, 2) hardware-based robotic movement, and 3) software-based information processing. The implementation of these components, including the specific hardware and software choices made to meet our design, is provided in Section 3.

These three components work together to serve two primary purposes, each of which is a separate mode of operation. First of all, Dancing Robots allows instructors to input a stored music performance and highlight key musical events (e.g., a chorus, verses, phrases, etc.) with various robotic actions. Whenever any of these events occur, the robots will perform a specific routine, emphasizing the event both visually and audibly. We believe that this extra feedback will help children learn musical concepts faster, especially for children who are visual learners.

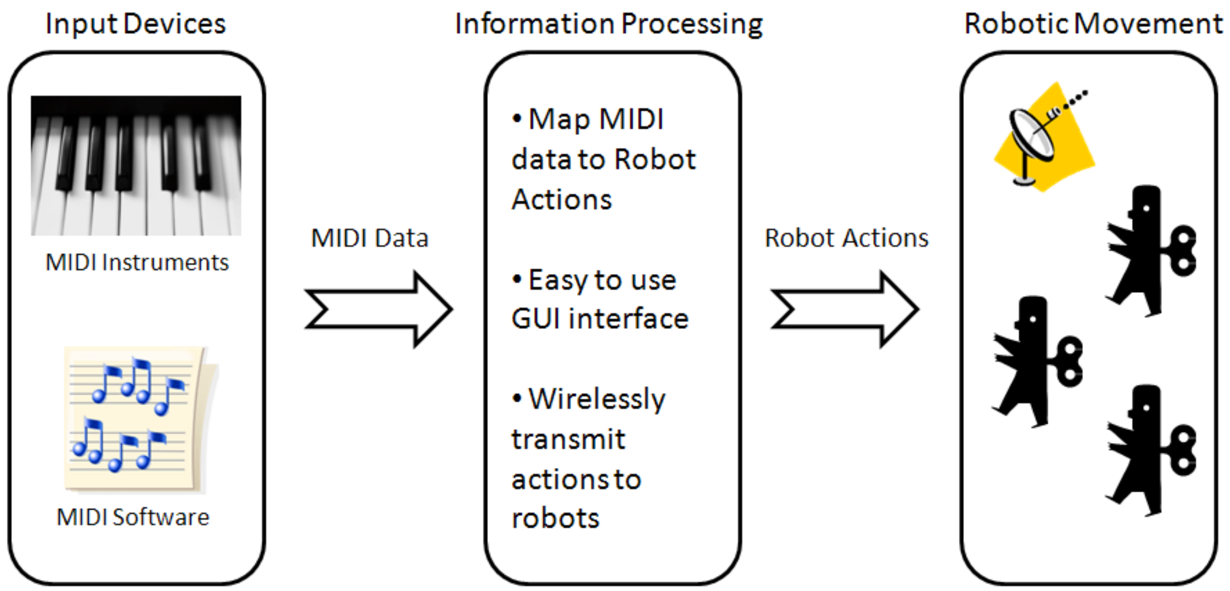


Figure 2.1: Dancing Robots! Design Components

Outside the realm of music education, this mode can also be used to create elaborate robotic dances which correspond to different movements and parts within a song.

Second, Dancing Robots can also be used in conjunction with live inputs to provide a Listen/Repeat game, similar to *Simon Says*, where an instructor plays a melody which must be played back by the children. Depending on the accuracy of the child's response, the robots will perform different routines. This mode serves to help children interact with music while providing a reward system to boost learning. This could especially be useful to piano teachers who might otherwise have a hard time encouraging children to become interested in their lessons.

2.1. Musical Input Component

The first primary component in the Dancing Robots! design involves several types of musical input devices. For the performance dancing mode, we support asynchronous inputs in the form of stored musical performances. These stored files can come from recorded performances, as well as song files created by musical software (e.g., Rosegarden [9] and Cubase [10]). To provide the Listen/Repeat game, our project also supports synchronous inputs through electronically attached musical instruments, (e.g., a digital keyboard).

When designing the musical input component of Dancing Robots!, we made the important decision that all music interpretation should be conducted through software rather than hardware. This simplifies the design, allowing us to use algorithms to decode important musical information rather than filters and other discrete electrical components. All three project members are computer engineers with stronger backgrounds in software development than electronics, so we felt it would have been a senior design project in-and-of-itself to interpret music through hardware, regardless of what we use the interpretation for. We also already need the software for the information processing component (described in Subsection 2.3), so decoding the music in this component was a natural fit in the project. Finally, not using hardware for music interpretation reduces the cost of our project and requires less power to drive the robots, enabling longer performances.

2.2. Robotic Movement Component

The next component in our design is the robotic movement component. This component consists of several anthropomorphic robots with many predefined actions across several degrees of freedom which can perform to music. They are programmable by the information processing component and can perform elaborate routines in synch with each other or independent from one another. One of the primary requirements of the robotic movement component's design is the robots must be controlled over wireless communications to avoid the clutter of wires, increase the distance they can be used from the computer hosting the software, and prevent robots from tripping over themselves while dancing.

2.3. Information Processing Component

The information processing component of our design is used as a bridge between the musical inputs to the dancing robots. It is responsible for organizing robotic actions to various portions of music, remembering and analyzing melodies for the Listen/Repeat game, playing back musical performances while the robots dance, and controlling the actions of the robots. Because this component is the primary interface to the user, we wanted it to have an easy to use graphical user interface (GUI) that both instructors and children could quickly understand and use. This includes pictures of the robots to help children know how to assign actions, as well as clearly defined buttons to control the toy.

To support both modes of operation, the information processing component consists of several key elements: 1) a module which can process data from musical inputs, 2) a song display for relaying information about a chosen song or inputted melody back to users, 3) the ability to split songs into various “segments” (e.g., chorus, verses, or other musical events) for the robots to dance within, 4) a means for assigning robotic actions to segments or accuracies of melody playback (for the Listen/Repeat game), and 5) song playback and robotic control. Since for larger songs, segmenting the song and assigning actions can become very tedious, the information processing component supports both manual choices by users and automatic assignments by the software.

Section 3: Project Implementation

In this section, we describe how we implemented each of the various components of our design while creating Dancing Robots! We begin with the musical input component, followed by the robotic movements, and end with information processing. Along the way, we also describe alternatives considered and provide a justification for their exclusion.

3.1. Musical Inputs Implementation

MIDI

To achieve both live (synchronous) and stored (asynchronous) inputs which can be processed entirely through software, we selected the MIDI (Musical Instrument Digital Interface) protocol created by the MIDI Manufacturers Association [11], which encodes musical events and relevant metadata into digital files and streams. MIDI is a well documented [12]-[14] language for music which discretizes musical events (e.g., notes on and off, pitch bends, beat changes, etc.) into sequences of bytes which can easily be decoded to interpret their meaning. This prevents us from having to interpret the songs in either the frequency or time domains in order to extract such information. This protocol is well established around the world with large libraries of stored performances for teachers to draw upon, representing all genres of music. It is also the format natively used by many electronic musical instruments and devices (e.g., digital keyboards, synthesizers, etc.), enabling such devices to be used by our Listen/Repeat game without modification.

We also considered several other alternatives when deciding upon what musical inputs should be used for Dancing Robots! We especially thought about using the popular MP3, AAC, or WMA file formats which many users already store their favorite songs in for use on their computers and portable music players (e.g., iPod or Zune). However, these formats are not discretized like MIDI, so we would have had to deal with the added overhead of extracting important information like when notes are playing and where tempos change, increasing the complexity of our project and slowing down the toys' performance. Additionally, these file formats are not supported my musical instruments, so we would have been left without the synchronous inputs required for the Listen/Repeat game.

3.2. Robotics Movements Implementation

Robosapien Robot

The robot we chose for our project was the Robosapien from WowWee toys [15]. This robot was chosen for several reasons. First of all, it is relatively cheap (see Appendix A for cost information), so it fits well with our goal of delivering an affordable toy. Second, the Robosapien has 67 available actions [16] across five degrees of freedom, so there are many predefined actions for users to take advantage of in their dances. Third, the robot already includes wireless communications in the form of IR, so adding our own wireless communications was relatively easy (see the next subsection for more details). Finally, the Robosapien has fostered a community of hobbyists and hackers, so there is a plethora of available information online for how the robots work and how to manually control them (e.g., [17], [18]). A picture of the Robosapiens in action is given in Figure 3.1.

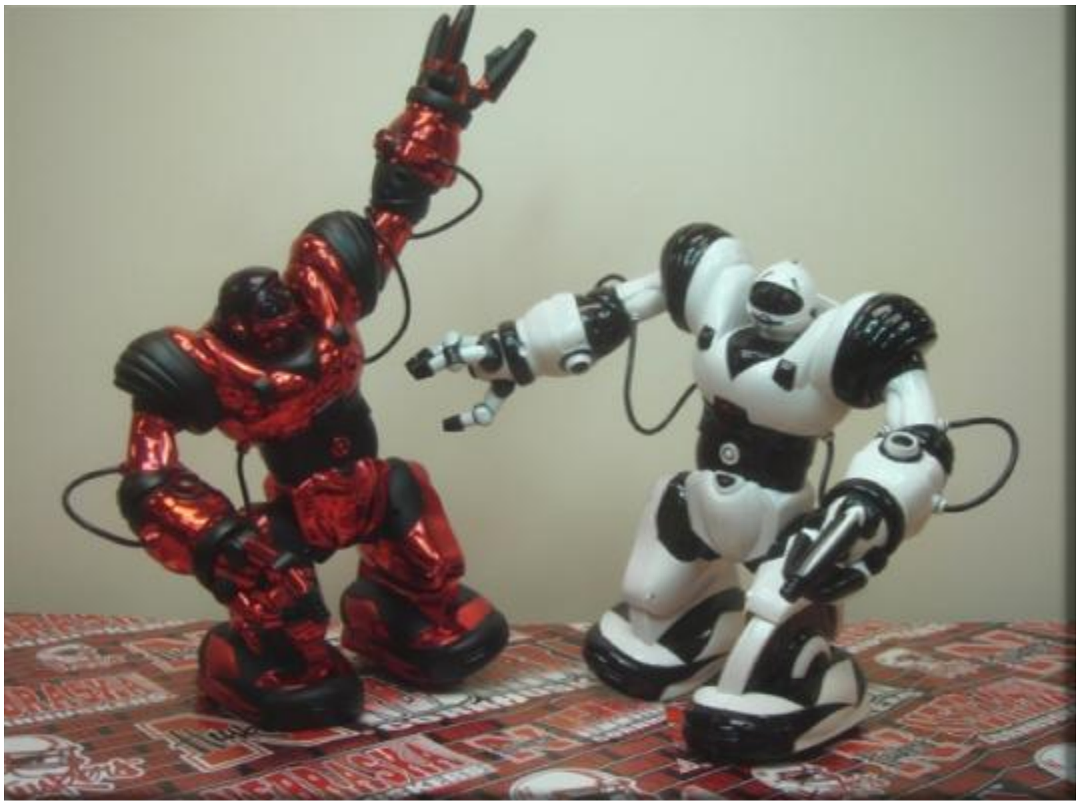


Figure 3.1: Robosapiens Dancing to Music

We also considered using other types of robots for this component of our design, including Lego robots. These toys would be a great inclusion because children could build their own robots, then watch their creations dance. The Legos would also provide a greater flexibility in what forms the robots could take on. However, these robots would have a limited number of available actions and would require extra configuration for use with the information processing component (since the actions would not be predefined). Furthermore, we wanted human-like anthropomorphic robots in our project, which would be difficult to achieve with Legos.

Bluetooth Wireless Communications

In order to control the Robosapiens over a wireless connection, we chose to use the Bluetooth protocol using the BlueSMiRF module from Sparkfun [19]. This choice was made for several reasons. First of all, we wanted a protocol which allows one device on the information processing component side to connect to multiple robots at once. Second, Bluetooth has the capability to act as a wireless serial connection without the need for additional packet processing on the receiver side. Third, Bluetooth is long range (up to 100 meters [19]) and is not line of sight, so obstructions are not a problem. Finally, Bluetooth is low power, conserving battery life on the robots.

Bluetooth was chosen over other wireless protocols, including the Robosapiens' built in infrared (IR) control. We replaced IR because that protocol is line of sight, creating problems depending on the placement of the user's computer. Additionally, each receiver receives the exact same signal, so it would be difficult to control the robots independently.

In order to interpret the Bluetooth signal sent from the information processing component to the Robosapiens, we included the Basic Stamp 2 (BS2) microcontroller [20]. This device receives a serially transmitted command over Bluetooth, then asserts the proper control sequence to the Robosapien's mainboard. This sequence is the exact same signal received over IR from the robot's remote control, and is explained in Appendix B. We tried to mimic this sequence simply using Bluetooth, since it sends serial bits, but we were unsuccessful given start and stop bits within asynchronous serial communications, hence the inclusion of the microcontroller. The control sequences outputted by the BS2 were determined using an oscilloscope connected to the robot while using the remote control, and through online sources [21]. A list of all sequences implemented, along with their timing and hex codes are provided in Appendix C. A block diagram describing the connection of the wireless communications components is provided in Figure 3.2, and a picture of their inclusion inside a Robosapien is shown in Figure 3.3.

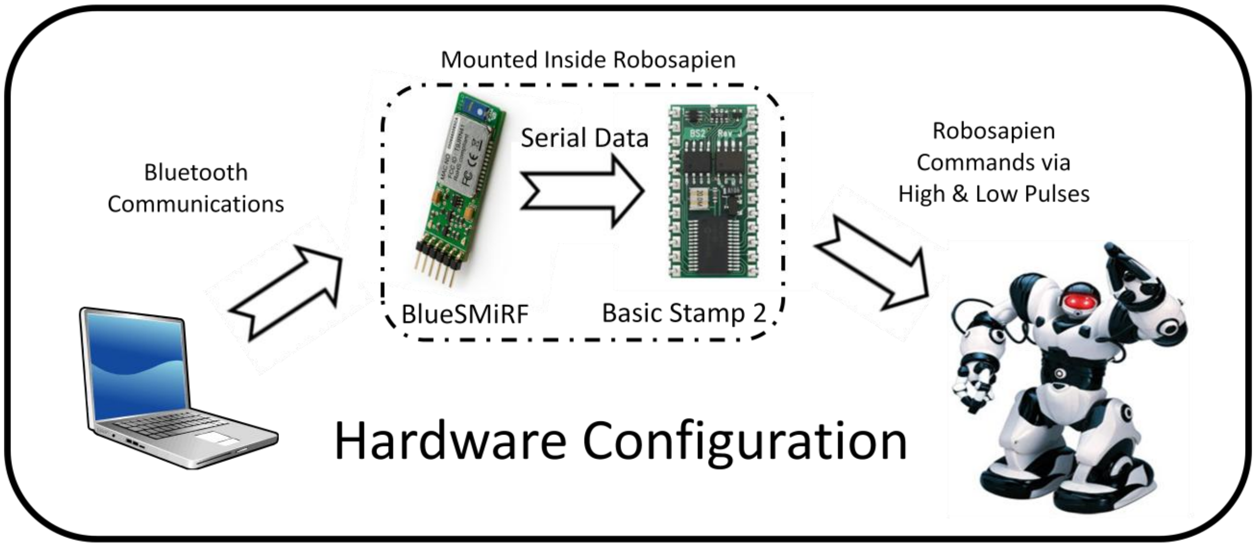


Figure 3.2.: Wireless Communications Diagram [16], [19], [20]

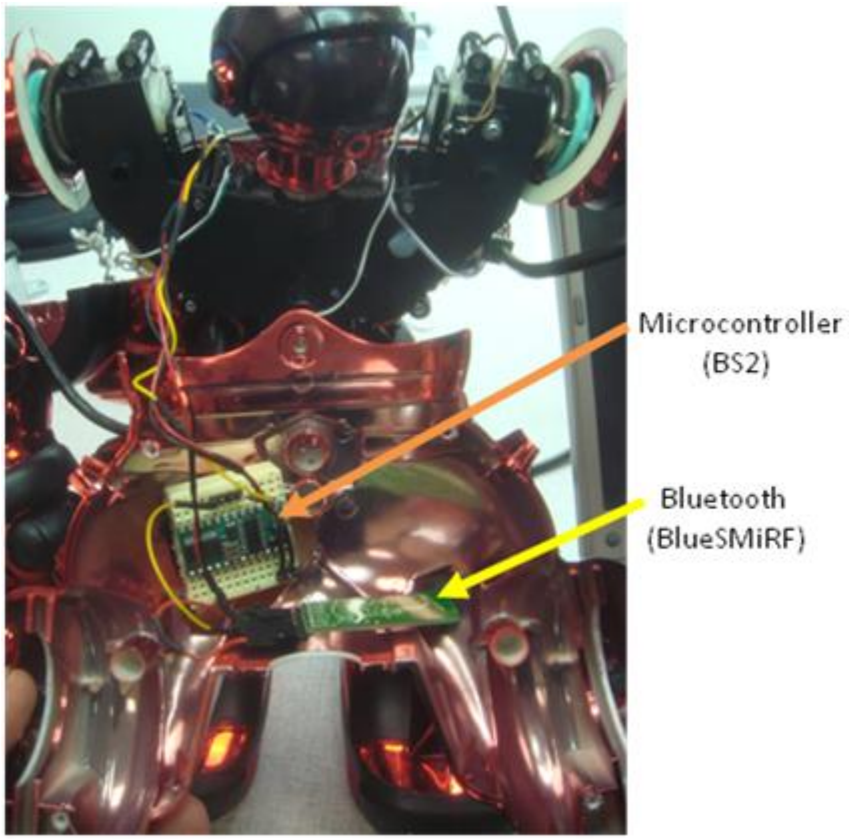


Figure 3.3: Location of Bluetooth and Microcontroller (front chest cavity of Robosapien)

3.3. Information Processing Implementation

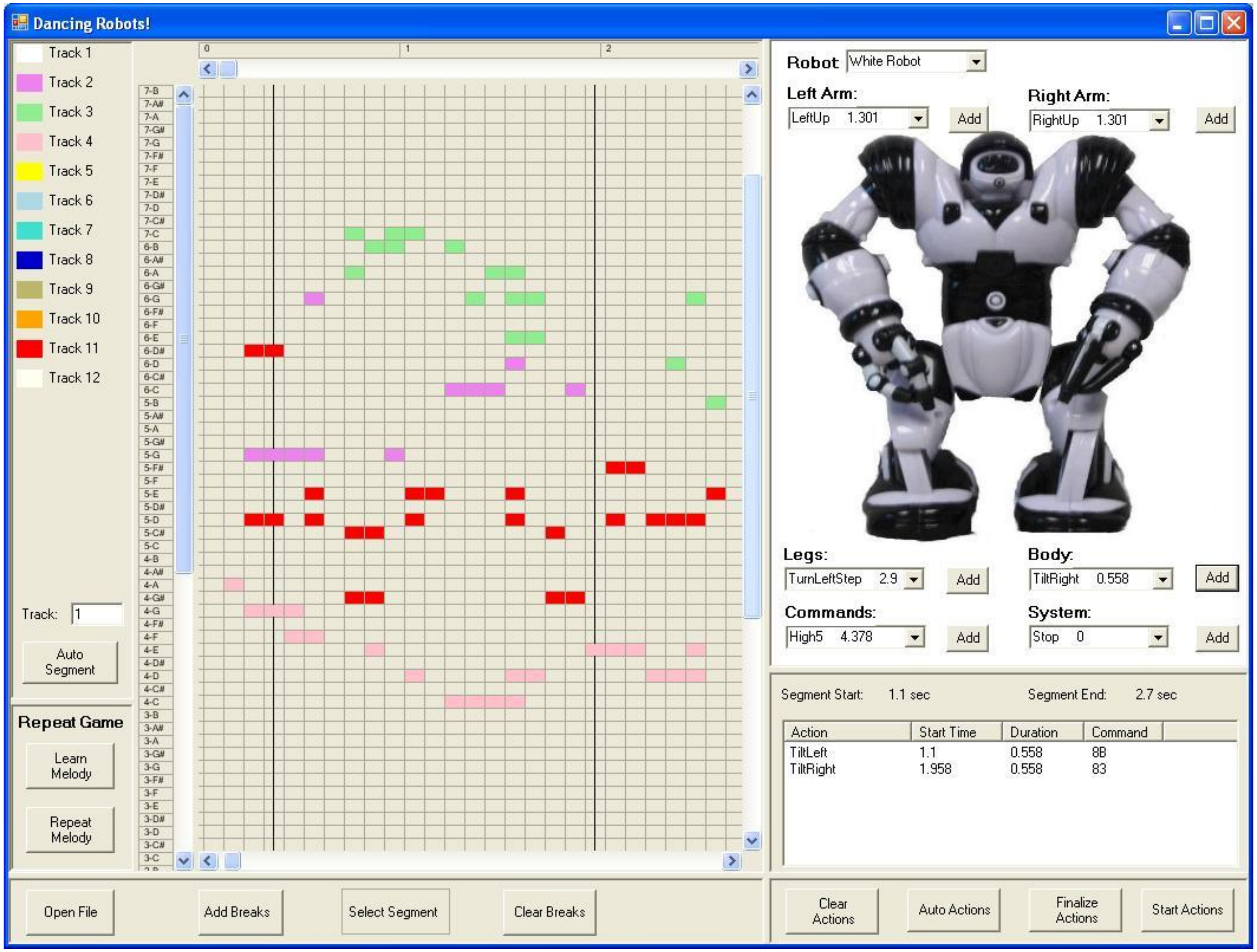
The information processing component consists of software written in C# running on a personal computer. We chose the C# programming language over alternatives like C++ or Java for several reasons. First, the design team as a whole has more experience writing for C# than other high level programming languages. Second, C# has strong support for building GUIs, which is required for our information processing component. Third, C# also has native support for threading, which was required for controlling the robotic actions and playing back songs to the users. Finally, there are many available libraries for necessary functionality, including MIDI support [22]. A screenshot of our software is provided in Figure 3.4.

Hardware Implementation

In order to use the information processing component, it must reside on hardware. This consists of a personal computer (desktop or laptop) with a sound card for playing MIDI files, USB connections to provide a Bluetooth connection to the modified Robosapien and for connecting to a MIDI device, a monitor for displaying the software, and a keyboard and mouse for user input. Based on these specifications, almost any personal computer would work and we had no problems using any of the design team's laptops.

MIDI Parser

In order to extract musical information from various inputs, the first main element of the information processing component is used to interpret raw MIDI data. This is done through a parser written by the team which reads in MIDI messages from an attached MIDI device, or extracts these messages from stored MIDI files. The messages are then converted into MIDI



Track: 1

Robot: White Robot

Left Arm: LeftUp 1.301

Right Arm: RightUp 1.301

Legs: TurnLeftStep 2.9

Body: TiltRight 0.558

Commands: High5 4.378

System: Stop 0

Segment Start: 1.1 sec

Segment End: 2.7 sec

Action	Start Time	Duration	Command
TiltLeft	1.1	0.558	88
TiltRight	1.958	0.558	83

Figure 3.4: Dancing Robots! Software GUI

events which wrap around messages to include timing information. Finally, based on tempo changes, volume settings, and timing within the data, the various musical notes are extracted from the MIDI data and saved for use by the rest of the software.

We would like to note here that all of the parser was written by the team members, but we relied on the open source C# MIDI Toolkit [22], written by Leslie Sanford, to grab MIDI message from attached devices (e.g., a digital keyboard).

Song Display

Once a series of notes have been extracted from a synchronous musical stream or a stored MIDI file, they are displayed in a piano roll format to the user, seen on the left side of Figure 3.4. This piano roll is shown as a two-dimensional table or matrix, where each row is a different pitch and each column signifies the passage of time. Higher vertical rows are higher pitches and time extends to the right on the horizontal axis. The piano roll draws each note as a contiguous sequence of cells in the table, and each track in the MIDI files are colored differently for easier readability. This format was chosen because it can display the contents of a song in an easy to understand format, and different tracks (or musical parts/instruments) can be easily distinguished.

The piano roll format was chosen over other alternatives, including sheet music and lyrics. Sheet music would be useful because it represents music in a compact form, but might be difficult for casual users and children to read, and songs with many tracks would be difficult to draw. Lyrics would be great for children because it is the words that they commonly associate with songs. However, lyrics are not often provided by MIDI files, nor are they available for synchronous inputs. Furthermore, we want Dancing Robots! to be a teaching tool, so we want children to learn something they don't already know.

Song Segmentation

When creating dancing performances to asynchronous, stored MIDI files (whose process is described by Figure 3.5), the actions performed by the robots are done to various segments within a song. These segments can represent different musical events (e.g., a chorus or verse, phrases, etc.), therefore they are worth emphasizing during teaching. Furthermore, by synchronizing actions to segments, the performances carried out by the robots are more in-synch with the music than without segments.

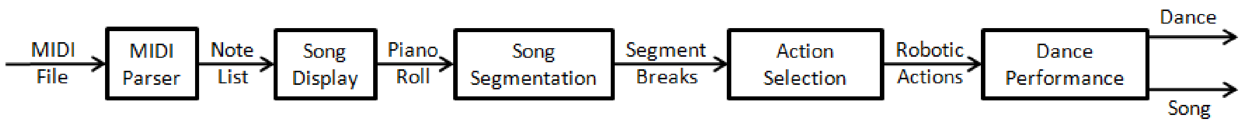


Figure 3.5: Flowchart of Robotic Dances to MIDI Files

To support both user control and flexibility, segments can either be manually chosen by users or automatically by the software. To manually create a segment, the user simply clicks on the "Add Breaks" button on the GUI, then they can click on any part of the piano roll to create a break between two segments, which is represented a vertical black line in the display. Also, if a user doesn't want to create their own segments, they can instead have the software create its own segments, based on the music in a user-specified track. This functionality uses open source code from the Melisma Music Analyzer [23] from CMU to generate a list of notes and beats in a song using the Meter program. Next, the Grouper program uses the note/beat list to find breaks between segments using a rule based system. These rules include inserting a break between long pauses in the song and keeping all segments as close to the same length as possible [24]. Once the Grouper program has finished, it returns the location of segment breaks to our software, which then stores those breaks internally and adds them to the piano roll.

Action Selection

Once a song has been segmented, different actions can be assigned for the robots to perform during the dances. This is conducted on the right side of the GUI, as shown in Figure 3.4. A picture of the Robosapien is provided as a reference, and all of the available actions are organized in combo boxes next to where on the robot they are performed. This was designed to make the GUI easier to use for children. Each action is represented by its name and duration and can be added by selecting them, then clicking the corresponding "Add" button. All actions for a selected segment (chosen by again clicking on the piano roll) are displayed to the user, along with their start times and duration, in a table below the picture of the robot. This provides a compact form for detailing robotic dances to be performed by the Robosapiens.

The actions within a segment must fit within the duration of the segment. Any actions added that violate this requirement result in a message box which alerts the user of their error. The timing values for each action, as well as the command to be transmitted over the Bluetooth wireless connection, are provided by XML configuration files read by the software. Using XML allows our software to support multiple types of robots and different command sets for each robot. For example, users could choose to limit which actions they want the robots to be able to perform. This can be done simply by editing these configuration files, which are described in Appendix D.

Each robot in the system can perform its own set of actions, so the user can select different robots to assign actions to by choosing the proper entry in the Robot combo box at the top of the display. When a different robot is chosen, the interface is updated to show the actions selected for that robot, and the picture of the Robosapien is updated to show the proper one (right now we have a white and red Robosapien).

If a user decides that they do not want to assign their own actions, they can choose to have the software automatically do so for them. To guide the automatic assignment of user actions to a set of segments, we store all action groupings performed by the robots in a MySQL database. After each performance, the software asks users whether or not they enjoyed the performance. This information is used to "learn" through reinforcement which performances users enjoy most. During automatic assignment, the groupings with the highest weights are more likely to be selected, tailoring the dances to a user's tastes. However, to introduce variability into the system, new performances are sometimes generated by the automatic action selection functionality.

Dance Performance

Once the song has been segmented and actions have been selected, the robots can now dance! The user must finalize their choices by pressing the "Finalize Actions" button, which converts the specified performance to a sequence of commands to transmit over Bluetooth, organized by when they occur. The user can then start the dance by pressing the "Start Actions" button. To allow the user to continue using the interface, the transmitting is done over a separate thread so it doesn't block other functionality. A timer is also used to start playing the song when the commands start transmission. This song playing is performed using the SequencerDemo project provided by the C# MIDI Toolkit [22].

Listen/Repeat Game

Performing the Listen/Repeat game is done a little bit differently using the software. Instead of loading a song, the teacher tells the software to listen to an inputted melody, played on an attached MIDI keyboard. Once the melody is finished (as indicated by the user), the software asks the teacher to specify what actions the robots should perform when the students are entirely correct (100%), close (>= 80%), or wrong (< 80%). Next, the students can try to repeat the melody. This input is then compared against the teacher's input and the proper dance is performed by the robots. A flow diagram describing the software elements used during the Listen/Repeat game is provided in Figure 3.6.

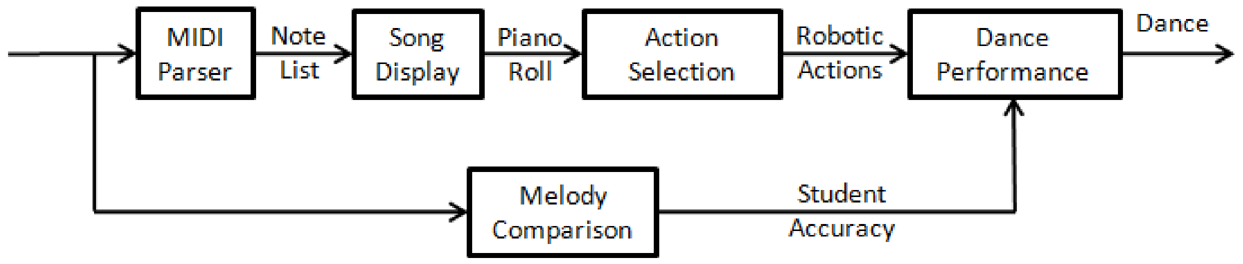


Figure 3.6: Flowchart for Listen/Repeat Game

Section 4: Conclusions

4.1. Project Summary

In conclusion, we have created an educational toy called Dancing Robots! which promotes musical education and technology to young children. Using the anthropomorphic Robosapien robot modified to use a Bluetooth wireless connection, Dancing Robots! allows children to interact with music using MIDI instruments and receive a reward for their efforts in the form of a robotic dance. Teachers can also use the segmentation feature of the Dancing Robots! software to highlight key musical events, helping students learn important musical features, such as what a chorus or verse is, or when a song exhibits vibrato or a sequence of staccato notes.

While the prototypes were a little pricy at $140 (see Appendix A for a cost analysis), we believe that in mass production while working with WowWee, this price could be greatly reduced to offer an affordable product to schools and homes. If available, we believe that such a product could motivate students to learn about music, increasing their skills in learning, reasoning, and retention of information.

For more detailed information about our design and implementation and a description of our testing procedures and results, please refer back to our previous progress reports and original proposal. These are available online at our project Wiki

<http://csce.unl.edu/wiki/index.php/CSCE489_Dancing_Robots>

along with our team meeting log and Adam's design log. The design log also includes inline links to most of our references, as well as programming resources and a datasheet for the BlueSMiRF (mostly unused during the project).

4.2. Future Work

To further improve Dancing Robots!, we have several ideas for future work that we would like to see be conducted: 1) improve connections between hardware components, 2) use a better microcontroller, 3) provide better control over robotic actions, and 4) use Dancing Robots! for a new form of performance art.

Improve Connections

First, we encountered several problems with connecting our software to both a MIDI keyboard and the Robosapiens. Whenever the robots respond to a child's playback of the melody, only one robot dances. However, both robots work fine up until this point if they are just dancing to stored MIDI files. Both of these connections rely on USB devices, so we think there is a conflict somewhere in the USB layer. Additionally, the Bluetooth connection was sometimes intermittent between the computer and robots. We used a pretty cheap USB dongle provided by the Computer Science and Engineering Department, so the problem could just be the quality of our parts. Fixing these problems would result in a more stable toy for children to enjoy.

Better Microcontroller

Second, we would like to see a better microcontroller used in the Dancing Robots! We chose the BS2 because it was cheap, available through the EE Shop on campus, and it offered a built in library for serial communications. However, the device uses a lot of our power (battery life is cut by 66% from 6 hours to around 2 hours when using the BS2), causing us to go through many sets of expensive D batteries. The BS2 does not support interrupts, so we had to use polling to listen for incoming Bluetooth commands. A microcontroller with interrupt support would greatly cut down on the power consumption and battery drain. In fact, if we were to ever produce this product, the mainboard of the Robosapien could be replaced by a Bluetooth-enabled microcontroller which contains all of the functionality we require.

Better Control over Robotic Actions

Third, right now we are only using the built in actions in the Robosapiens. Unfortunately, only one action can be performed at once, and some actions take several seconds to complete. This is why our design only has robots dance to segments of songs instead of individual music events – the robots are just not responsive enough to do anything else. However, the Robosapien has all of its motor servos exposed, so we could control the motors themselves directly using the microcontroller instead of mimicking the IR signal through its current mainboard. This would give us a wider range of available actions, and it would also make the robot more responsive, making it possible for them to directly respond to music.

Performance Art

Finally, Dancing Robots! represents a relatively new form of performance art – robots dancing to music. We would like to see how this project could be extended beyond the education domain to performance arts. This could create a whole new genre of dance and as technology advances, exciting new possibilities could occur at the intersection of music and robots.

References

[1] Hetland, L. Learning to make music enhances spatial reasoning. *Journal of Aesthetic Education*, Vol. 34(3-4), 2000, pp 179-238.

[2] Schellenberg, E.G. Music lessons enhance IQ. *Psychological Science*, Vol. 15(8), Aug. 2004, pp. 511-514

[3] Rauscher, F.H., and Zupan, M.A. Classroom keyboard instruction improves kindergarten children's spatial-temporal performance: a field experiment. *Early Childhood Research Quarterly*, Vol. 15(2), 2000, pp. 215-228.

[4] Hasbro, "Tiger Electronics", available at <http://www.hasbro.com/tiger/default.cfm>, accessed on October 21, 2007.

[5] Michalowski, M. and Kozima, H., "Keepon & the BeatBots", available at <http://beatbots.org/>, accessed on October 21, 2007.

[6] Tanaka, F., Fortenberry, B., Aisaka, K., and Movellan, J.R. Developing dance interaction between QRIO and toddlers in a classroom environment: plans for the first steps, in *Proceedings of the 2005 IEEE International Workshop on Robots and Human Interactive Communication (RO-MAN '05)*, Nashville, TN, Aug. 13-15, 2005, pp. 223-228.

[7] Tanaka, F., Cicourel, A., and Movellan, J.R. Socialization between toddlers and robots at an early childhood education center, *Proceedings of the National Academy of Sciences*, Vol. 104(46), Nov. 13, 2007, pp. 17954-17958.

[8] Lowe, S. Sony soon to deliver child robot, *The Age*, Dec. 23, 2002, available online at <http://www.theage.com.au/articles/2002/12/22/1040510966660.html>, accessed on April 28, 2008.

[9] Rosegarden, "Rosegarden: music software for Linux", available at <http://www.rosegardenmusic.com/>, accessed on October 21, 2007.

[10] Steinberg, "Cubase 4 :: Steinberg Media Technologies GmbH", available at <http://www.steinberg.net/983_1.html>, accessed on October 22, 2007.

[11] MIDI Manufacturers Association, "MIDI Manufacturers Association", available at <http://www.midi.org/>, accessed on October 21, 2007.

[12] Glatt, J., "MIDI Technical/Programming Docs", available at <http://www.borg.com/~jglatt/tech/miditech.htm>, accessed on November 21, 2007.

[13] Harmony Central, “Harmony Central – MIDI Documentation”, available at <http://www.harmony-central.com/MIDI/Doc/>, accessed on November 23, 2007.

[14] Sapp, C.S., “Variable Length Values”, available at <http://www.ccarh.org/courses/253/handout/vlv/>, accessed on November 21, 2007.

[15] WowWee Ltd., “Welcome to the Official Robosapien site”, available at <http://www.wowwee.com/robosapien/robo1/robomain.html>, accessed on December 2, 2007.

[16] WowWee Ltd., “Wowwee Roboitics Robosapien”, available at <http://www.wowweestore.com/index.asp?PageAction=VIEWPROD&ProdID=2>, accessed on December 3, 2007

[17] “ROBOSAPIEN.tk - The unofficial Robosapien Hacks and Mods site”, available at <http://home.planet.nl/~pruim006/index2.htm>, accessed on November 17, 2007.

[18] Brown, Chance. “BasicX-24”, available at < http://www.evosapien.com/robosapien-hack/knitsu/html/basicx-24.html>, accessed on February 22, 2008.

[19] Sparkfun, “Bluetooth Modem -BlueSMiRFGold” , available at <http://www.sparkfun.com/commerce/product_info.php?products_id=582 >, accessed on February 3, 2008

[20] Parallax, Inc., “BASIC Stamp 2 Module”, available at <http://www.parallax.com/Store/Microcontrollers/BASICStampModules/tabid/134/CategoryID/9/List/0/SortField/0/Level/a/ProductID/1/Default.aspx>, accessed on February 18, 2008

[21] “[AiboHack] RoboSapien IR Codes”, available at <http://www.andrew.cmu.edu/user/ebuehl/robosapien-lirc/ir_codes.htm>, accessed on February 3, 2008.

[22] The Code Project, “CodeProject: C# MIDI Toolkit”, available online at <http://www.codeproject.com/KB/audio-video/MIDIToolkit.aspx>, accessed on March 28, 2008.

[23] Sleator, D. and Temperley, D. “The Melisma Music Analyzer”, available online at <http://www.link.cs.cmu.edu/music-analysis/>, accessed on March 23, 2008.

[24] Thom, B., Spevak, C, and Hoethker, K. 2002. Melodic segmentation: evaluating the performance of algorithms and musical experts, in *Proceedings of ICMC’02*, Goeteborg, Sweden, Sept. 2002.

Appendix A: Cost Information

The idea behind Dancing Robots! may be solid, but it means nothing if buyers cannot afford to purchase them. The prototypes for Dancing Robots! were built from a combination of readily available parts that were relatively inexpensive. Table A.1 shows the actual cost of each prototype as 140 US dollars. As mentioned in the Future Work (Subsection 4.2) the main controller board of the Robosapien could easily be modified as to include the needed Bluetooth functionality. Working with WowWee on design changes would also allow for added functionality and improvements to be added. Judging from the fact that WowWee is easily able to sell these robots for 50 US dollars, these small changes would probably not tip the scales. As with any product that is ramped up to mass production, there is a bulk discount. Taking advantage of this discount, our team projects that the cost to produce a Robosapien with all functionality could be lowered to around $65. This results in a total projected cost of $70 if batteries are included. This low price would allow a great market price of around $100. Dancing Robots! may soon be in a store near you!

Table A.1: Cost Analysis for Dancing Robots!

Component	Our Cost	Projected Cost
Robot (Robosapien)	$50	$65
Bluetooth (BlueSMiRF)	$65	-
Microcontroller (BS2)	$20	-
Batteries ((4) D cells)	$5	$5
	$140	**$70**

Appendix B: Robosapien Control Sequence

A command is issued to the Robosapien from our BS2 microcontroller as follows. Each command starts with a sequence of 0's (eight to be exact), each 1 bit in the original command is actually sent as 1110, and each 0 bit is sent as 10, all asserted at 1200 baud. An example command and its attached waveform are shown below in Figure B.1.

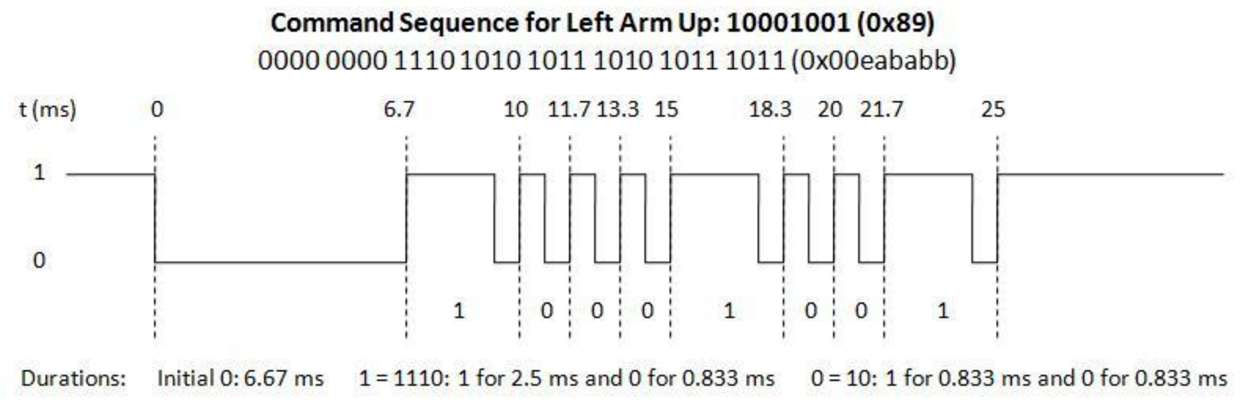

Command Sequence for Left Arm Up: 10001001 (0x89)
0000 0000 1110 1010 1011 1010 1011 1011 (0x00eababb)
t (ms)
0
6.7
10
11.7
13.3
15
18.3
20
21.7
25
1
0
1
0
0
0
1
0
0
1
Durations:
Initial 0: 6.67 ms
1 = 1110: 1 for 2.5 ms and 0 for 0.833 ms
0 = 10: 1 for 0.833 ms and 0 for 0.833 ms

Figure B.1: Command Sequence for Left Arm Up

Appendix C: Robosapien Commands with Timing

Table C.1: Robosapien Commands with Timing

Hex Code	Command	Work?	Delay (S)	Delay (S)
80	Turn Right	Yes		
81	Right Arm Up	Yes	0.772	1.301
82	Right Arm Out	Yes	0.820	0.504
83	Tilt Body Right	Yes	0.558	
84	Right Arm Down	Yes	0.873	0.343
85	Right Arm In	Yes	0.823	0.397
86	Walk Forward	Yes		
87	Walk Bakcward	Yes		
88	Turn Left	Yes		
89	Left Arm Up	Yes	0.772	1.301
8a	Left Arm Out	Yes	0.820	0.504
8b	Tilt Body Left	Yes	0.558	
8c	Left Arm Down	Yes	0.873	0.343
8d	Left Arm In	Yes	0.823	0.397
8e	Stop	Yes		
a0	Right Turn Step	Yes	2.969	
a1	Right Hand Thump	Yes	1.885	
a2	Right Hand Throw	Yes	3.053	
a3	Sleep	Yes		
a4	Right Hand Pick-Up	Yes	2.202	
a5	Lean Backward	Yes	0.453	
a6	Forward Step	Yes	2.005	
a7	Backward Step	Yes	2.005	
a8	Left Turn Step	Yes	2.969	
a9	Left Hand Thump	Yes	1.885	
aa	Left Hand Throw	Yes	3.053	
ac	Left Hand Pick-Up	Yes	2.202	
ad	Lean Forward	Yes	0.453	
b1	Wakeup	Yes		
c0	Right Hand Strike 3	Yes	2.678	
c1	Right Hand Sweep	Yes	1.618	
c3	Right Hand Strike 2	Yes	4.452	
c4	High Five	Yes	4.378	
c5	Right Hand Strike 1	Yes	3.107	

c8	Left Hand Strike 3	Yes	2.678	
c9	Left Hand Sweep	Yes	1.618	
cb	Left Hand Strike 2	Yes	4.452	
cd	Left Hand Strike 1	Yes	3.107	
d6	Karate Chop	Yes	2.683	
f6	Feet Shuffle	Yes	2.735	
fc	Raise Arm Throw	Yes	3.036	
ab	Listen	Not		
ae	Reset	Not		
c2	Burp	Not		
c6	Bulldozer	Not		
c7	Oops	Not		
ca	Whistle	Not		
cc	Talkback	Not		
ce	Roar	Not		
90	Master Program	Not		
91	Play	Not		
92	Right Program	Not		
93	Left Program	Not		
94	Sonic Program	Not		
98	Quiet Execute	Not		
9a	Quiet Execute with Subroutines	Not		
b0	Master Execute	Not		
b2	Right Execute	Not		
b3	Left Execute	Not		
b4	Sonic Execute	Not		
d0	All Demo	Not		
d1	Power Off	Not		
d2	Demo 1	Not		
d3	Demo 2	Not		
d4	Dance	Not		
fb	Nothing	Not		

Appendix D: XML Configuration Files

D.1. Bluetooth Settings

This configuration file determines whether or not we are using a robot and which COM port its Bluetooth is connected to. Here, we wrote this without considering the colors of our robots, so White is the white robot and Black is the red robot.

```
<?xml version="1.0" encoding="utf-8" ?>
<Settings>
	<White>
		<Conn>COM5</Conn>
		<Use>True</Use>
	</White>
	<Black>
		<Conn>COM7</Conn>
		<Use>True</Use>
	</Black>
</Settings
```

D.2. Command Settings

The command settings configuration file defines all of the actions available to the robots, groups them by category, and provides their commands to be transmitted over Bluetooth. The file is organized as follows: the root is the type of robot, the branch is the action category, and the leaves are the actions with values equal to their commands.

```
<?xml version="1.0" encoding="utf-8" ?>
<Robosapien>
        <LeftArm>
                <LeftUp>89</LeftUp>
                <LeftOut>8A</LeftOut>
                <LeftDown>8C</LeftDown>
                <LeftIn>8D</LeftIn>
                <LeftThump>A9</LeftThump>
                <LeftThrow>AA</LeftThrow>
                <LeftPickup>AC</LeftPickup>
                <LeftStrike1>CD</LeftStrike1>
                <LeftStrike2>CB</LeftStrike2>
                <LeftStrike3>C8</LeftStrike3>
                <LeftSweep>C9</LeftSweep>
        </LeftArm>
        <RightArm>
                <RightUp>81</RightUp>
                <RightDown>84</RightDown>
                <RightIn>85</RightIn>
                <RightOut>82</RightOut>
                <RightThump>A1</RightThump>
                <RightThrow>A2</RightThrow>
                <RightPickup>A4</RightPickup>
                <RightStrike1>C5</RightStrike1>
                <RightStrike2>C3</RightStrike2>
                <RightStrike3>C0</RightStrike3>
                <RightSweep>C1</RightSweep>
        </RightArm>
        <Body>
                <TiltLeft>8B</TiltLeft>
                <TiltRight>83</TiltRight>
                <LeanBackward>A5</LeanBackward>
                <LeanForward>AD</LeanForward>
```

```
	</Body>
	<Legs>
		<TurnLeftStep>A8</TurnLeftStep>
		<TurnRightStep>A0</TurnRightStep>
		<ForwardStep>A6</ForwardStep>
		<BackwardStep>A7</BackwardStep>
		<FeetShuffle>F6</FeetShuffle>
	</Legs>
	<Commands>
		<High5>C4</High5>
		<RaiseArmThrow>FC</RaiseArmThrow>
		<KarateChop>D6</KarateChop>
	</Commands>
	<System>
		<Stop>8E</Stop>
		<Sleep>A3</Sleep>
		<Wakeup>B1</Wakeup>
	</System>
</Robosapien>
```

D.3. Timing Settings

The timings settings configuration file defines the longest duration each of the actions in the command settings configuration file take to execute.

```
<?xml version="1.0" encoding="utf-8" ?>
<Robosapien>
        <LeftArm>
                <LeftUp>1.301</LeftUp>
                <LeftDown>0.873</LeftDown>
                <LeftIn>0.823</LeftIn>
                <LeftOut>0.820</LeftOut>
                <LeftThump>1.885</LeftThump>
                <LeftThrow>3.053</LeftThrow>
                <LeftPickup>2.202</LeftPickup>
                <LeftStrike1>3.107</LeftStrike1>
                <LeftStrike2>4.452</LeftStrike2>
                <LeftStrike3>2.678</LeftStrike3>
                <LeftSweep>1.618</LeftSweep>
        </LeftArm>
        <RightArm>
                <RightUp>1.301</RightUp>
                <RightDown>0.873</RightDown>
                <RightIn>0.823</RightIn>
                <RightOut>0.820</RightOut>
                <RightThump>1.885</RightThump>
                <RightThrow>3.053</RightThrow>
                <RightPickup>2.202</RightPickup>
                <RightStrike1>3.107</RightStrike1>
                <RightStrike2>4.452</RightStrike2>
                <RightStrike3>2.678</RightStrike3>
                <RightSweep>1.618</RightSweep>
        </RightArm>
        <Body>
                <TiltLeft>0.558</TiltLeft>
                <TiltRight>0.558</TiltRight>
                <LeanBackward>0.453</LeanBackward>
                <LeanForward>0.453</LeanForward>
        </Body>
        <Legs>
                <TurnLeftStep>2.969</TurnLeftStep>
                <TurnRightStep>2.969</TurnRightStep>
        <ForwardStep>2.005</ForwardStep>
        <BackwardStep>2.005</BackwardStep>
        <FeetShuffle>2.735</FeetShuffle>
    </Legs>
    <Commands>
        <High5>4.378</High5>
        <RaiseArmThrow>3.036</RaiseArmThrow>
        <KarateChop>2.683</KarateChop>
    </Commands>
    <System>
        <Stop>0</Stop>
        <Sleep>0</Sleep>
        <Wakeup>0</Wakeup>
    </System>
</Robosapien>
```