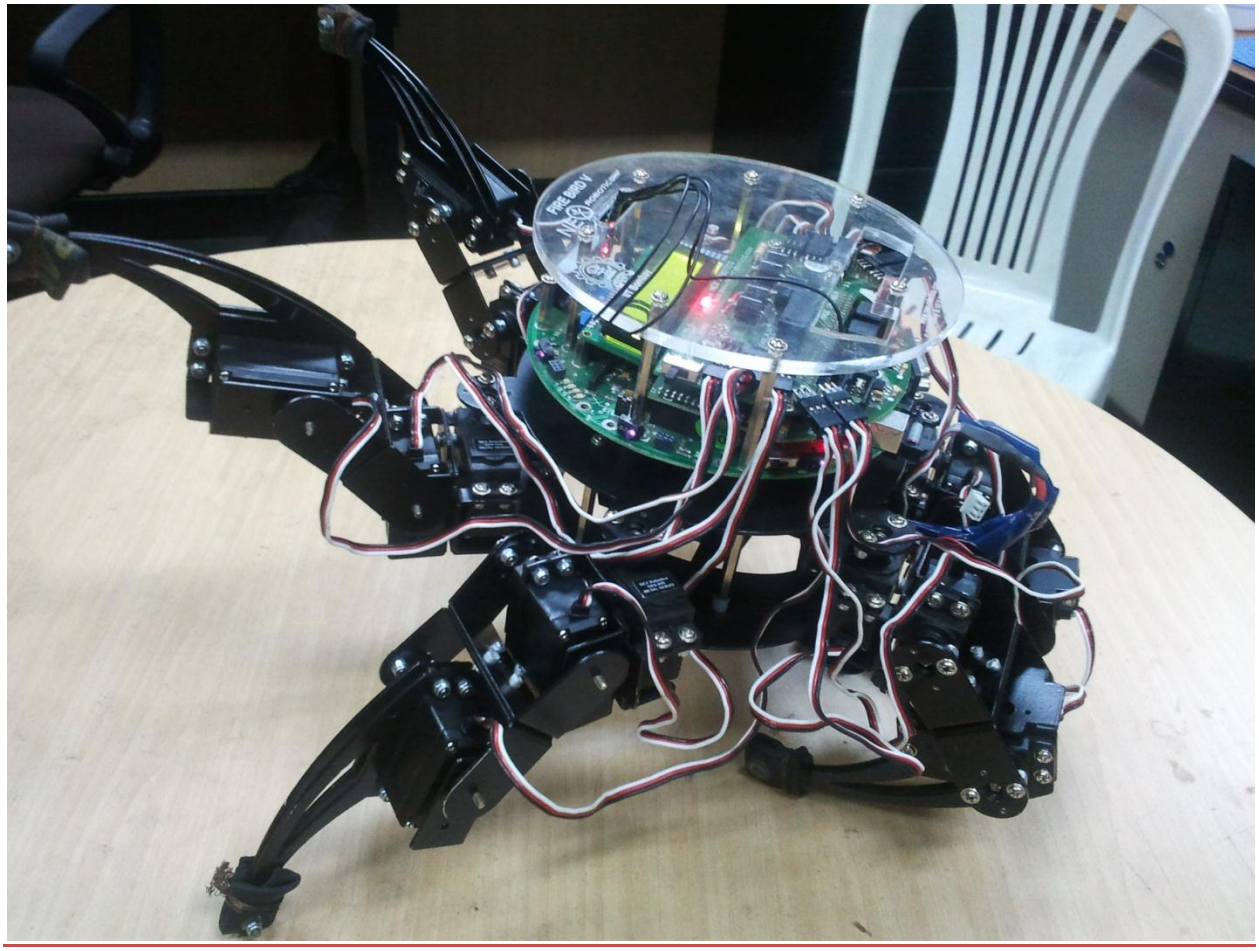
Dancing Hexapod Report

CS 684 Embedded Systems Course Project



GROUP 12

Manas Chaudhari [09D26003]
Ayesha Mudassir [09007014]
Kedar Tatwawadi [09D07022]
Siddharth Sarangdhar [09026007]

Contents

1 Introduction

A hexapod robot is a mechanical vehicle that walks on six legs. Since a robot can be statically stable on three or more legs, a hexapod robot has a great deal of flexibility in how it can move.

Most often, hexapods are controlled by gaits, which allow the robot to move forward, turn, and perhaps side-step. Some of the most common gaits are as follows:

- Alternating tripod: 3 legs on the ground at a time.
- Quadruped.
- Crawl: move just one leg at a time.

Gaits for hexapods are often stable, even in slightly rocky and uneven terrain.

Usually, hexapods are controlled using remote controls. In some hexapod dance competitions, a variety of hexapod dance moves are programmed on the hexapod. However, these dance steps are hardcoded and are not according to the beats of the song. In this project, we explore the interesting idea of making the hexapod dance to any song.

2 Definitions

1. **Gait (dance moves):** Wikipedia gives definition of gaits as:

> *… the pattern of movement of the limbs of animals, including humans, during locomotion over a solid substrate.*

Hexapod robots are biologically inspired by Hexapod locomotion .In the project by "gait" we mean a dance move of the hexapod [defined by a set of sequence of motor actions]

2. **Phase and frequency of beats**: An important part of our project is identifying the beats in any given song. Typically in an upbeat song (dance songs) a wide variety of percussion instruments are used and they constitute the rhythmic element of the song.

These beats can typically be modeled as a superposition of periodic functions with some period and phase (time signature).

[Using sound processing our software will communicate the frequency of these beats and their time signature (phase) so that the bot can match its movements to the beat.]

3. **Metadata:** This is the data sent by the android phone to the hexapod wirelessly. It would consist of all the information necessary for the hexapod to plan and execute the dance moves. This information would be structured in such a way that the hexapod will have to perform minimal processing in order to dance.

4. **Dance Step:** Typically, songs have a beats frequency of 1 – 5 Hz. There is a constraint on speed of performing an action for the hexapod. Hence, any gait (dance move) is broken down into a sequence of basic dance steps. Each dance step will be executed at a beat.

5. **Bot:** Hexapod

3 Problem Statement

In our project, we designed an autonomous dancing hexapod, a Hexapod which will dance to any song played. The user has to play any song on his Android based phone, and the hexapod will dance according to the song played. The Android phone will communicate wirelessly with the hexapod using bluetooth.

4 Requirements

4.1 Functional Requirements:

1. Liveness: The hexapod should perform dance -moves after reading the metadata sent to it.
2. Safety: The hexapod should not lose balance or cause damage to itself due to excess stress/strain
3. Timeliness: Delay between the dance step and the corresponding beat must be low so that it is not apparent visually.
4. It should have at least 5 aesthetic moves to choose from.
5. A UI through which the user will interact with the hexapod. The UI will allow all types of functions such as play/pause/change song.

4.2 Non-functional Requirements:

1. The android phone must communicate wirelessly with the hexapod.
2. Hexapod movement should be fast and smooth.
3. Hexapod must not leave the arena.
4. The user must be alerted in case of failure.

4.3 Hardware Requirements:

- Hexapod Firebird research platform
- Bluetooth Module with UART support
- Android device

5 Implementation:

Our project has 4 important parts:

1. Sound processing- Designing the beat detection and beat identification algorithm.
2. Designing and Implementing different gaits(dance moves) for the hexapod.
3. Developing an android app for implementation of the sound processing algorithms developed.
4. Setting up wireless communication between the android phone and the hexapod.

5.1 Beat Detection:

As stated earlier, our project aim was making the hexapod dance based on the song and in sync with the beats of the song. For achieving this sync, its necessary to detect the beats, tempo/period of the song. We now describe the beat detection algorithm which we designed and implemented in JAVA (Android 2.3.6 compatible).

The algorithm specifications are given below:

- Most of the standard beat Detection algorithms employ quite sophisticated techniques like high-pass filtering followed by spectral change etc.(eg: Have a look at the classic paper by Simon Dixon http://www.eecs.qmul.ac.uk/~simond/pub/2001/icmc.pdf) . But, most of algorithms aren't pseudo-real time (almost real time). Also, the algorithm was targeted for a simple Android compatible smartphone. So, its difficult to directly implement the existing algorithms, and that we had to come up with our own simplified algorithms which will work in pseudo-real time (but for a smaller subset of songs)

- We considered one of the most simple properties of beats: they represent local energy maximas in the spectrum. Not that this isn't a very good representation and will work with songs having strong percussion instruments (for example drum beat tracks).

- We split the algorithm in 2 parts, namely : beat detection algorithm and period detection algorithm.

- The beat detection algorithm performs the following :

    1. Finding average chunk energies

        We take a chunk of samples (eg: chunks corresponding to 5ms duration ,i.e 200 samples for a sampling rate of 44100 Hz)and take their average. We store these averages as

average chunk energies. This Averaging helps in reducing random noise due to smoothening. Also the other benefit of averaging is reducing the number of computations by the significant factor (by a factor of 200 almost!)

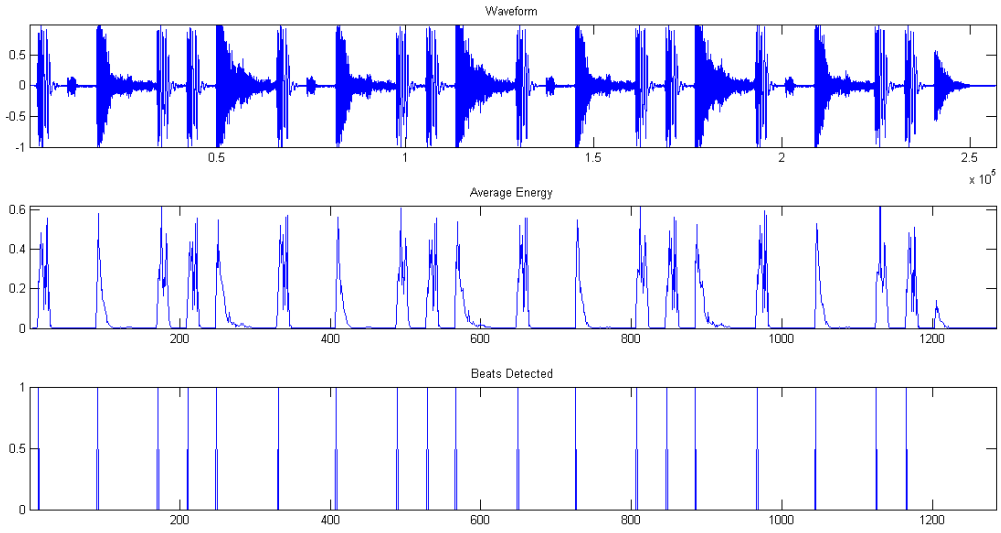
2. Thresholding the chunk energies

   Once we have found the average chunk energies, we perform thresholding . The thresholding is dependent upon the maximum peak chunk average energy in the signal.This removes spurious indivisual peaks which weren't removed by averaging.

3. Performing thinning to obtain beat locations

   Now, even after performing averaging and thresholding, still it is observed that a single beat corresponds to multiple chunks. To solve this problem, we perform thinning based on the maximum number of beats possible per second (or equivalently, the minimum inter-beat spacing). This finally gives us the beat locations.

- The figure below describes the beat detection algorithm simulation in matlab.

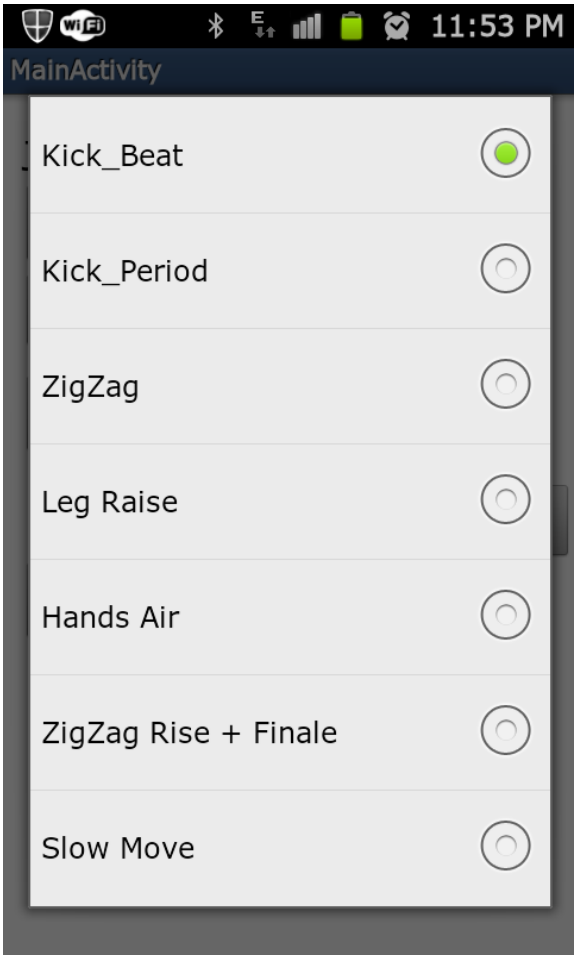


- The next part is finding the period/tempo. The standard approach is to use autocorrelation of the time domain signal, and finding the peaks , gives the period. But, for a 20 s song with a sampling rate of 44100 Hz, this would imply 10,00,000 samples to be processed for finding each autocorrelation coefficient, which would be extremely slow and redundant for our application.

- Hence, we came up with a modified period detecting algorithm, in which we consider the autocorrelation of the beat locations, rather than the entire song. This tremendously reduces the complexity from 10^{12} computations (10,00,000 samples for ~ 2,00,000 autocorrelation coefficients) to ~ 10^{3} computations (at max 40 beats locations for ~20 modified correlations).
- While considering the correlation, we insert lower thresholds instead of exact period matching, as the beat locations detected might not be perfect.
- Please refer to the doxygen documentation of the JAVA implementation, and the matlab simulation files for more details.

5.2 Dance Moves:

Each dance move of the hexapod is split into multiple steps. A step consists of a single motion which can involve multiple motors also. An example of dance move split into steps is discussed in Section 5.4.2 below.



5.3 Communication Protocol:

All the dance moves are given unique single character ids from ‘a’ to ‘g’. The hexapod listens for the following commands:

a -> g	Select dance move
Y	Execute next step in the dance step
Z	Cancel dance move and return to standing position

In the start, the move id of the selected dance move is sent to the hexapod. On receiving the move id, the Hexapod remembers the move and waits for the command to proceed to next step i.e. ‘y’. For ‘Smooth’ type of dance moves (explained in Section 5.4.1.3 below), an extra character is sent following the move id. This argument contains the speed at which the dance move has to be performed.

Thus, whenever the audio playback head reaches the beat location, the command ‘y’ is sent to the hexapod.

5.4 Scheduling:

This was one of the challenging tasks of this project. The basic goal was to come up with a strategy to select beats from the beat pattern of the song for performing the steps of the dance move. We have come up with three types of selection procedures.

Different procedures are suited for different dance moves depending on the speed of execution of dance steps and style. Speed usually depends on the load and the maximum change. For example, lifting a single leg is fast and also reversible. Reversible means the action can be reversed even if it is not complete without causing any damage to the hexapod.

5.4.1 Beat-Step Selection Types:

5.4.1.1 By Beat:

Steps are performed on every beat. This type is suitable for dance moves in which steps are very simple and fast. Every beat is selected for the step as long as the difference from the previous selected beat is greater than the minimum step time requirement for the current dance move.

5.4.1.2 By Period:

The beat detection module also calculates the period of the beats. Period is usually around 1-2 seconds while minimum step time requirements for dance moves are around 200-500ms. Depending on the ratio of period and minimum step time requirement of the dance move, maximum number of steps that can be performed in a period is calculated. The number of steps in a period is chosen to be the greatest power of 2 which is less than the maximum number calculated. Power of 2 is chosen because most of the songs are 8-beat in nature. Hence when the period is divided into 2, 4, 8, 16, they match the tempo of the song.

5.4.1.3 Smooth:

Some dance moves need not be performed on beats. They are simply combinations smooth steps. However, unlike the above two types, the speed of performing these steps can be different which needs to be matched to the speed of the song. For these types of dance moves, although the motion is split into steps, the steps are continuously executed one after the other without a break. Hence, for these steps, instead of minimum step time requirement, we have used the minimum total move duration as a characteristic. The dance move duration is usually greater than the period. Hence, the speed of the dance move is chosen such that the move spans multiple periods. The ratio of minimum dance move duration and period duration gives the minimum number of periods per dance move. Number of periods per move is chosen by rounding this ratio to an upper value. The value corresponding to this is sent to the bot just after the move ID.

To maintain uniform communication protocol, the number of steps in these moves is assigned to be 1. Thus, the next step signal 'y' is equivalent to starting the dance move.

5.4.2 Time Calculations:

From the specifications of NRS-993 servo motor used in the hexapod we found that the motor takes 150 ms to turn by 60˚.

Based on this speed, we calculated the minimum step time requirement for every dance move. Minimum step time requirement is the time required between consecutive steps so that the motions do not overlap. As there is no feedback mechanism for getting the angles of the hexapod [See 7.1.2] these time calculations are very important.

Example Calculations for dance move 'Hands Air':

This move is comprised of 8 steps. In first step, legs 1 and 6 are lifted up while the hexapod bends a little through legs 2,3,4,5. The second step is exactly the inverse of the first. Next 4 steps are similar for different pairs of legs that are lifted. In the last two steps, the hexapod bends and returns respectively through all legs.

Angles at standing position: (90, 60, 20) for motors (A, B, C).

Angles at the end of step 1

legs 1,6 : (90, 0, 100)

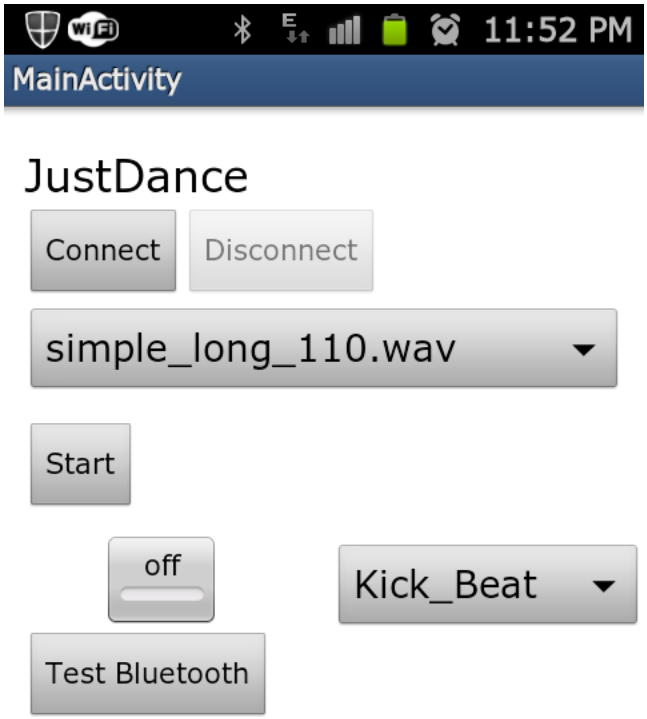
legs 2,3,4,5 : (90, 40, 0)

Thus, maximum change is for motors 1C and 6C from 20 to 100 degrees. Thus, maximum angle change for first six steps is 80 degrees. For the last two steps, all legs go to position (90, 40, 0) and back. Thus, maximum angle change is 20 degrees.

Thus, the overall maximum angle change for this dance move is 80 degrees. Therefore, at the speed of 150ms/60 degrees, it would take a maximum of 200ms to complete one step.

As there is usually a gap between any two dance steps almost equal to the duration of steps (generally, dance is performed on alternate beats), we take the minimum step time requirement for this dance move as twice the calculated value i.e. 400ms in this example. This would also give a safety factor of 2 considering that the motion can get slowed under load.

5.5 Android UI:



The user interface consists of a Connect and Disconnect button through which the user can connect to or disconnect from the hexapod. The MAC address of the Bluetooth module on hexapod is used to identify the device.

Through the two drop down lists (‘simple_long_110.wav’, ‘Kick_Beat’), the user can select the sound track and the dance move respectively.

The Start button does the following tasks:

- Extracts PCM data from the selected track using musicg library [See References for link]
- Detects beats using the extracted PCM data
- Schedules the dance steps of the selected move on the detected beats to obtain the locations on which the dance steps are to be performed
- Send the dance move Id (and argument if any) to the hexapod
- Generate an AudioTrack object containing the PCM data
- Set the marker of the audio track to the next beat location and subscribe to the MarkerReached event.
- Play the audio

5.6 Synchronization:

The AudioTrack class (in Android SDK) manages and plays a single audio resource. It allows streaming PCM audio buffers to the audio hardware for playback. This is achieved by "pushing" the data to the AudioTrack object using one of the write(byte[], int, int) and write(short[], int, int) methods

It also supports setting a marker (in terms of frame/sample of audio data) so that notifications are received when this marker is reached. When the marker is reached, two tasks are performed:

- Signal the hexapod to perform step on the right time
- Set the marker for next step

6 Testing Strategy

Our project has three parts which need to be evaluated. They can be evaluated independently.

1. *Detection of beats in music:*
   We organized the dance tracks in increasing order of difficulty as follows:

   1. Audio with single periodic beats
   2. Audio having single beat but with variable periods
   3. Audio with multiple beats with same periods
   4. Audio with multiple beats with different periods
   5. Audio with lyrics or other instruments having rhythmic elements which are not beats.

   Our hexapod can dance to the tracks of first four levels. Due to hardware limitations explained later [Section 7.1.2 below] we could not implement the 5th level on Android.

2. *Performing hexapod moves:*

   We have designed 6 dance moves for the bot. We have categorized dance moves into three types:

   1. By Beat: Step is executed on every beat
   2. By Tempo: Steps are evenly distributed over a period
   3. Smooth: Single step dance move distributed over multiple period. Can be executed at variable speeds by sending an argument to hexapod.

3. *Synchronization of moves:*

   A dance move is successful if the system is able to execute its dance steps in a fashion in which there is no visible delay. In our implementation, the only delay between the beat and the dance step is due to the Bluetooth communication latency. This delay is not visible.

7 System Description

7.1.1 Dance moves:

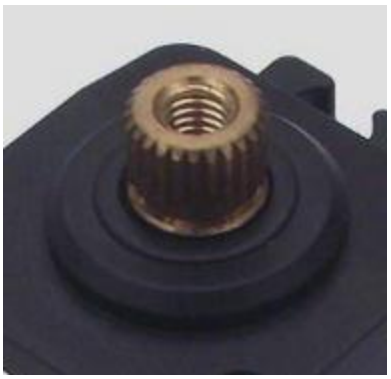
When deciding the angles required for a dance move, it was very important to ensure that the resultant motion of the hexapod's leg is in the right direction. Changing a single motor's angle (per leg) would result in a rotational motion. To obtain linear motion, change of multiple motors is required. For example, very frequently, it is required to lift the leg from the floor. If only motor B is used, it results in a jerk causing instability. Thus, along with B, motor C also has to be changed so that the result is a vertical motion.

7.1.2 Hexapod Limitations:

- The design of the hexapod was bulky which makes it not suitable for dance moves. Making the hexapod stand and rise is difficult due to its own weight and design.

- Tips of the legs are rubber padded which results in jerks. The rubber pads provide a lot of unnecessary friction. Special care had to be taken that the edge of the rubber pad was horizontal at all times. If this is not taken care of properly, the bot might topple due to the impulses generated by the friction.

- Non-symmetric servo motor placement:



  - The clamp is connected to the geared shaft of the servo motor in the above figure. To ensure that all the six legs are symmetric, we tried to adjust the placing of clamp. However, as the placing can be adjusted in a discrete manner only, it was not possible to resolve this matter easily. We tried to solve this issue to some extent by using software corrections. For example, we stored the error values for each servo motor so that setting servo to (the desired value + error) would give the actual required position.

- No Feedback:

  - As servo motors have been used, we can only set the angles to a particular value. There is no provision to get the current angle. Thus, there is no way to check if the dance move is complete. Also, if this mechanism existed, it would have been possible to dynamically choose next move whose starting position matches the current position.

  - As a result, the only safety measures that could be taken were with respect to time. Time delay between consecutive steps is ensured to be greater than a pre-determined threshold assuming that the move would be performed in that threshold.

  - Due to lack of this feature, it is not possible to implement runtime safety measures.

  - As it is not possible to get the current position, there was no definite way to get the angles for different positions while implementing a dance move. For example, say we wanted to move the leg to a particular position which is set on the hexapod while the motors are off. It was not possible to get the angles of the three motors at that position. Hence, we had to use the trial and error approach to obtain the hexapod's position.

7.2 What we could not achieve:

7.2.1 Beat detection on songs with lyrics:

To filter out the lyrics and other non-rhythmic elements, the audio has to be passed through a high pass filter of at least order 50.

Implementation of 50 ordered High Pass Filter for beat detection on vocal songs is too intensive for Android device. Although we did perform this on a PC and it could detect beats on normal songs as well.

7.2.2 Longer Track Duration:

The static audio playback buffer size for the Android device is limited. As per our observations, it supports static audio playback up to 30 seconds. For tracks greater than 30 seconds, an exception gets raised.

This limitation can certainly be overcome by implementing a streaming type audio playback. This could be one of the best extensions for our project as we have discussed in 8.1.1.

7.2.3 Energy Calculations:

As the hexapod has 18 servo motors, it has a very high current requirement around 15A. Also, due to this high current requirement, the hexapod only runs on the battery. It could not be run using an external power supply. Hence, current values cannot be measured.

Hence, experimental energy profiling was difficult. We performed an approximate analysis:

- Battery rating of the battery: 5000 mAh, 7.4V
- Time required for discharging: 30 mins
- Thus, Average current when servo motors switched on <= 10A ()
- Average power requirement <= 74 W
- <= because of full discharge assumption.

7.3 Problems Faced:

- Different hexapod than what was used in the past project.
  - The hexapod which we received was different than the hexapod used in past projects. Hence we could not use the code from the past projects.
  - Also, the GUI for calibrating the servo motors could not be used.
  - As a result, we had to write most of the code from scratch which resulted in an unexpected delay.
- Bluetooth communication:
  - Firebird V has spare UART 2 and UART 3 ports which can be used with external UART based communication devices. However, in the Hexapod Firebird, these ports have been allotted to servo motor controls.

  - As a workaround, we used the Zigbee communication port for Bluetooth communication by replacing the Serial In, Data out with the RXD, TXD of the Bluetooth module. [See Appendix 1: Using the Bluetooth Module]

- Late reception of hexapod
  - We received the hexapod battery after the first demo due to which we got lesser time than expected. Hence, we were not able to implement certain features such as Streamed Processing. [See 8.1.1 below]

8 Future Work

8.1.1 Streamed Processing

To implement streamed processing instead of static processing. This would involve performing all the tasks on chunks of song instead of the whole song. While the current chunk is played, the next chunk could be processed. With this extension, length of the song would not remain a constraint. Hence, longer tracks can be used. This would make many other extensions useful.

Note that if the chunk size is small enough, it would be possible to make the bot dance to the music as the sound is recorded. Processing complex tracks is quite computationally expensive. Hence, it would be difficult to achieve this. But, if the complexity of supported tracks is reduced by considering only simple tracks, it would be a very great extension to implement a real time version of this project i.e. a Hexapod which dances to the beat by recording the audio through a microphone.

8.1.2 Dance move planning

Currently, dance move to be performed is chosen by the user. This could be done automatically, based on the tempo of the song and other characteristics. In our case, the track duration was limited. Hence, implementing autonomous dance move selection did not seem very beneficial. But, if streamed processing was implemented [8.1.1 above], track duration would not be limited and hence, this extension would be very cool.

8.1.3 DSP implementation

The Android device was essential for computation as the microcontroller on hexapod was not sufficient for implementing beat detection. A DSP could be mounted on the hexapod to do this computation. Thus, Android would be required only to provide an interface for choosing song. If real time extension is implemented as suggested in 8.1.1 above, then by mounting a microphone along with a DSP, the need for android could be eliminated. Thus, the hexapod would become completely autonomous.

9 Conclusions

Overall the project was a nice experience for us, as along with Real Time Systems, we got to work with extremely varied and interdisciplinary areas including sound processing, beat processing, android app development, communication protocols etc. Also, at the end of the project, we were able to create a working model of the Dancing Hexapod.

10 References

- Android-Bluetooth Module Communication: http://bellcode.wordpress.com/2012/01/02/android-and-arduino-bluetooth-communication/
- Hexapod Dance Video: http://www.youtube.com/watch?v=msaWXY3OuQQ
- Musicg library: http://code.google.com/p/musicg/
- Beat Detection: http://www.eecs.qmul.ac.uk/~simond/pub/2001/icmc.pdf

11 Appendix

11.1 Using the Bluetooth Module

In the basic Firebird V, one of UART 2 or UART 3 can be used for the Bluetooth module. But in case of Firebird Hexapod, both these ports have been allotted to servo motors. Hence, we removed the Zigbee module and used the RXD, TXD for UART0.

11.2 Using our code

11.2.1 Requirements:

- Recommended Operating System: Windows 7
    - Unsigned driver installation is not permitted in Windows 8. We could not find any workaround
- ICCAVR 7 (https://www.imagecraft.com)
- Eclipse
- Android SDK with Android 2.3.3 platform tools

11.2.2 Android Application:

1. Launch eclipse
2. Select Import from File menu.
3. Select Existing projects into workspace in General category
4. Click on browse button for selecting the root directory. Choose the directory of this readme.
5. Check both JustDance and SoundProcessing. Also check copy to workspace.
6. Press Import.
7. To run application on your phone, follow the instructions specified here: http://developer.android.com/training/basics/firstapp/running-app.html

11.2.3 Hexapod Code:

1. Open the Hexapod Code Project in ICCAVR
   (Project --> Open --> DancingHexapod.prj)

2. Rebuild all the files (Project --> Rebuild all)
   This creates a Hex file "DANCINGHEXAPOD.hex"

3. To burn the .hex file on the Hexapod, open the command prompt and go to the Hexapod Code folder.
   Burn the code using the following command:

   ```
   avrdude -c usbasp -p m2560 -P usb -U flash:w:DANCINGHEXAPOD.hex
   ```

4. Once the Hexapod is in its standing position, you can keep it on any plain surface.

5. Connect the Hexapod's Bluetooth to your Android Phone's Bluetooth using the Connect button in our Android app [See 5.5 above].

6. Select a song and a dance step in the android app and select Play to enjoy the Hexapod dance.

7. To switch off the Hexapod, first switch off the servo motors and then the main power supply switch.

Always ensure that the servo motor switch is off when you switch on the hexapod. Servo switch should be turned on only after the initialization() and stand() command has been executed.

We have used a buzzer to indicate that it is safe to turn on the switch.