SOLEBEAT

Put Some Beat In Your Feet

The SoleBeat Pedometer/Audio System

Created by Chip Snyders, Brian Geistwhite, and Nick Macuinas

Introduction

Studies have shown that listening to music enhances motivation and makes exercising a more enjoyable experience[1,2]. However, runners often find that it is hard to create playlists that will correspond to their workouts or to adjust their music mid-run.

This report chronicles our fabrication of a cadence-sensing shoe that enables pace-matched song selection with dynamic tempo adjustment while simultaneously keeping track of pedometric and chronometric information. Our technology utilizes an FSR embedded in the heel of the right shoe to detect strides, as well as an FSR that can be fastened to the users' clothing to allow the user to change the song itself or the current song's tempo in accordance with the user's pace changes.

Design Goals

1) Play one of many songs that corresponds to the wearer's running cadence:
   The shoe/stamp board system will output the shoe's FSR data to a local computer. The computer will process consecutive inputs to generate a stride tempo (steps/minute) and play the song with a tempo closest to that of the stride. The song's tempo will then be dynamically adjusted to exactly match the stride tempo. When stride tempo is closer to a different song's tempo and a signal is given by the user, the song being played is changed. During a song change, there is a brief fadeout period to prevent thrashing between the songs (attributable to ChucK).
2) Modulate a song's speed to correspond precisely with the wearer's running cadence:
   The local computer will gather data and produce a continuously updated tempo as described above. As aforementioned, within the tempo bounds, the song's tempo will be adjusted in accordance with running tempo.
3) Afford user control of song playing/tempo changes.
   The computer will also accept data from another FSR attachable to the user that acts as a switch, allowing the user to start/stop music play and to select their current cadence to be used as the tempo that music plays at.
4) Collect pedometric and chronometric information
   The local computer will continuously track the number of strides and the duration of the run.

---

[1] http://www.runtheplanet.com/trainingracing/training/cadence.asp,

[2] http://www.nytimes.com/2008/01/10/fashion/10fitness.html?_r=2&oref=slogin&oref=slogin

The User

Our technology is intended for use by runners of any level. While casual runners will be drawn to the motivational aspect of pace-synchronized song play, competitive runners can benefit from the performance boost provided by consistent pacing. It is well known that setting and maintaining a running pace is crucial to optimizing speed during a race.

Equipment and Materials

2 square FSRs.
1 BASIC stamp. We're using Sniff, which was taken from the HCI lab.
1 MIDISport 2x2 MIDI signal switch. Lifted from the Sound Lab.
1 MIDI connector cable. Lifted from the Sound Lab.
1 regular-USB-to-fat-little-squarish-USB-end connector. Lifted from the Sound Lab.
1 10-pin ribbon cable.
Plenty of extra long multicolored wire for device connection.
Electrical solder
1 tic-tac container (for providing a hard base for the FSR)
Electrical tape and duct tape.

Software specifications

PBASIC code: Running on the Basic Stamp, takes in FSR output and analyzes it to detect footfalls and music change requests and sends MIDI signals to our system's laptop that indicate the occurrence of these events. [for actual code, see **Appendix A**]

ChucK code: Running on a laptop, interprets the MIDI event signals from the Basic stamp in order to assess duration of run, record the number of footfalls (a pedometer function), and switch music when the user indicates that she/he wants a change. [for actual code, see **Appendix B**]

Software Design

Our PBASIC code continuously monitors FSR output, and uses very basic finite state automata to create footfall events and controller FSR events. Each FSR has a “high threshold” output value. When a FSR’s automaton is the default state, meeting or exceeding this value causes an event signal to be sent and a the automaton changes to an event state. Afterwards, the detection of a FSR output below a “low threshold” output value will result in the FSR’s automaton to change back to the default state. Event signals are only sent once per default/event state cycle, allowing the device to correctly identify exactly one event per physical event. The program was initially designed such that the detection of multiple successive outputs above/below a threshold value without an “interrupting” output below/above the opposite threshold would signal the change from the default state and the event state. This design decision was made as a ward

against chaotic FSR output, but proved buggy and not useful for the pressure signals that we received.

The ChucK code calculates cadence information by taking the system time difference between footfalls to generate a target song Beats Per Minute (BPM). A database of songs with their actual BPM is maintained2. When a controller FSR event is received, a song with the actual BPM most similar to the target BPM is selected. Its play rate is multiplied by (target BPM/actual BPM) to match the song's tempo precisely with the runner's cadence; usually, .9 < multiplier < 1.1, resulting in minor song pitch distortion. The ChucK code also tracks the total number of footfall events, and the total duration of the run.

Fabrication

Our first step was getting the shoe ready for an embedded FSR. To this end, we cut out a section of the insole of the shoe near the rear of the shoe (below the heel). We then took a tic-tac box, cut out the shorter sides of the box (preserving the two largest areas, the face and the rear). After wrapping them in duct tape to prevent cracking, we placed these where the insole had been removed. This provided a hard base for the FSR to rest on. We then attached the FSR to this and placed the cut out insole on top of the FSR, providing nearly the same amount of cushioning that was available in the original shoe. We then cut a slit in the back of the shoe so that we could run wires from the FSR out to the Basic Stamp board.

We also attached two safety pins to an FSR using tape (this serves as the music changing controller). The FSR was to be fastened onto the user's shirt in the chest area.

We soldered insulated wires to the FSRs and then, after cutting off the IDCs at the end of the ribbon cables, soldered those wires into a single ribbon cable that we would later hook up to the Basic Stamp.

Our Basic Stamp fed MIDI signals to a MIDIbox, which in turn relayed the information via USB to a laptop.

See **Appendix C** for a block diagrams, schematics, and photos of the technology.

Group Responsibilities

Chip Snyders|*ChucK programming and a little bit of everything else*

Chip handled the ChucK coding, ensuring that all of the signals that came out of the MIDIbox were processed properly and that music played smoothly and matched the runner's pace. Chip also helped to attach the loose wiring to a belt in order to make the device useable, among other things like taping and designing.

Brian Geistwhite|*PBASIC Coding, Calibration, and Wiring*

Brian worked on the PBASIC coding (which entailed designing a program that recognized footfall or control FSR signatures and output MIDI messages accordingly) and analyzed FSR outputs in order to calibrate the code to properly register footfalls from FSR fluctuations. He also performed all of the soldering in the project, and connected a lot of wires.

Nick Maciunas|*Basic Coding, Wiring, and Shoe Fabrication*

Nick worked on the PBASIC coding with Brian, helping perform the functions listed above. He cut up the shoe, installed the FSRs, and taped it up again. In addition, he helped with wiring and other assorted tasks.

Testing Phases

Initial Testing:

Our initial stage of testing was done in the lab before we had mounted the FSRs. We ensured that our software functioned properly when the FSRs were simply connected to the Basic Stamp.

Basic Testing:

This testing involved our group using the device in various contexts in order to get it operating correctly, i.e. calibrating the tempo. We watched for this performance by watching/graphing printed output of the inner workings of the SoleFedge system and by actually using the SoleBeat and ensuring that it operated as expected. We were able to do this by running in place, or by simply tapping on the FSRs. This allowed us to ensure that our software correctly registered signals from the FSRs and MIDI signals from the Basic Stamp, and that our software correctly converted those signals into properly paced songs.

Once we were satisfied with our device's functionality in the controlled setting of the room, we moved on to our actual testing phase.

Outdoor Testing:

We then took the shoe outside for more in depth testing. One of our group members would sit in a swivel chair in an open area with a laptop, basic stamp, and MIDIbox on his lap. The test subject was placed in the shoe and the music changing sensor was pinned to the subject's upper chest. We then had the subject run around the swivel chair as our group member rotated with the runner.

Our two most important criteria for correct operation were (1) consistent registration of one footfall signal per physical runner footfall, and (2) that song and tempo switches occur precisely in the manner as our algorithm outlines.

In order to test this, we verbally counted the number of footfalls over multiple trial periods of one minute, and compared that value to the number recorded by ChucK and the Basic Stamp outputs.
We also used the following tests to determine algorithm correctness and robustness:

a) Controller FSR pressed for the first time while the wearer is standing still, and then pressed again while running at a moderate pace
b) Controller FSR pressed for the first time while the wearer is running unnaturally slowly, and then pressed again while running at a moderate pace

c) Controller FSR pressed for the first time while the wearer is running at a moderate rate, and then pressed again while running at a moderate pace
d) Controller FSR pressed for the first time while the wearer is stepping at an unnaturally fast rate, and then pressed again while running at a moderate pace
e) Controller FSR pressed a number of times while a consistent, unnaturally fast running pace is maintained.
f) Controller FSR pressed a number of times while a consistent, moderate running pace is maintained.
g) Controller FSR pressed a number of times while a consistent, unnaturally fast running pace is maintained.
h) Pace goes from a moderate pace with moderate tempo music to an unnaturally slow pace without controller FSR being pressed.
i) Pace goes from a moderate pace with moderate tempo music to an unnaturally fast pace without controller FSR being pressed.

Treadmill Testing:

Our treadmill testing was essentially the same as our outdoor testing, but primarily was completed as a precursor to our official demonstration. We walked through the same algorithm correctness tests and checked for footfall detection consistency.

Potential Further Testing:

We had initially anticipated on running a series of tests on a decently sized subset of people. These tests were designed to see if performance and enjoyment were enhanced by the SoleBeat system. After actually constructing our shoe, we realized that such a series of tests was impractical, given our time and budget constraints, for a number of reasons.

The first reason is that the materials that we used for our shoe were simply not durable enough to be used for a very extended period of time. Since our system was not customized for different user sizes (both of foot and height), the stresses placed on the system were not uniform. As a result, soldering occasionally broke (despite heavy wrapping in electrical tape) and FSR calibration was often required on treadmills because of unwieldy FSR outputs. Additionally, we only had one pair of shoes to work with, and cleanliness of the shoe became an issue with prospective test subjects.

However, given the success of our device in functionality testing and the level of positive feedback that came from viewers during portions of our testing and our trial run, we suspect that studied feedback would also have been similarly positive.

Once the system is working to our specifications, we will test the SoleBeat with real runners, testing (1) if the system is desirable and (2) if running performance is affected.

The participants will first take a survey:

1) Do you run? If so, how many miles per week, on average?
2) Do you listen to music while running?
3) If you had the choice to listen to music while running, would you?

The participants will then spend five minutes running on the treadmill without any music, five minutes listening to music on an i-Pod with the same selection of songs as are implemented on the SoleBeat, and five minutes using SoleBeat system. The order in which they do these three activities will be varied for each participant so as the balance different permutations.

Testing running performance is still tentative. The user's selection of treadmill pace will be discretely monitored during each phase of the experiment. Our policy will be to not give the runner any advice on what pace settings to use. Since the order in which these activities are done will be "randomized," we should be able to detect any tendency of the runner to increase/decrease tempo without instruction.

After running, participants will take another (followup) survey:

> 1) Please rank your satisfaction with running without music, running with an iPod, and running with the SoleBeat (for this question, we will randomly order the three possibilities on each survey):
>
> *WITHOUT MUSIC*
> Completely Dissatisfied Completely Satisfied
> 1 2 3 4 5 6 7
> *IPOD*
> Completely Dissatisfied Completely Satisfied
> 1 2 3 4 5 6 7
> *SoleBeat*
> Completely Dissatisfied Completely Satisfied
> 1 2 3 4 5 6 7
>
> 2) Did you feel that the SoleBeat changed your running experience? In what way?
> 3) What features of the SoleBeat did you like?
> 4) What features of the SoleBeat did you dislike?
> 5) What additional features do you think would augment the SoleBeat?

The method of participant selection will be made with the importance of sample randomization and symmetry in mind.

Results

Note: All FSR outputs are measured on a scale of 0-255, where 0 corresponds to the lowest voltage input, and 255 corresponds to the highest

Initial Testing: This testing phase was done informally. The FSRs were found to give consistent and clear responses to pressure inputs. We found that the controller FSR would show outputs well below 100 without pressure, and outputs well above 100 when hit gently with a closed fist. The foot FSR would show outputs at around the maximum value when heel significant pressure was applied to it, and noticeably lower outputs otherwise. This information was reliable enough that we successfully calibrated the system to accurately detect each footfall and control FSR press. This information was transferred to a Macintosh running the ChucK portion of SoleFedge via Midi messages, which required only minor debugging before the system seemed to operate correctly.

Stepping on the FSR while seated or jogging in place indoors, having a second person press activate the control FSR, the system successfully started playing music, modulated individual song tempo within a short range when "running" cadence changed slightly, switched from song to song when "running" cadence changed significantly, and seemed to played songs at an identical tempo as the simulated running (as judged by ear).

Outdoor Testing: Our first set of tests required three test subjects to wear the SoleBeat system and run at a consistent cadence in a loop whose running surface included brick, dirt, and grass. The FSR outputs were recorded over time:

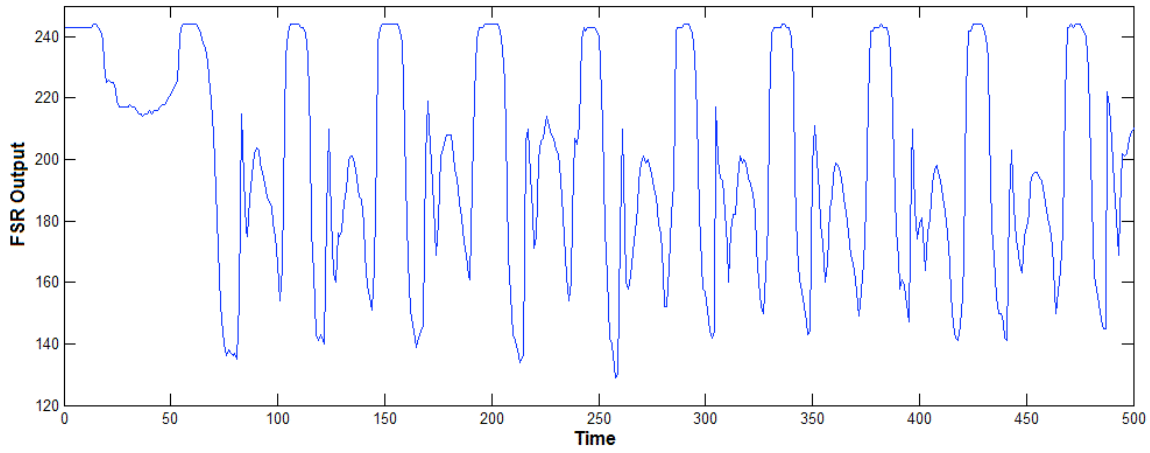

Figure 1: Shoe pressure output while running on a varied surface, test subject #1

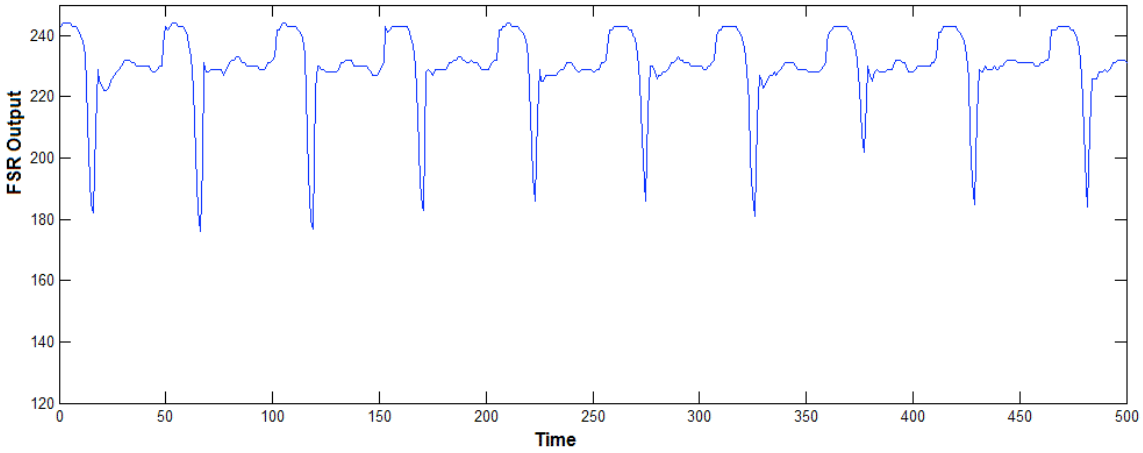

Figure 2: Shoe pressure output while running on a varied surface, test subject #2

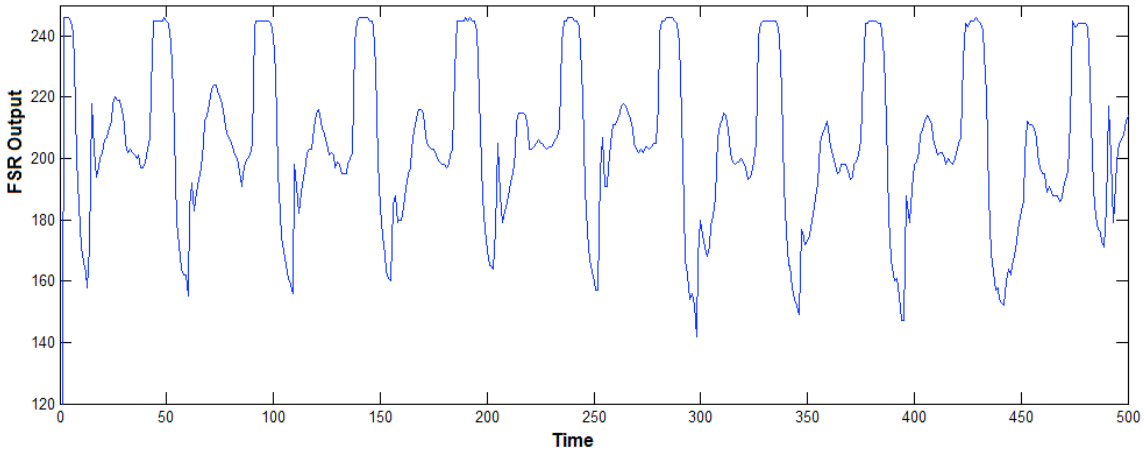

Figure 3: Shoe pressure output while running on a varied surface, test subject #3

As the data shows, each test subject shows a slightly different foot pressure pattern per footfall, however, pressure is so great while the heel is on the ground that FSR outputs always plateau at a value above 240 on each footfall, and always hit a value of 230 or lower when the heel is not on the ground, which allows a system calibrated to this data to consistently pick up each footfall. Its also worth note that, while the running loop was made up of patches of different running surfaces, each footfall output was fairly regular.

We also had each test subject use the control FSR as they would while running, with the control FSR attached to their chest, hitting it with their hand, like one of those communicators in Star Trek. The output of the control FSR was recorded over time:

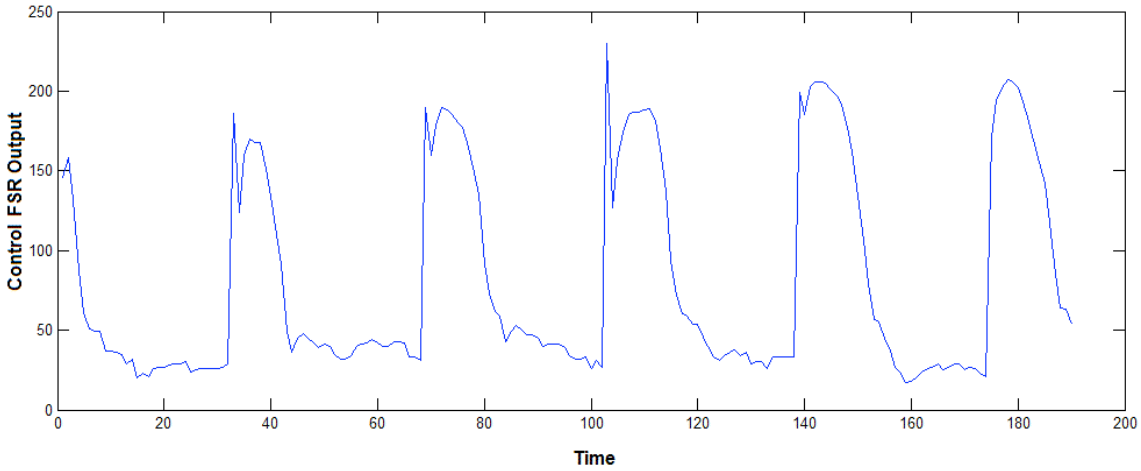

Figure 4: Using the control FSR, test subject #1

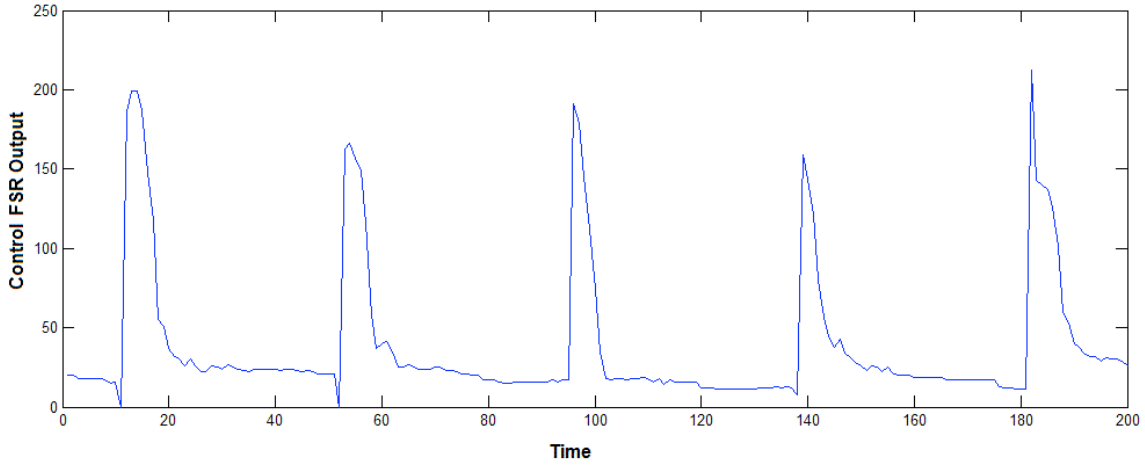

Figure 5: Using the control FSR, test subject #2

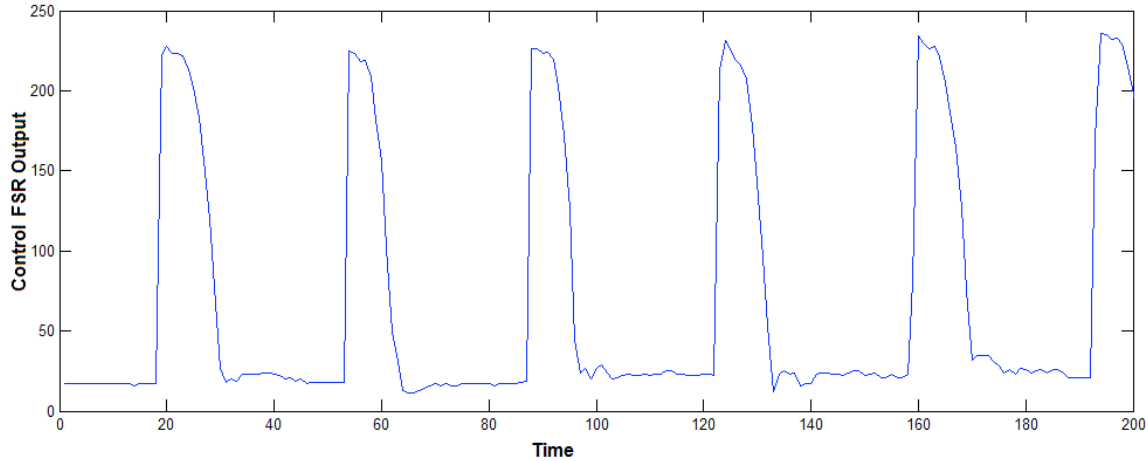

Figure 6: Using the control FSR, test subject #3

Again, each test subject had a slightly different pressure pattern when using the control FSR, but the patterns are similar enough that a high and low input threshold can be hard-coded into the system that will allow a variety of users to use the system and still maintain accurate control FSR use.

Next in our outdoor tests, we made sure that the system would operate and play/switch songs as predicted by our algorithm in each of the scenarios given in the "Testing" section. A test subject put on the SoleBeat system and ran on and in the area near the outdoor testing loop.

Tests (a)-(i) [see *Outdoor Testing* under **Testing**] were performed successfully.

Treadmill Testing: This set of testing was done the day after the outdoor testing. Starting up the system and doing a preliminary test running in place, we recorded the following output:

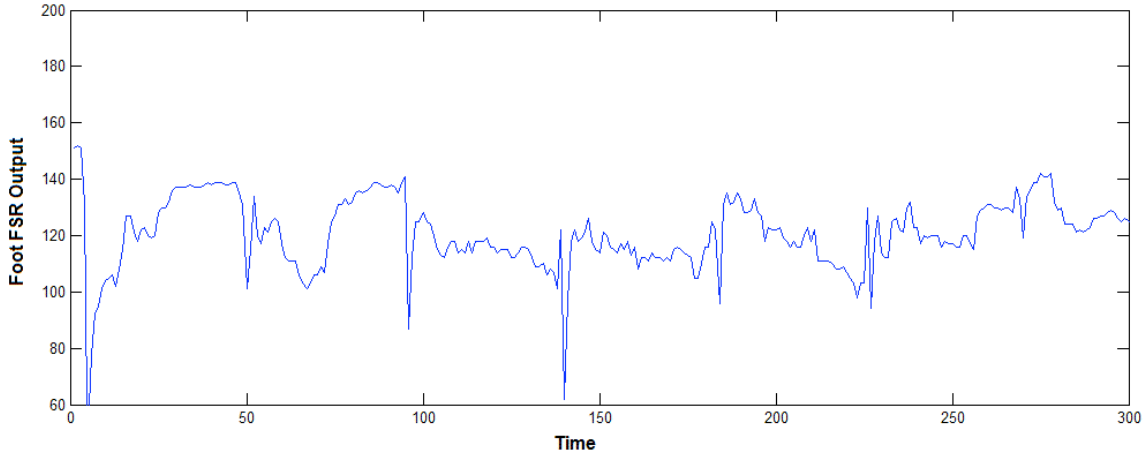

Figure 7: Shoe pressure output while running in place, test subject #3

Note the vertical scale change between this graph and figs. 1, 2, and 3. FSR output has decreased significantly, from ~150-250 to ~50-150, and the output pattern has changed somewhat. The system had had a repair on one of the wires connecting the shoe FSR to the BASIC STAMP the day before, and only marginal testing was done at that time to ensure that the FSR was still detecting pressure.

Still, we continued testing, hoping to recalibrate SoleFedge to work with a new output pattern. We recorded a number of runs on the treadmill, and tried to recalibrate the device. Figures 8-11 are a few excerpts from those runs. Note that the scale of the x-axis is not consistent between graphs.

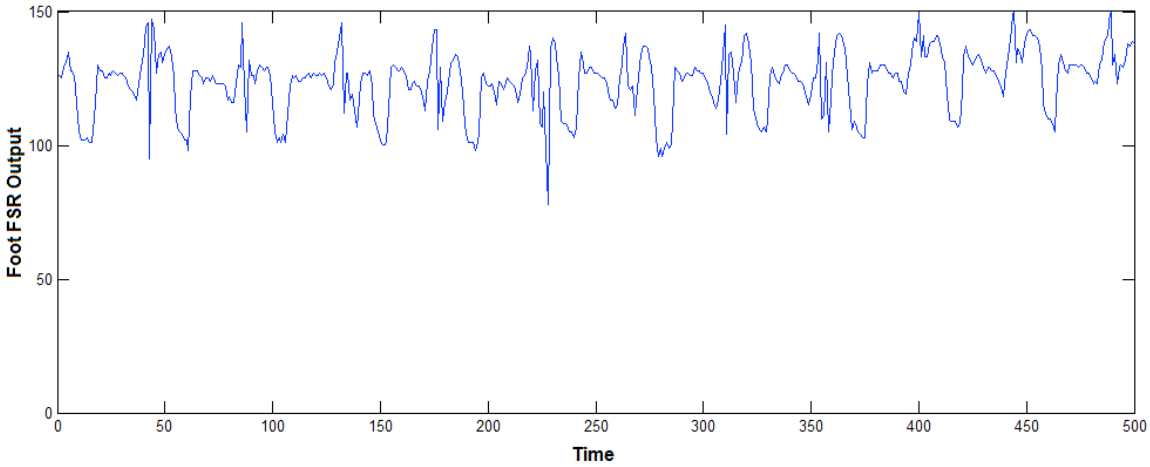

Figure 8: Shoe pressure output on a treadmill, test subject #1

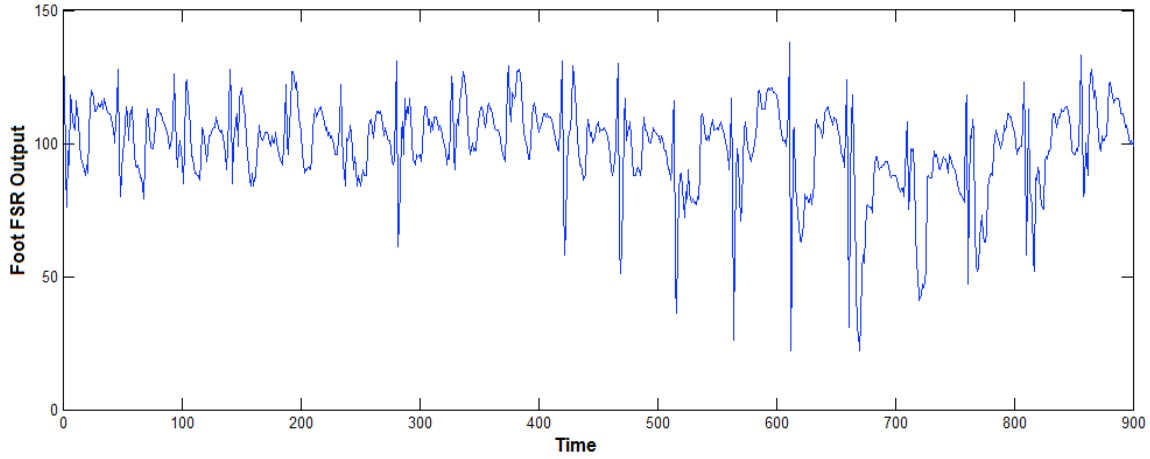

Figure 9: Shoe pressure output on a treadmill, test subject #1

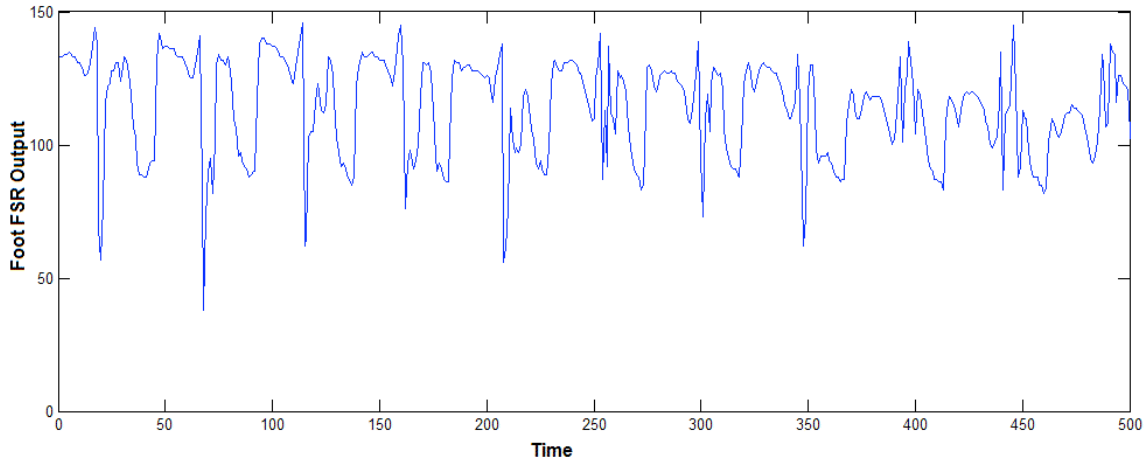

Figure 10: Shoe pressure output on a treadmill, test subject #1

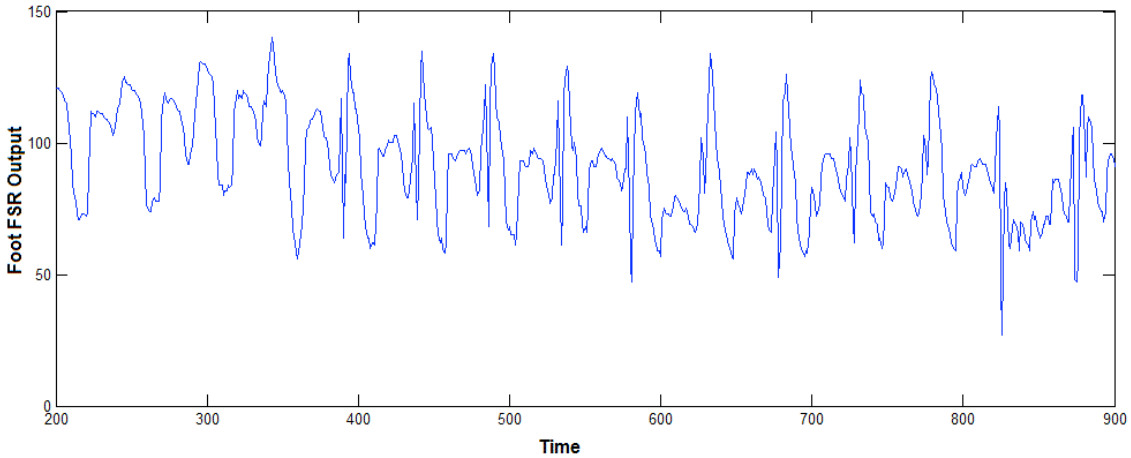

Figure 11: Shoe pressure output on a treadmill, test subject #1

Calibration efforts were only successful for short lengths of time. As is visible in the graphs, the pattern of FSR output ranges wandered slowly with time, just enough to

prevent us from encoding a consistent high value threshold and low value threshold that would allow the device to correctly register footfalls.

When test subjects exchanged the device, an even more drastic change in FSR output occurred, as visible in Figure 12.

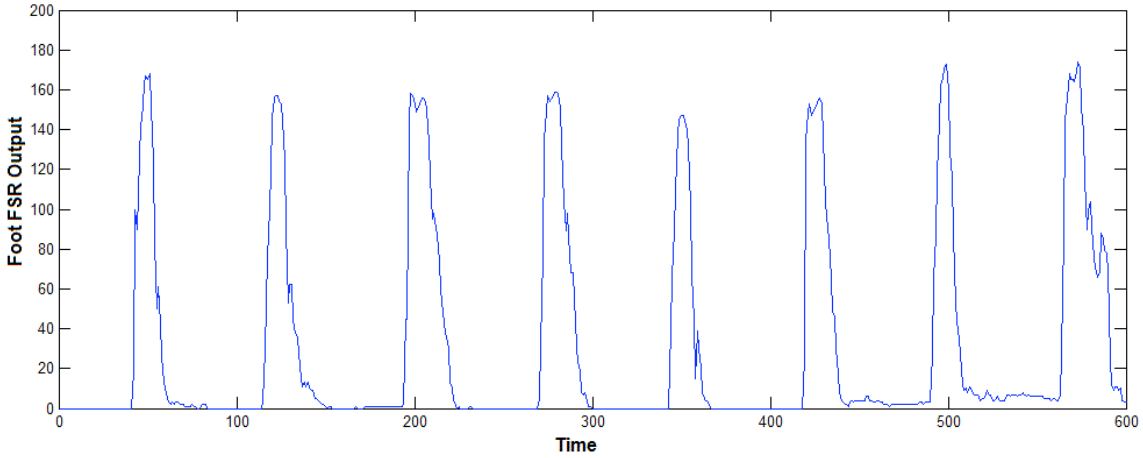

Figure 12: Shoe pressure output on a treadmill, test subject #3

Output values were very high during footfalls, and read 0 otherwise. Soon afterwards, all shoe output values dropped to zero and the system no longer showed output values for pressure on the shoe FSR. Examination of the device gave little indication of the precise location of the malfunction.

Similar Work

Apparently, Apple has caught wind of our idea, and implemented their own version: http://www.apple.com/ipod/nike/run.html. Their product gives you info on your time, distance, and pace, but does not select songs or change their tempo according to your cadence. However, Apple has already applied for a patent on music tempo modulation for runners, so we probably won't be able to market the SoleBeat.

Conclusion

We were able to create a software system which accurately translated pressure signals to control the playing of music across a broad spectrum of tempos, build a hardware system that allowed this information to be transferred between several components, and construct a physical system which allowed a runner to use the system relatively unencumbered, though very limited in range from supporting hardware. We were able to meet the listed design goals. Real-world experimenting satisfied tests for correct operation. Performance and evaluation testing was fully planned but was limited by degradation of the system. In terms of personal satisfaction, it was very cool to go from talking about a shoe that could play the perfect running music beat to actually being able to build and use the system, albeit for a short period of time.

Future Work

Despite the success of our technology, there are many improvements to be made.

First of all, the playlist is hardwired into the programming, so users lack flexibility in choosing their own songs. In the future, we would program a user interface that enables users to create a playlist (via iTunes) and that accesses a website to determine the tempo of that song (or allow user to enter in manually). An enhanced controller would also be developed to enable users to perform such operations as select their own song while running (override pace-based selection) and select from multiple songs indexed to the same or similar tempo range.

In future development stages we would also consider implementing an algorithm that maintains pitch when the song speed is changed. Currently, when tempo is increased or decreased within the tempo range of the song, the song is slowed or sped up, and there is a concomitant pitch shift.

The functionality of our shoe was significantly trammeled by the wires and the sundry hardware required to process inputs and deliver music output. Future improvements would be aimed at streamlining: integrating a chip onto the shoe that would preprocess the signal and send information wirelessly to a small computing unit (similar to an iPod). The computing unit could provide visual feedback as well, providing the user with instantly accessible information about song play, pedometry, and chronometry.

Wireless capabilities would also enhance durability, as our device suffered the most defects due to wiring issues. The next step would involve improving the durability of the sensors and the robustness of the software. A consumer-level product must be able to withstand a high degree of wear and tear and perform robustly.

Finally, future development would seek “swapability.” The current design requires modification of a shoe itself, causing damage to it. In the future, a modular design would allow the device to be integrated and removed from any shoe with little or no effect on the shoe. Users would thus be able to share the device or transplant it among shoes with ease.

References

The ChucK website: http://chuck.cs.princeton.edu/doc/

Ge Wang: He's instructed us on the use of ChucK and how to get our system's components to communicate.

Acknowledgements

We'd like to thank Professor Perry Cook for giving us insight into some of the challenges that we'd face doing this project, and giving us references to other similar projects; Ge Wang for explaining how to get all of our system components to work together; and Rebecca Fiebrink for general help (including a cool article in the NYTimes).

Appendix A: Glossary

Footfall - A unit used to describe the running action of having a foot impact the ground, push forward, and be raised again once.

Controller FSR – The FSR used to effect song changes based on current pace. When the system is initialized, the laptop plays no music. On the first controller FSR press, music begins playing at the calculated tempo. Upon subsequent controller FSR presses, running cadence is recalculated and music tempo is modulated/new songs are selected as appropriate

Running Cadence - the rate at which a person makes footfalls while running.

SoleBeat - Refers to the whole SoleBeat system which is comprised of the hardware and software (See SoleFedge) and shoe that allows us to play cadence-matched music.

SoleFedge - The software system that powers the SoleBeat, written in PBASIC and Chuck.

Appendix B: PBASIC Code

```
' {$STAMP BS2}
' {$PBASIC 2.0}

' Footfall Detector

' Reads input data from SoleBeat FSRs and generates messages that signify footfall events and control FSR use events.
' Footfall Detector uses two very basic finite-state automata to only send event signals when the FSR inputs have
' alternated below and above certian input thresholds.  FSR events are sent out as MIDI signals.

' Reuses a lot of code from Princeton COS 436 labs.

i VAR Word

CS  CON  2  ' Chip Select Pin
CLK  CON  3  ' Clock Pin
DIO_n  CON  4  ' Data In/Out Pin
config VAR Nib ' configuration bits
AD0 VAR Word 'a d channel 0
AD1 VAR Word 'a d channel 1
h_count VAR Word 'high count
l_count VAR Word 'low count

status0 VAR Word
status1 VAR Word

min_num CON 1 ' number of counts above/below a threshold value to enter the "Footfall" or "Non-Footfall" state
'Output high/low thresholds for the foot FSR
h_thresh_AD0 CON 238 ' not calibrated
```

```
l_thresh_AD0 CON 225 ' not calibrated
'Output high/low thresholds for the controller FSR
h_thresh_AD1 CON 145 ' not calibrated
l_thresh_AD1 CON 60  ' not calibrated

channel = 13
mbyte2 = 58

status0 = 0
status0 = 1

myloop:
   GOSUB convert0
   GOSUB convert1

   IF AD0 > h_thresh_AD0 THEN Is_High_Zero
Is_High_Zero_return:

   IF AD0 < l_thresh_AD0 THEN Is_Low_Zero
Is_Low_Zero_return:

   IF AD1 > h_thresh_AD1 THEN Is_High_One
Is_High_One_return:

   IF AD1 < l_thresh_AD1 THEN Is_Low_One
Is_Low_One_return:

  'Print sensor outputs for debugging
   DEBUG "AD0: "
   DEBUG DEC AD0', CR
   DEBUG "  AD1: "
   DEBUG DEC AD1, CR
GOTO myloop:

Is_High_Zero:
   IF status0 = 0 THEN Send_Signal_Zero
   Send_Signal_Zero_return:
   status0 = 1
GOTO Is_High_Zero_return

Is_Low_Zero:
   status0 = 0
GOTO Is_Low_Zero_return

Send_Signal_Zero:
   'mbyte2 = AD0/2    ' note number
   ' DEBUG "Shoe Down DOWN DOWN DOWN DOWN DOWN DOWN", CR
      mbyte2 = 1
      GOSUB midion:
      'PAUSE 200
      GOSUB midioff:
GOTO Send_Signal_Zero_return

Is_High_One:
   IF status1 = 0 THEN Send_Signal_One
   Send_Signal_One_return:
```

```
   status1 = 1
GOTO Is_High_One_return

Is_Low_One:
   status1 = 0
GOTO Is_Low_One_return

Send_Signal_One:
   'mbyte2 = AD0/2   ' note number
   ' DEBUG "********************Button Pressed************************", CR
      mbyte2 = 2
      GOSUB midion:
      'PAUSE 200
      GOSUB midioff:
GOTO Send_Signal_One_return


convert0:    ' Convert channel 0 only
 config = %1011
 LOW CS
 SHIFTOUT DIO_n,CLK,LSBFIRST,[config\4]
 SHIFTIN DIO_n,CLK,MSBPOST,[AD0\8]
 HIGH CS
RETURN

convert1:    ' Convert channel 1 only
 config = %1111
 LOW CS
 SHIFTOUT DIO_n,CLK,LSBFIRST,[config\4]
 SHIFTIN DIO_n,CLK,MSBPOST,[AD1\8]
 HIGH CS
RETURN

'*****************************
'*****   USEFUL MIDI FUNCTIONS
' Set channel to 0 - 15
' NOTE: mbyte2 and mbyte3 must be in range 0-127!!

channel  VAR Nib
mbyte2   VAR Byte
mbyte3   VAR Byte
lastnote VAR Byte

noteOn   CON    144
channel0 CON    0
noteOff  CON    128
controlr CON    176
atouch   CON    208
bender   CON    224
prgchang CON    192
MIDIPIN  CON   7
MIDITIME CON   12   '****  12 for BS2, 60 for BS2SX *'

'*** Standard Note On *'
midion:
 SEROUT MIDIPIN, MIDITIME, 0, [noteOn + channel, mbyte2, mbyte3]
```

```
'     debug "NoteOn ", dec mbyte2, "  ", dec mbyte3, cr
RETURN

'*** Standard Note Off, uses real NoteOff status byte *'
midioff:
 SEROUT MIDIPIN, MIDITIME, 0, [noteOff + channel, mbyte2, 64]
'     debug "NoteOff ", dec mbyte2, cr
RETURN
```

Appendix C: ChucK Code

```
/***************** IMPORTANT GLOBAL VARIABLES *****************/

false => int debug; //test code?

//establish signal flow; envelope employed to smooth song transitions
SndBuf buf => Envelope e => dac;
0.5::second => dur fade => e.duration; //duration of fade between songs

// array of sound files (songs)
["./Dammit.wav", "./Dopeman.wav", "./GoodLovin.wav", "./MysticBuzz.wav",
"./WelcometoParadise.wav", "./Christmas.wav", "./MeaningofLife.wav", "./TheOne.wav",
"./BleedAmerican.wav", "./MeltwithYou.wav", "./GoodDieYoung.wav", "./HereitGoesAgain.wav",
"./RockandRoll.wav", "./JerkItOut.wav", "./ReadytoGo.wav","./TillHearIt.wav","./Discotheque.wav",
"./AntsMarching.wav", "./Numb.wav", "./mmmBop.wav", "./BringEmOut.wav", "./MyLifeStory.wav",
"./InDaClub.wav", "./ComeTogether.wav", "./Amsterdam.wav", "./LoveMeDo.wav",
"./DareYoutoMove.wav"] @=> string songs[];

songs.cap() => int numsongs; //number of songs

//duration of two beats in corresponding song
//useful for comparison because we want the duration of a stride (2 steps) to be 2 beats
[0.56603774::second, 0.60000000::second, 0.61538462::second, 0.64864865::second,
0.66666666::second, 0.68571429::second, 0.70588235::second, 0.72727273::second, 0.75000000::second,
0.77419355::second, 0.80000000::second, 0.82758621::second, 0.85714286::second, 0.88888889::second,
0.92307692::second, 0.96000000::second, 1.00000000::second, 1.04347826::second, 1.09090909::second,
1.14285714::second, 1.20000000::second, 1.26315790::second, 1.33333333::second, 1.41176471::second,
1.50000000::second, 1.60000000::second, 1.71428571::second]
@=> dur twobeat_duration[];

if(debug){
   //ensure arrays match in size
   if(numsongs != twobeat_duration.cap())
      <<<"Warning: array sizes incompatible">>>;

   //ensure tempos are in order (greatest --> least BPM)
   for(1 => int i; i < twobeat_duration.cap(); i++)
   {
      if(twobeat_duration[i-1] > twobeat_duration[i])
      {
         <<<"Warning: Tempos not in order">>>;
         break;
      }
   }
}
```

```
now => time prevtime; //time at last step
now => time currtime; //time at most recent step

0::second => dur stride_duration; //duration of stride (2 steps)


/**********************    MIDI SETUP    *************************/

// midi event variables
MidiIn min;
MidiMsg msg;
// open midi receiver, exit on fail
if (!debug && !min.open(0))
{
   <<<"No midi device detected. Program terminated.">>>;
   me.exit();
}


/**********************   FUNCTION(S)    *************************/

0 => int total_strides;
0::second => dur total_duration;
-1 => int index;      //index of song to be played/currently being played
                      //-1 when no song is currently playing
twobeat_duration[numsongs-1] + twobeat_duration[numsongs-1]*0.1 => dur longest_duration; //slowest
pace
twobeat_duration[0] - twobeat_duration[0]*0.1 => dur shortest_duration; //fastest pace

fun void change()
//play new song or modulate the tempo of current song, depending on pace
{
   <<<"CHANGE">>>;
   0 => int i;
   shortest_duration => dur prevtempo;
   dur nexttempo;
   dur thistempo;
   dur lowerthreshold;
   dur upperthreshold;

   while(i < numsongs) {
      //if tempo is in the interval
      if(i != numsongs-1) twobeat_duration[i+1] => nexttempo;
      else longest_duration => nexttempo;

      twobeat_duration[i] => thistempo;

      thistempo - (thistempo-prevtempo)/2 => lowerthreshold;
      thistempo + (nexttempo-thistempo)/2 => upperthreshold;

      if( (lowerthreshold < stride_duration) && (stride_duration <= upperthreshold) )
      {
         //if the song was playing, don't do anything, otherwise, play
         if(i != index)
         {
```

```
            i => index;
            //fade out to avoid stutter
            e.keyOff();
            fade + 100::ms => now;
            //restore volume and switch songs
            1 => e.value;
            songs[index] => buf.read;
          }
            //match tempo of song with pace
          twobeat_duration[index] / stride_duration => buf.rate;

          break;
        }

        thistempo => prevtempo;
        i++;
    }
}

/********************   MAIN EXECUTION LOOPS   **********************/

if (debug)
//play each song for 5 seconds each
{
   numsongs => int x;
   while(x > 0) {
     twobeat_duration[x-1] => stride_duration;
     change();
     3::second => now;
     x--;
   }
   <<<"DONE TESTING!!!!">>>;
}
else {
//THE REAL LOOP
while( true )
{
   // wait on midi event (stomp)
   min => now;

   // receive midi message(s)
   while( min.recv( msg ) )
   {
     // catch only noteon
     if( msg.data1 != 157 ) continue;
     if(msg.data2 == 1)
     // caculate duration of stride
     {
        now => currtime;
        currtime - prevtime => stride_duration;
        currtime => prevtime;
       total_strides++;
       stride_duration+=> total_duration;
       <<<"">>>;
       <<<"Elapsed time =", total_duration/second, "seconds">>>;
       <<<"Total number of strides =", total_strides>>>;
```

```
    }

    else if(msg.data2 == 2)
    // change song or modulate tempo of song when signaled,
    // if stride tempo is within appropriate parameters
    {
       if(stride_duration > shortest_duration && stride_duration < longest_duration)
                change();
    }
  }

  200::ms => now;
}
}
```

Appendix C

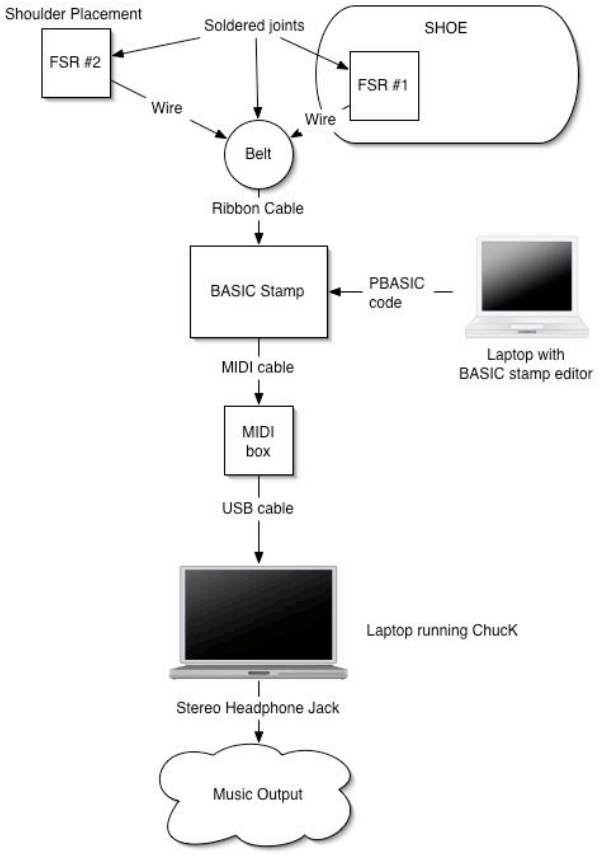

Fig. 13 Block Diagram

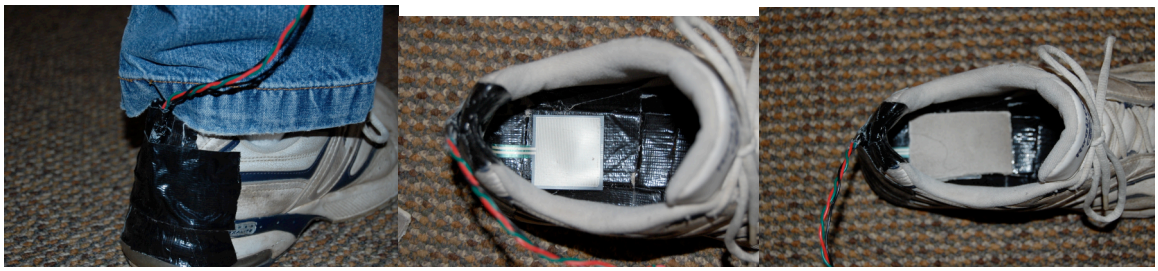

Fig. 14 Shoe (from left to right: heel, step-sensing FSR, cover of FSR)

Fig. 15 Wiring

Fig. 16 Controller FSR