Chapter 1 - Introduction

There is a growing number of researches on humanoid robots. Many groups have developed highly articulated humanoid robots for research [*ref to ASIMO, Hubo, Geminoid, WE-4RII, Kismet, Cog*], and many companies developed small humanoid robots for hobbyists like KHR-1, Bioloid, iSobot, RoboSapien, and Robonova. The research topics on humanoid robots ranges from the bipedal walking, running [*ref to ASIMO, Hubo, KHR from Kaist*], dextrous object manipulation using human-like hands [*ref to WE-4RII*], behavior modeling and socially interactive robotics [*ref to Kismet, Cog*], autism therapy [*ref to Keepon, Mataric*], facial expressions [*ref to H. Ishiguro's Geminoid*], procedural animation [*ref to iCat*] and others.

We are particularly interested in human-robot interactions using non-verbal modalities such as gestures and prosody. The motivation of our approach is similar to that of Cynthia Breazeal's with her robot, Kismet [*ref to Kismet*]. Kismet can interact with a person using garbled voices with prosodic qualities to respond, and shows facial expressions depending on its 'mood' which is determined from the interaction with the person, or lack thereof. We extend the concept of Kismet; from expressing/communicating solely through facial expressions, to expressing/communicating through the whole body gestures of head, arms, and torso. In contrast to Kismet, the puppets in The Muppets Show and Sesame Street are able to convey many expressions through their body gestures, yet their facial expressions do not change much, or do not change at all. How the puppeteers in The Muppets Show are able to animate the puppets so well, that the puppets become believably alive, is intriguing to us. We know for sure it is not in that the puppets are human-like in appearance, but rather, we believe, in their body gestures and character or perceived personality. Similarly, animated movies and films, either in 2 or 3 dimensions, can 'fool' the audience that the characters seen on screen do exist and believably alive [*ref to Lasseter's*]. Well-known animation methods such as Disney Animation Principles, and movement theories such as Laban Movement Analysis provide an extensive guideline for creating believable animations.

One particular concept from Laban Movement Analysis (LMA) is called phrasing. Pharsing in LMA is described as similar to phrases in music, and deals with where and how movement of the actor starts and stops, also how movement components (other LMA concepts) are combined together. In music, a song is divided in phrases. There is a classic musical rhythm of AABA, where a phrase pattern 'A' is repeated two times, followed by a different phrase pattern 'B', and ended with phrase pattern 'A'. We generalize these concepts for robot animation. Each phrase pattern is a gesture. Each phrase pattern (gesture) has observable spectral characteristics such as pitch and intensity in its joint angle data, which we map to acceleration and range of motion of the animation, respectively. GOOD IDEAS! Are the any references?

The idea of doing spectral analysis on motion data is not new. Unuma et. al [*ref to Unuma's*] used Fourier Analysis to interpolate or morph between two periodically-similar motion data. Amaya et. al. [*ref to Amaya's*] used Fourier Analysis to 'extract' motion characteristics by calculating the Fourier coefficient difference of an expressive motion (e.g. 'tired walk') from its neutral variant (e.g. 'normal walk'). The motion characteristics (e.g. 'tiredness') was then applied to a different neutral motion (e.g. 'normal run') and this second motion becomes expressive (e.g. 'tired run'). Spectral analysis IS THIS ALWAYS FOURIER OR MAY BE WAVELET, HADAMARD OR COSINE OR OTHER>? is also used in motion laboratories in hospitals to diagnose anomalies in a patient's musculoskeletal system. Also, in the medical field, spectral analysis is used to give prognosis on diseases such as Parkinson's

disease by analyzing tremors in the patient's movements [*ref to McNames' FIND PAPERS WHAT IS THERE?*]. Going beyond spectral analysis, Bruderlin and Williams showed that filtering methods and other common signal processing methods can be used to quickly modify or prototype motion characteristics such as exaggeration, and blending two motions together [*ref to Bruderlin and Williams*].

Our goal is to find a formalized method to create expressive robot behaviors, taking inspiration from techniques used in logic synthesis, signal processing, natural language processing (such as Regular Expressions), linguistics, music, and performing art. We are inspired by techniques from logic synthesis and Regular Expressions to generate the motion 'phrases.' The structure/grammar for the phrase generation will be influenced by the concept of phrases in music. Next, melodic information (e.g. pitch, intensity, tone, tempo, holds) is extracted from a music file, and then mapped to their animation counterparts (e.g. acceleration/deceleration, range of motion, repetitions, pauses) to modify the motion phrases. We create a library of gestures and program each gesture off-line based on the taxonomy of gestures from McNeills [*ref to McNeill's Hand and Mind*]. Each motion phrase will consists of a sequence of gestures from the gesture library. VERY GOOD IDEAS! EXTEND!

The ideas above are implemented in the context of Robot Theatre. The robot which we applied our method of synthesizing gesture to is called the Robot Actor. The sequence of gestures is referred to as Scenario. In this context, there is no feedback from the audience or user to the robot.

In Chapter 1 we gave a brief introduction to our problem, and a glimpse at the contribution of this thesis. Chapter 2 will review some researches on expressive robotics. In chapter 3 related researches in motion synthesis for performance robotics are presented. The proposed system and each component of the system is described in Chapter 4. We present our results and experiments in chapter 5. Finally, conclusions and future directions are presented in Chapter 6.

Chapter 2 – Related Works in Expressive Animations of Humanoid Robots

2.1 Interactive Humanoid Robots

Kismet is a robotic head developed by Cynthia Breazeal at MIT [ref to Designing Sociable Robots]. Breazeal used Kismet mainly as a platform to model social interaction behaviors, and in particular, emotive effects in social interactions. The behavior model used in Breazeal's seminal work was that of a child. The robot was programmed to always seeking attention/interaction. For example: the robot becomes 'happy' when a person is playing/interacting with it, and becomes 'sad' when nobody is interacting with it, or when it is being ignored. Kismet perceives user interactions through cameras (for face detection and object tracking), touch sensors, and prosody recognition. In return, Kismet responds by executing pre-programmed emotive facial expressions and prosodic voices without semantic language. Kismet was also used to model interaction norms such as turn-takings, gaze direction, and personal space [ref to Context-Dependent Attention System].

The WE-4RII robot was developed by the Takanishi Lab at Waseda University, Japan [ref to Effective Emotional Expressions with WE-4RII]. The WE-4RII robot has a face which is capable of many emotional expressions comparable to Kismet, a torso, and a pair of humanoid arms and hands. The robot is capable of many behaviors, those similar to Kismet's, such as object tracking, emotional facial expressions accompanied by speech/voice. Many other features include a pair of artificial 'lungs' for smell/chemical detection, Electro-Luminiscent sheets on its cheeks that changes colors, pressure sensitive grip, voice localization, and some touch sensing. Like Kismet, WE-4RII was also programmed with a mental model. The mental model for WE-4RII was derived from behavioral psychology theories such as Maslov's Hierarchy of Needs and C. L. Hull's behavior theory [ref to WE-4RII Website, Behavior Model of Humanoid Robots Based on Operant Conditioning]. The speed of the movements of the robot is affected by the amount of 'drive' levels from its emotional states, while the kind of expressive responses are chosen from a set of pre-programmed expressions. Hull was a psychologist who developed a theory that all living creatures' behaviors are based on motivations to satisfy basic biological needs such as hunger, thirst, pleasure, and pain avoidance.

What are the principles of Hulls' theory?

The Public Anemone is an animorphic (animal-like) robot developed by MIT [ref to Interactive Robot Theatre]. The Public Anemone itself is actually a Robot Theatre, where the anemone robot does a set of pre-programmed behaviors which repeat indefinitely in some kind of a stage. The audience, however, can interact with the robot and trigger the robot to perform some gestures. The whole stage is equipped with several cameras to detect the presence and positions of the audience (e.g. the number of people in the audience, skin tone tracking). For example: by approaching the anemone robot (e.g. trying to reach the robot with a hand), the anemone robot will perform a 'recoiling' gesture by moving away from the hand and shaking – as if in fear/defensive. Once the hand is moved away, (as if the robot no longer feels threatened) the robot goes back to doing its 'daily routine.'

Try to find Zeno by Henson, this is amazing robot to be soon delivered.

The robot iCat is a robot head which was designed to look like a cartoon cat character, built by Albert van Breemen et. al. at the Philips Research Laboratory [ref to iCat]. The iCat robot is capable of many facial expressions (e.g. sleepy, angry, happy, surprised), synchronized lip movements with speech, input through speech (speech recognition), vision (face and object detection), and touch. In many ways, the idea behind iCat is very similar to the Cynthia Breazeal's vision for Kismet: a research platform for human-robot social interaction. Of particular interest is the animation system for iCat. To animate iCat, a motion library is used, and each motion in the library was handcrafted using the Disney Animation Principles [ref to *Illusion of Life*]. The animation of the robot is managed using five Animation Channels, a merging logic, and transition filter [ref to iCat animation engine]. Will you explain in more detail what are merging logic and transition filter?

2.2. Signal-based Animations

All of the robots mentioned above use a set of pre-programmed motions as their response. The biggest issue with the pre-programmed motion libraries is that the robot can quickly become *boring*. This is because often there are only a few ready motions in the library, and there are few circumstances the robot can respond to. Thus, the result is the perception of limited interaction and repetitive responses. Researchers in the animation field try to find ways to enable real-time response generation for virtual agents (e.g. video game characters). Bruderlin and Williams showed that by representing motion data as signals, common signal processing techniques such as multiresolution filtering, waveshaping, timewarping, and interpolation can be applied to the motion data [ref to *Motion Signal Processing*]. As signals, motion data can be manipulated in real-time to be exaggerated, subdued, or blended with other motions. Unuma et. al. used Fourier Analysis to create transitions between two periodic motions using normalized coefficients [0,1] between the Fourier coefficients of the two motions [ref to Unuma]. Also, using Fourier Analysis, motion characteristics (e.g. 'tiredness') can be extracted by calculating the difference between the coefficients of a 'neutral' motion (e.g. walk), and its variation (e.g. tired walk). This extracted motion characteristic can then be applied to other motion (e.g. run) to create a similar characteristic on this other motion (e.g. tired run) [ref to Unuma].

Ken Perlin developed a method to generate pseudo-random noises that can be used for creating noise in animation (one-dimension) to animated solid textures (four-dimensions) [An Image Synthesizer]. Perlin's noise-generating method (popularly known as *Perlin Noise*) have been used to create noise in the movement of a virtual character [GPU Gems] or robot [Improv, Experiences with Sparky] to simulate those little movements such as breathing, blinking, fidgeting, or sways. Originally, Perlin Noise is often used to create and animate movements of textures in nature such as water, clouds, fire, and other elements [Curl-noise , Real-time 3D Clouds]. Perlin Noise is generated by creating a sequence of random noise. The sequence of noise usually starts with a smoothed (i.e. interpolated) low frequency noise to *n* number of higher frequency noise, where each subsequent noise frequency is an octave higher than the last ($fn = 2*fn-1$). The sequence of noise is then added together, with the contribution of each random noise decreases as the frequency of the noise signal increases (noises with higher octave have less contribution than the lower octave noises to the final Perlin Noise). The Perlin Noise is especially useful to alleviate the 'static look' when a virtual character or robot is idle; instead of being still without any movement, the little movements (i.e. noise) give an impression of breathing or heartbeats, thus giving the illusion that the agent is 'alive.'

Sergey Levine demonstrated a coordinated synthesis of prosody in speech with beat gestures [Body Language Animation]. The beat gestures (e.g. moving the hands up and down) and their mappings to certain prosodic characteristics are done using Hidden Markov Model (HMM). Levine used motion capture data synchronized with speech, and made his system learn the probabilities of observing a beat gesture given a prosodic characteristic is detected. Using the derived model, the prosodic characteristics from an arbitrary speech input (e.g. internet chat) are used to synthesize the beat gestures on the 3D human model arms and hands, such that the arm and hand gestures complement the speech by giving gestural stresses and emphasis, as indicated by McNeill [McNeill's Hand and Mind] .

Others used prosodic information of pitch and intensity to control the animation of a 3D head model [Natural Head Motion Synthesis, Prosody-Driven Head-Gesture Animation]. All these prosody-driven animation methods used a very similar approach. The mappings of prosody and head gesture of a person is learned using HMM, then the learned mappings are applied to arbitrary speech and the animation of a 3D head model.

2.3 Motion-driven Music

Camarri et.al. [Communicating Expressiveness] used information from motion to control the dynamics in music, such as tempo, pitch, and intensity. A user's motion is captured by camera, and the dynamics of the motion (such as: acceleration/deceleration and breaks/pauses) are analyzed using a technique called Silhouette Motion Images (SMI). SMI calculates a sum of a series of the person's silhouettes minus the current silhouette (a variation of Motion History Images) and extracts motion information such as sequences of pause and motion (stroke) phases, motion qualities (e.g. hesitation, fluency, rigidness, contraction, expansion, directness). The motion information is then mapped to certain controls of musical notes, such as pauses or duration of the note, the note pitch, and intensity or loudness (volume).

Chapter 3 – Goals, Hypothesis, and Evaluation Methodology

3.1. Goals

The main issue addressed by this thesis is on the animation of a humanoid robot for interaction or entertainment purposes. In particular, in the matter of alleviating the 'mechanical' movement qualities in robots, where the movements are mainly executed with constant speed or abrupt transitions (e.g. during change of directions), or complete motionless/rigidness when no motion is provided - how to make the movement of the robot *look and feel natural*. In other words, the goal of this thesis is to improve the quality of the movements of a robot from 'mechanical' to 'organic.' Thus, some of the criteria of achieving this goal are: to have both variable and constant speed, abrupt or smooth transitions, at the appropriate contexts, and some amount of ambient movements (e.g. breathing-like movements) when the robot has no motion to execute.

3.2. Hypothesis

From literature research, the idea of the relationship between symbolic and prosodic qualities in sound and motion is found to hold a big potential in improving human-robot interaction experience. Prosody has been shown to give enough affective (e.g. emotional) effects in communication [Kismet], and an even better experience when prosody is complemented with gestures [Levine, *Natural Head Motion Synthesis*, *Prosody-Driven Head-Gesture Animation* , Wall-E]. Thus, the focus of this thesis is on the mapping of prosodic information (such as pitch, intensity) to whole body gestures.

In most verbal conversations, gestures are often done synchronized with the prosody in the speech to create emphasis, clarify by adding illustration, and so forth. Levine [Levine] focused on the synchronization of prosody in speech to beat gestures using hand/arm motion. [*Natural Head Motion Synthesis*, *Prosody-Driven Head-Gesture Animation*] focused on the control of the head gesture also synchronized with prosody from speech. [Kismet] shows that using prosody in human-computer interaction is enough to create affect, although the prosody information is not directly matched to the motions of Kismet. All the above experiments are done using computer-generated 3-D model/graphics.

This thesis tries to generalize the aforementioned concepts in three directions. First, mapping prosodic information to other types of gestures such as iconic, metaphoric, and deictic, in addition to the beat gestures, or any combination of the gesture types from McNeill's gesture taxonomy [ref to McNeill]. Second, using prosodic information from music, which in addition to pitch and intensity information, there are also *rhythm* and *phrasing*, or information of *patterns*. And third, applying prosody-controlled animation concept from computer-generated 3-D models to a physical, humanoid robot.

3.3. Evaluation Methodology

To test our hypothesis, we show our robot platform performing a gesture or a sequence of gestures - a *Scenario* to an audience. The robot will perform the gesture (or gestures) in the context of a Robot Theatre - as if the robot is acting a scene in a theatrical play. We limit the duration of Scenario to be relatively short (about 10 seconds) [ref to *attention span*], so the audience can remember most of the robot's performance. On the first run, the robot performs the Scenario without rhythm and prosodic

qualities. Next, the robot performs the same Scenario with rhythm and prosodic qualities. We ask the audience to rate their preference between each performance using a 5-point Lichter scale (1 = very bad, 5 = very good).

3.5 Measurement

In the area of performance robotics, there is still a large, open discussion for a standardized, objective, and quantitative measure of the robot's performance. For the sake of clarity in the context of this thesis, 'performance robotics' refers to robots in the following categories: socially interactive, social robots (not necessarily interactive), robot actors (robots used in performing arts), androids, and personal robotics. Essentially, robots which primary function is to interact, communicate, and entertain with people using human communication modalities (e.g. speech, gestures).

In many studies in the field of performance robotics, the objective of the research is often on how the robot can create *affect*. Affect refers to the ability of the robot to express its own 'emotions' and also invoke and evoke emotions of people [ref to Picard's Affective Computing], such as: Kismet, WE-4RII, Zeno, iCat, AIBO. In other words, the quality of these types of robots is measured by the quality of their human-robot interaction capabilities, in particular using human communication modalities. The difficulty comes from the different (i.e. subjective) ways people perceive communication cues, depending on culture, social norms, context, and other factors.

For example, Brezeal used a questionairre form to evaluate the correctness of Kismet's facial expressions - how well people are able to recognize the emotional facial expressions of Kismet. Breazeal and her team also performed interaction evaluation to evaluate the interaction dynamics between a person and Kismet. For the interaction evaluation, an observer (or a group of observers) records the interaction between Kismet and the person, and the notes were discussed between the observers [ref to *Emotion and Sociable humanoid robots*]. A study of preference of personallities on iCat was done by interview [ref to *Animabotics*]. A group of children interacts with an iCat, each time the iCat was programmed with different personalities. The group of children was then interviewed on which personality they liked best. The WE4-RII from Waseda University was also evaluated for the recognition of facial and body emotional expression using a survey [ref to *Effective Emotional Expressions*].

Scholtz suggested that the evaluation of the quality of human-robot interaction depends on the *role* of the human in his/her interaction with the robot [ref to Scholtz]. Scholtz defined five roles: *supervisor, operator*, *mechanic, bystander,* and *teammate*. As a supervisor, the person oversees one or more robots while the robot(s) is working autonomously. The role of the person is then to make sure the robot(s) is doing its job, and only intervenes when there are changes to the tasks or problems with the robot, but does not directly control the robot. The role of the operator is to directly manipulate the robot, such as giving waypoints, programming trajectories, or controlling the robot's manipulators to perform a particular task. The mechanic is responsible to perform hardware-related assessment and repair, such as on the robot's actuators and sensors. The teammate is a person who works together with the robot(s) to perform a particular task (or tasks). Finally, as a bystander, the person works and exists in the same environment as the robot, but has no knowledge or training of the robot. The bystander has then to learn about the behaviors of the robot by him/herself. The role of the person/people with respect to the type of robot in this thesis falls into the category of bystander, where the person can simply observe the behaviors/actions of the robot. For the bystander role, Scholtz suggested the metrics: *predictability of behavior*, *capability awareness*, *interaction awareness*, and *user satisfaction*.

'Predictibility of behavior' measures user's expectations to the actual behavior of the robot. For example: when the user asks the robot a question, will the robot respond with an answer to the question, or do something irrelevant to the question? 'Capability awareness' refers to the degree of match between user's expectations of what the robot can do and what the actual capabilities of the robot. For example: if the robot is humanoid biped, users may expect the robot to be able to walk. 'Interaction awareness' measures the match between the user's mental model of the robot's interaction ability and the robot's actual abilities. For example: if the robot is dog-like (e.g. AIBO), does it respond to a pat on the head or stroke on its back like a dog? 'User satisfaction' measures user's interaction experience with the robot.

Chapter 4 – System Design

4.1. System Overview

Similar to the works that inspired this thesis [*ref to Levine, Natural Head Motion Synthesis, Prosody-Driven Head-Gesture Animation , Kismet, Music Marionette*], the system presented here takes one or more sound files as its input, and outputs a set of movement instructions for the robot. Figure 4.1 presents a high-level view of the system.

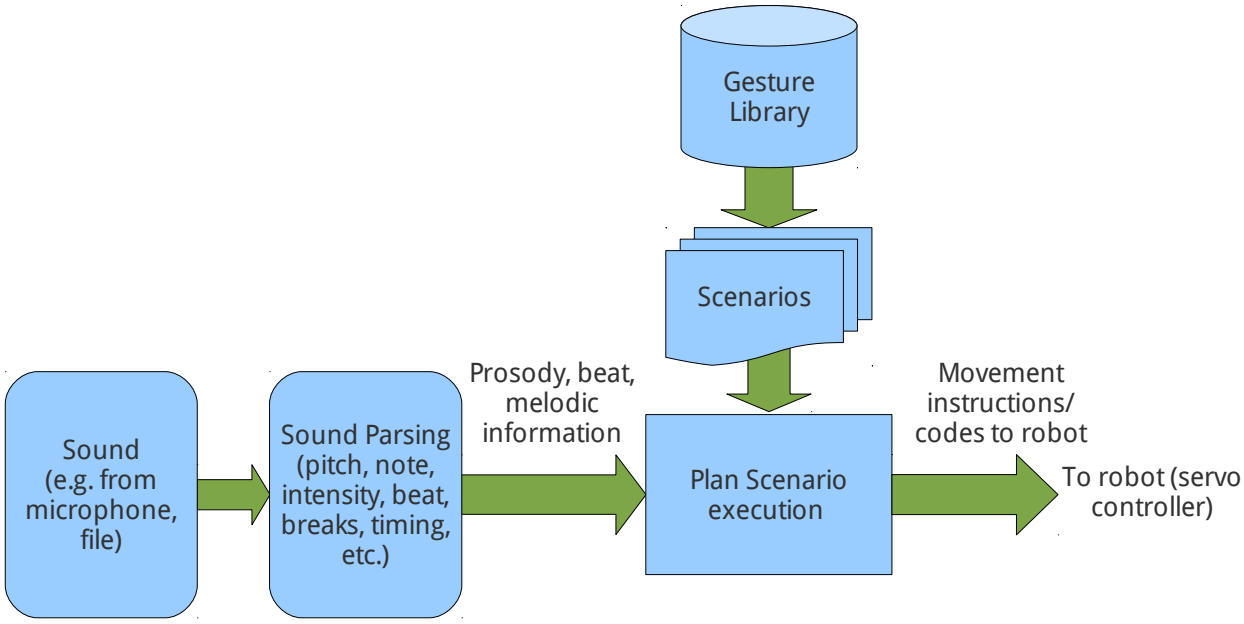

Figure 4.1 The block diagram of the proposed system

The central idea of the system is to have the robot gestures executed according to the dynamics in a sound input such as music and speech, but music/song in particular. Essentially, to see if affect in sound qualities can be translated to motion with the same kind of affect. Dynamics in the sound input are indicated by: start of a note beats in a song, the end/stop of a note, the intensity of the note, the change of the intensity of a note, the change in a sequence of notes (i.e. melody), the timing of the beats, the patterns of the note or beat (e.g. rhythm), the unvoiced parts (i.e. 'rests'), and finally, the transition to a note or rest and the quality of the transition (e.g. sudden or gradual). For example: when a note abruptly stops (i.e. a 'strong' rest), the dancer also stops her movement abruptly. This example also shows that in addition to the synchronized timing of the stop, the *quality* of the stop of the movement is also synchronized with the quality of the stop of the note. If the event is a gradual change such as of intensity, or pitch, a 'hit' may also be perceived as when the dynamics in the movement change according to the event. For example: when the intensity goes from low to high (e.g. *crescendo*), the dancer may gradually increase her movement velocities. Naturally, such reaction (i.e. expression) to the events in the sound is often a subjective interpretation of the person or performer. Thus, the system presented here mainly concerns with the time synchronization between the sound events. The 'interpretation' of the quality of sound events to the quality of movements such as range of motion, acceleration/deceleration, and speed, are assumed to be proportional by some empirical models using

normalized coefficients described below.

Since music is a structured arrangements of sound signals, which generates some pleasing qualities to the listener (in most cases), and with some indication that motion can be thought of as signals, would it be possible to create motion signals with the similar structure and pleasing qualities as in the music? In a sense, this hypothesis is a variation of Unuma's work in [ref to Unuma]: if characteristic functions can be extracted from motion data and represented as a signal, is it possible to use music signal as the characteristic function for a motion data? Also, music has been used to control in real-time coordinated execution of special effects in live entertainment (e.g. concerts, parades) such as through the MIDI Show Control (MSC) protocol [*ref to MIDI Systems and Control*]. Can this concept be extended to execute or synthesize motion? And finally, if sound events such as in speech can be detected and synchronized with the execution of gestures such as in [*Natural Head Motion Synthesis*, *Prosody-Driven Head-Gesture Animation*] to add expressiveness, would a richer sound information such as in music provide a more powerful expressive control of gestures? The idea is then to take/extract information from a piece of music, and use the information to create or manipulate robot motion.

The sound input is parsed to extract the prosodic and musical information in the sound such as: *pitch, amplitude/intensity, sustains, rests*, and tempo. Pitch refers to the frequency components that appears at a certain time window in the sound. For example: in speech sound, pitch information can indicate the pronunciation of certain vowels or consonants using formants. In music sounds, pitch indicates musical notes, and timbre (sound characteristics of different music instruments). Amplitude/intensity simply refers to the loudness of the sound at a certain point in time. 'Sustain' is defined in this thesis as the event when a note is played and sustained for a period of time. 'Rest' is the event when no sound appears either in the speech or song. Similar to 'note holds', 'restsre' is an event and has duration information. 'Tempo' indicates the timing of the sound, measured in the number of beats per minute (bpm), and only applicable to songs.

A second input to the system are gestures, or a sequence of gestures called Scenarios. A set of gestures is created off-line and stored in a Gesture Library. The gestures in the Gestures Library are selected to be representative gestures for McNeill's gesture taxonomy. The Gesture Library is discussed in more depth in Section 4.2.2. The user is provided with a graphical user interface (GUI) and have the freedom to select any number of gestures from the Gesture Library, and arrange the gestures in the desired order to create a Scenario. For example: using the GUI, the user may select the following sequence of gestures:

Gesture: [point up, point down, draw a box, look right]

and add them to the Scenario List. Then, the user may re-arrange the order of the gestures in the Scenario List to be:

Re-arranged gesture: [look right, point down, point up, draw a box].

Once the user is satisfied with the arrangement of the gestures, the Scenario List is saved to a file.

Then, the melodic information from the sound file is used to control the execution of the Scenario. For example: changes in intensity/amplitude will proportionally change the range of motion of the gestures; the louder the sound, the bigger the range of motion of the gesture. Another example: when a rest in the sound is detected, the whole movement stops according to the time and duration of the rest in the sound. In the event of a note hit (i.e. when a note or beat appears in the song), the next phase of the

gesture is executed. If the sound input is a song, a gesture may be executed according to the duration of a *musical phrase* in the song. These are just a few examples of how the information from the sound input can be used to control the execution of the Scenarios. The design details and full implementation of the system are explained below.

4.2 Input

4.2.1 Sound

As mentioned above, the sound input of the system can either be a speech sound or music. Currently, this study focused on music, and uses only sounds recorded as digital files have been used in the experiments, such as: .mp3, and .mid (MIDI).

In this study, instead of real-time listening and processing of sound, a digital sound file is used and analyzed off-line to extract the prosodic and melodic information. The MIDI (Musical Instrument Digital Interface) is a standard that allows electronic musical instruments to be connected and synchronized to an ordinary personal computer (PC) [*ref to MIDI*]. Incidentally, using MIDI, musical elements can be created digitally directly from the PC, especially periodic ones such as percussions. Music stored in digital files using the MIDI format (.mid) is mostly stored as *event messages* such as: *note on*, *note off* (or *release*), the *note* itself , *aftertouch*, *control change*, *program change*, *channel pressure*, and *pitch wheel*. 'Note-on' is the event which indicates the start when a note is being played on the musical instrument. This also sometimes referred to as 'onset'. Conversely, 'note off' indicates when a note stopped being played. 'Note off' may also referred to as 'release'.

The note information is coded in MIDI as a function of the frequency of the note (i.e. *pitch number*) by MIDI convention, which is defined as [*ref to Puckette's The Theory and Techniques of Electronic Music draft*]:

$$m = 69 + 12\log_2(f/440) \tag{4.1}$$

It is much easier to think of the note in terms of its frequency with respect to the pitch number:

$$f = 440 * 2^{\frac{(m-69)}{12}} \tag{4.2}$$

Where m is the MIDI code for the note which ranges from 0 to 127, f is the frequency of the note, 440 is the reference frequency of the note A above the middle C note, and 69 refers to the code for the note A. The number 12 refers to the twelve sub-intervals of the octave in the Western musical scale, i.e.: c, c#, d, d#, e, f, f#, g, g# a, a#, b. In this MIDI convention, the middle C (C4) note is assigned to pitch number 60 (f=261.63Hz*)*.

'Aftertouch' refers to the amount of pressure applied to a key (key press) after the note-on event, such as on an electronic keyboard. Aftertouch, then indicates the dynamics of the note. A strong pressure usually creates a loud sound (*forte*), while a soft pressure creates a quieter sound (*piano*). A gradual increase or decrease of pressure makes the note being played increasingly louder (*crescendo*) or quieter (*decrescendo*), respectively. 'Control change' is the message which indicates a change of MIDI controller or device that will synthesize the sounds. 'Program change' indicates a the program being

played by a MIDI device (i.e. an electronic music instrument). 'Channel pressure' denotes the average pressure being applied to a MIDI channel. While aftertouch only corresponds to individual key presses, channel pressure indicates the average pressure level for all the keys (e.g. of an electronic piano) at a given moment. For example: a musician might not apply the same amount of pressure to all keys as he plays the music on an electronic piano. At every instance of time, channel pressure automatically calculate the average pressure, and applies the the difference to all the keys so they sounded like being played with equal amount of pressure. 'Pitch wheel' indicates when a note pitch is being 'slided' up or down. So for example: a middle C note with the pitch wheel increased (up), may sound like the note middle E. These last five MIDI features are currently not used in the proposed system here. The usefulness of these features in creating expressive animation needs to be explored in future studies.

Because all these information are already explicitly recorded in the MIDI sound file, the MIDI format is convenient to use to extract melodic information from a song, and incidentally, as control signals. In fact, an industry-standard MIDI Show Control (MSC) protocol was established in 1991 which has been used in many entertainment performances which use music-synchronizations [www.midi.org].

4.2.2 Gesture Library

The Gesture Library is a database of gestures. The gestures in the Gesture Library are categorized into *iconic gestures, deictic gestures,* and *beat gestures* based on McNeill's taxonomy of gestures [McNeill]. For each iconic, deictic, and beat gesture category, several gestures are created manually and off-line. For the iconic gesture category there are two gestures: (drawing a) box gesture, and horizontal line gesture. In the deictic gesture category, all the gestures are pointing gestures in six directions: up, down, left, right, forward, and back. And for the beat gesture category there are: up-down, left-right, and forward-back gestures.

Originally, McNeill's taxonomy of gestures consists of five categories. In addition to the three aforementioned gestures, there are also *metaphoric gestures*, and *cohesives*. However, metaphoric gestures and cohesives can be performed using either iconic, deictic, or beat gestures. What determines a gesture is metaphoric instead of, say, an iconic gesture, is the *context* in the conversation where the gesture is performed. For example: clapping the hands together. When the gesture is performed to illustrate an impact (a concrete action), the gesture is considered an iconic gesture. When the gesture is performed to illustrate togetherness (an abstract concept), the gesture is considered a metaphoric gesture. Cohesives, on the other hand, is more of an *event*, where a speaker interrupts his/her story to give some extra illustration or clarification (using gestures) to the listener. For example: a speaker is telling a story how her friend is trying to unlock his locker: “he tried to unlock his locker's lock... you know, the round one <*making a C shape with the index finger and thumb – an iconic gesture*> where you turn the knob dial to enter the code <*making a pinching and twisting gesture*>... and the knob fell off...” In that story, the part where the speaker goes off to illustrate the lock, is the cohesive part. Notice that the gestures she made are all iconic gestures. Since the robot will not speak or respond with speech/sentence output, thus no speech-dependent contexts, the metaphoric and cohesive categories are excluded from the Gesture Library.

4.2.2.1 Gesture Representation

Each gesture constitutes of three phases: *preparation, stroke*, and *recovery*. The preparation phase is

the initial movement of the gesturing part (e.g. end effector such as hand) from a *rest* (or *home*) position to the *gesture space* where the *stroke* phase will take place. Next, the stroke phase is where the actual gesture takes place, such as drawing a circle. And finally, the recovery phase is when the gesturing part moves back to its home position. For example: a person is to do a circular gesture over his head with his hand. Assume the home position of the person's right hand is on his right side. The person then has to move his right hand above his head (the preparation phase). Once his right hand is above his head, he draws the circle (the stroke phase). After he has done the circling gesture, he puts down his right hand back to his right side/home position (the recovery phase). Let's represent the preparation phase with the bold capital **P**, the stroke phase as **S**, and the recovery phase as **R**. Therefore, a gesture **G** can be represented as a string of characters:

$$G = P\,S\,R \tag{4.3}$$

Each character representation of the gesture phases (**P**, **S**, and **R**) assumes one stroke (i.e. movement in one direction, no change of direction). But of course, a gesture may involve more than one stroke. To clarify: there are three phases in a gesture: preparation, stroke, and recovery phases. Each phase consists of at least one movement, referred to as a *stroke*; not to be confused with the stroke *phase*. The preparation and recovery can be assumed to always involves only one movement. Therefore, a little modification to equation 4.3:

$$G = P\,S^{+}\,R \tag{4.4}$$

Equation 4.4 is the general form of a gesture representation, which takes into account the possibility of having more than one stroke in the stroke phase, represented by the symbol $\boldsymbol{S}^{+}$. The superscript '+' following **S** indicates there should be *at least one* stroke or more in the stroke phase. For example: a gesture of drawing a box would have four strokes in the stroke phase: up, left, down, right; each represent one side of the box. The box gesture may be written symbolically as:

$$G_{box} = P_{box}\,S_{box}\,R_{box}$$

Where $\boldsymbol{S}_{box}$ consists of four strokes (in order) $\boldsymbol{S^0}_{box}$, $\boldsymbol{S^1}_{box}$, $\boldsymbol{S^2}_{box}$, and $\boldsymbol{S^3}_{box}$. The superscripts indicates the index of the strokes in the stroke phase, while the subscript indicates the name of the gesture.

$$S_{box} = S^0_{box}\,S^1_{box}\,S^2_{box}\,S^3_{box}$$

Thus:

$$G_{box} = P_{box}\,(S^0_{box}\,S^1_{box}\,S^2_{box}\,S^3_{box})\,R_{box}$$

Another example is a deictic (pointing) gesture. Deictic gestures are considered a special case in the stroke representation since the stroke phase does not involve additional movements such in the case of the box gesture. Instead, the 'stroke' of a deictic gesture is often just a *delay*. Therefore, a deictic gesture may be represented as:

$$G_{deictic} = P_{deictic}\,S(t)_{deictic}\,R_{deictic} \tag{4.5}$$

The subscript 'deictic' is supposed to be the name of a deictic gesture, such as: point up, point down, and so on. The stroke phase is now a function of time (i.e. delay) *t*. When the gesture is executed, the

gesturing part will first execute the preparation stroke ($\boldsymbol{P}_{deictic}$), then the end position of the preparation stroke will be held for time *t*. After time *t* has elapsed, the response stroke ($\boldsymbol{R}_{deictic}$) is executed, and the gesturing part is returned to its home position. If *t=0* (i.e. no delay), the stroke phase is ignored, and the gesture is represented as:

$$\boldsymbol{G}_{deictic} = \boldsymbol{P}_{deictic}\,\boldsymbol{R}_{deictic} \tag{4.6}$$

Which means the gesture only consists of two strokes: the preparation and recovery strokes.

4.2.2.2 Gesture Transition

A gesture may be followed by another gesture. To create a smooth transition from one gesture to the next, let's assume that the recovery stroke of the first gesture and the preparation stroke of the next gesture will be blended using some transition $\boldsymbol{T}$. Suppose there are two gestures that are to be executed in sequence: $\boldsymbol{G_1}$ and $\boldsymbol{G_2}$. Also, let's suppose gesture $\boldsymbol{G_1}$ has two strokes in the stroke phase, and gesture $\boldsymbol{G_2}$ has three strokes in the stroke phase.

$$\boldsymbol{G}_1 = \boldsymbol{P}_1 \boldsymbol{S}_1^0 \boldsymbol{S}_1^1 \boldsymbol{R}_1 \quad \text{and} \quad \boldsymbol{G}_2 = \boldsymbol{P}_2 \boldsymbol{S}_2^0 \boldsymbol{S}_2^1 \boldsymbol{S}_2^2 \boldsymbol{R}_2$$

The transition $\boldsymbol{T}$ is defined as a function of the recovery stroke of $\boldsymbol{G_1}$ ($\boldsymbol{R_1}$), and the preparation stroke of $\boldsymbol{G_2}$ ($\boldsymbol{P_2}$). The problem now is to define what the transition function *is*. Van Breemen used a *transition filter* which happens within a certain time window (t_1, $t_1 + t_t$] after the end of the first motion, at the beginning of the second motion [ref to iCat animation engine]. Therefore, the first motion will always be executed until the last position, while some of the early parts of the second motion is interpolated from that last position. The transition is calculated from that last position of the first motion, using a scaling coefficient α, to interpolate to the second motion. The transition filter is shown in equation 4.7 (from [ref to iCat animation engine]).

$$s_i(t) = \begin{cases} s_i^A(t) & t < t_t \\ \alpha(t) - s_i^B(t) - (1-\alpha(t)) - s_i^A(t) & t_1 \leqslant t < t_1 + t_t \\ s_i^B(t) & t \geqslant t_1 + t_t \end{cases} \tag{4.7}$$

Where $s_i(t)$ is the position of transition at time *t*, $s^A{}_i(t)$ is the position of motion A at time *t*, t_t is the start time of the transition window, t_1 is the time of the end of motion A, $\alpha(t)$ is the scaling coefficient which is a function of time *t*, and $s^B{}_i(t)$ is the position of motion B at time *t*.

Van Breemen noted regarding the transition filter:

"The scalar α linearly depends on the time; making it depend exponentially on the time will make the interpolation even smoother."

Therefore, per Van Breemen's suggestion, the sigmoid function is chosen to be the function for the scaling coefficient $\alpha(w)$.

$$\alpha(w) = \frac{1}{1 + e^{-w}} \tag{4.8}$$

The transition function $\boldsymbol{T}$ is slightly different to Van Breemen's transition filter. Instead of doing the transition after the first motion (i.e. gesture) is finished, where the transition function is only being applied to the second gesture, the transition starts slightly before the end of the first gesture. The transition is now being applied to both gestures, for a transition window $w = [-5, +5)$[1]. The scaling coefficient applied on the first gesture is an inverted sigmoid where the sigmoid function decays to zero, while for the second gesture a non-inverted sigmoid function (increasing to one) is used. The complete transition is a function of time of the sum of the product between the inverted sigmoid with the position data of the recovery phase of the first gesture, and the product of the sigmoid function with the preparation phase of the second gesture. The transition window parameter is added to the transition function. Or:

$$\boldsymbol{T}(w) = ((1-\alpha(w)) * \boldsymbol{R}_i) + (\alpha(w) * \boldsymbol{P}_i + 1) \quad \text{for } -5 \leq w < 5 \tag{4.9}$$

Using the transition function in equation 4.9, the two gestures are 'blended' by gradually nullifying the recovery phase of the first gesture (i.e. will never reach the home position), while at the same time gradually increasing the contribution of the preparation phase of the second gesture. The result is a gradual transition from the first signal to the second signal which the transition gradually begins before the first signal ends. Figure 4.2 illustrates the effect of the transition function between two arbitrary signals.

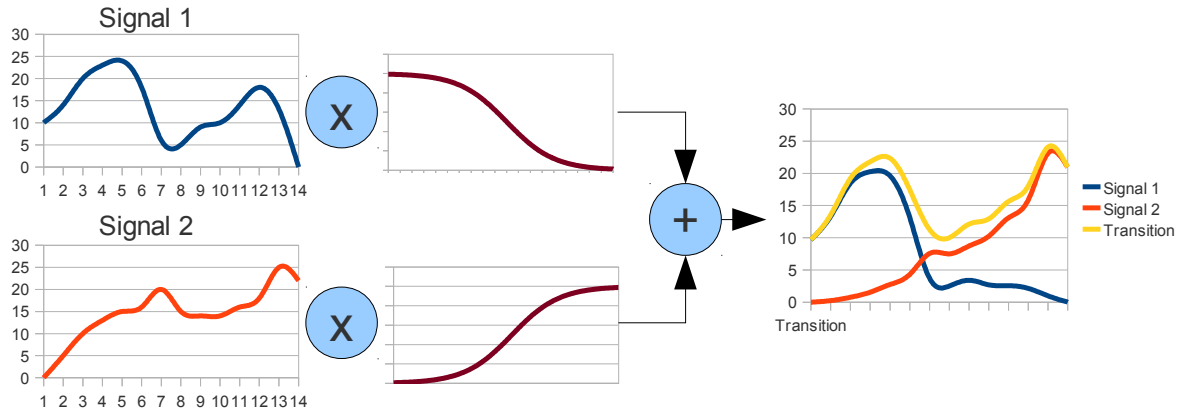

Figure 4.2 Transition Function

Also notice in Figure 4.2, that Signal 1 ends with 0, and Signal 2 starts with 0. This is to illustrate the convention that is used in the proposed system: every pre-programmed gesture has to start and ends with the HOME position.

This transition behavior is also aligned with a principle in the Disney Animation Principle about motion transition called *overlapping*. The overlapping principle says that to create continuation between movements, the second movement must already started before the first movement is completely finished [*ref to Overlapping*].

1 [-5, +5) is the range where the value of the sigmoid function approaches 0 and 1, respectively [ref to Wolfram Alpha on Sigmoid].

4.3 Processes

There are four main processes in the system: parsing prosodic and melodic information from the sound input, arrangement of gestures into Scenarios, scheduling/timing/planning the execution of the gestures in the Scenario based on the prosodic and melodic information, and finally translating the execution plan into instructions for the servo controller board. Each of these processes are explained below.

4.3.1 Sound Parsing

Sound parsing is the process to extract prosodic and melodic information[2] from a sound input. This process can be thought of as the *listening* and *analyzing* process. Robert Rowe created the Cypher software to analyze and synthesize music in MIDI format [ref to Rowe's Interactive Music Systems 1993]. The sound parsing process presented here is analogous to the listening component in Rowe's Cypher. The listening component in Cypher 'listens' to sound input and try to extract feature information from the input such as *register* (pitch information), *density* (simultaneous events), and *dynamics* (changes in tempo, loudness), which is the same task the sound parsing process tries to achieve here.

In this thesis, the information of interest are: *note-on/note-off*, *pitch, intensity, beat, tempo, rhythm, rests,* and *sustains*. Note-on or note-off is the MIDI terminology for the event when a certain note is struck or released, respectively. Pitch refers to the detection of a fundamental frequency in a time window of the sound (e.g. a particular musical note). Intensity refers to the measured sound-pressure level (in dB SPL) [ref to Praat on Intensity]. Beat refers to the periodic low frequency (downbeat) and high frequency (upbeat) bursts in songs, which usually are created by percussion instruments such as drum (except in classical music). Tempo determines the period of the beats. Rhythm refers to patterns of beats, or pitch contour (i.e. melody). Rests refers to unvoiced parts in the sound. Sustain is the event when a certain sound or note (i.e. pitch) is being voiced for some length of time.

Extracting prosodic information from speech, and melodic information from a song are often done using spectral analysis such as Fourier transform, and sometimes using functions in time domain. For example, in [*Prosody-Driven Head-Gesture Animation*], pitch information is obtained by analyzing the sound in the time domain using the autocorrelation method. The autocorrelation method is a pattern-finding method which also has been applied in the financial field. Autocorrelation measures the similarity of a signal with several delayed versions of itself. The pattern is found at the delay where the measured energy between the matched signals is the highest. The autocorrelation method also has been used to analyze a song to find beats [ref to *Techniques for automatic music transcription*], but only works well on a select few types of music where the bass sound is prominent. While the tempo of the beat can be approximated, the phase (e.g. start) of the beat cannot be determined using only autocorrelation. Eric Scheirer presented an algorithm that detect beats and the phase of the beats (when the beats happen) using frequency filterbanks, envelope extractors, half-wave rectifiers, and comb filters [Scheirer].

Ideally, it is desired to be able to extract all the above prosodic and melodic information from arbitrary sound input in real-time (e.g. listening through microphone). The Musical Marionette project detects two frequency bands (20 – 100Hz and 400Hz and beyond) using Butterworth filters by 'listening' to a

2 Prosodic and melodic information refers to mostly pitch, intensity, rhythm (contour) information from sound. The only difference is 'prosody' is the term used for speech sound or vocal, while in this thesis, 'melodic' information refers to the same set of information but in a song/music.

song in real-time [*ref to Musical Marionette*]. The 'listening' is achieved by directly connecting the audio output line to the filters, and subsequently to an ADC. This way, ambient noise from the environment is prevented from disturbing the song signal which would make the filtering and detection more complicated, such in the case of taking the song input from a microphone. The Musical Marionette experiment illustrates one of the challenges of extracting information from sound in real-time, which is noise.

Using MIDI files, most of the music information are stored as events, which can be extracted using simple parsing. The MIDI files used as inputs are formatted as XML files, and not directly the .mid files. The .mid files were read by the Muse program into Muse projects, and it is this projects that are saved as Muse file format (.med or .mpt) in XML form. The main part being extracted from the .med file is as follows:

```
<muse version="2.0">
  ... <!-- Additional setting information (non-musical) -->
  <song>
    ...  <!-- Additional setting information (non-musical) -->
    <miditrack>
      ...  <!-- Additional setting information (non-musical) -->
      <part>
        <name>Lullaby of Birdland</name>
        <poslen tick="0" len="218042" />
        <selected>1</selected>
        <color>0</color>
        <event tick="1152" type="1" a="262145" />
        <event tick="1152" type="1" a="7" b="100" />
        <event tick="1152" type="1" a="10" b="64" />
        <event tick="1156" type="1" a="11" b="105" />
        <event tick="1156" type="1" a="91" b="80" />
        <event tick="1156" type="1" a="93" />
        <event tick="1532" len="568" a="92" b="110" c="64" />
        <event tick="1534" len="568" a="80" b="89" c="64" />
        <event tick="1540" len="558" a="56" b="85" c="64" />
        <event tick="1542" len="550" a="55" b="98" c="64" />
        <event tick="1542" len="554" a="60" b="89" c="64" />
        <event tick="2196" len="96" a="92" b="110" c="64" />

        ...  <!-- shortened -->

        <event tick="215692" len="2342" a="55" b="98" c="64" />
        <event tick="215694" len="2336" a="51" b="89" c="64" />
        <event tick="215696" len="2330" a="48" b="89" c="64" />
        <event tick="215700" len="2334" a="32" b="89" c="64" />
      </part>
    </miditrack>
    ...
  </song>
</muse>
```

Figure 4.3 A snippet of the .med format.

4.3.1.1 Timing

The most important components in this data are the ones in the <event /> tags. The '*tick*' parameter indicates the time of the events in units of 'ticks.' After some observations it was found that the convention used by Muse notation is: one beat = 384 ticks. Therefore, to calculate the position of each beat in number of ticks, one can simply multiply the beat number by 384. For example: beat number 5 starts at tick: 5 * 384 = 1920. The '*type*' parameter indicates other MIDI control events such as Pitchbend, Control Change, Program Change, and other controls. Those control events are currently unused in the current proposed system, and can be ignored for now. As mentioned above, future research may utilize these control events to possibly achieve more powerful motion control systems.

The '*len*' parameter indicates the length or duration of a note being played. Again, the unit is in ticks. So, to calculate how many beats the note is played (e.g. quarter-note, half-note, two beats) the value of '*len*' is divided by 384. For example: in the above event list at tick=1532, *len*=568. 568/384 = 1.48 or roughly one and a half beats. The '*a'* parameter corresponds to the actual note (in MIDI code) being played. For example: (again) from the event list above at tick=1532, *a* = 92. This value means the note being played is g#6 or note g# at octave number 6. Or, in terms of frequency:

$$f = 440 * 2^{\frac{(92-69)}{12}} = 1661.22\ Hz$$

In the proposed system, the start time (t^i_{on}), end time (t^i_{off}), and duration (t^i_{len}) of a note *i* is stored in units of ticks. From time to time, the units may be converted into seconds to be used for certain functions such as delays, or frequency analysis.

4.3.1.2 Rests

Rests are defined as the events in the sound when there are no pitch occurs, or silent which usually lasts for about one beat or more. To detect rests, the time between the last was played (I.e note-off event) and the time of the next note starts (i.e. note-on) is calculated. If the difference is equal or more than the duration of one beat, then a rest is detected. Rests result in pauses in the robot's motion; essentially, the robot will stop moving according to the duration of the rest. For example: a music with tempo of 100 beats per minute (bpm) means that the duration of each beat is 60 seconds/100 beats= 0.6 seconds for each beat. Suppose a rest is detected for one and a half beats, then for 1.5*0.6 seconds = 0.9 seconds the robot will stop moving.

There are instances when there appears to be no new note-on events for several beats, but the last note that was played was being *sustained*. Sustained means the note still being played. In this case, a rest event is not said to occur, and the robot will keep moving according to a specified position. Rests are calculated as the following:

```
1. For every beat:
   1. Is there a note being played (sustained) OR a new note event
      occurs?
      1. If yes:
         1. If this is not the beginning of the song: record the
            last rest information
         2. set rest flag = False
         3. count rest = 0
      2. Otherwise:
         1. If this is not the beginning of the song: record the
            last note information
         2. set rest flag = True
         3. count rest += 1
```

```
    2. Evaluate next beat
```

Figure 4.4 Rest detection process

4.3.1.3 Note-on and Note-off Velocities

The '*b*' parameter inside the <events /> tags denotes the *note-on velocity* of the note, which indicates the amount of force applied to the keys (e.g. on an electronic keyboard) at the moment the note was struck. Note-on velocity corresponds to the *loudness* of the note at strike; the stronger the strike, the higher the note-on velocity, and the louder the note will be played. The opposite is true with weaker force.

The value of note-on velocity ranges from 0 to 127, with 0 being the least amount of force (or considered as *note-off* event), and 127 as the most force (loudest). The '*c*' parameter is the parameter for the *note-off velocity* of the note, which indicates how quickly a note is to be released (i.e. 'decays') with a range of: 0 (slowest) to 127 (fastest) [*ref to MIDI Specification*]. If not specified, the default value for the note-off velocity is 64. In the General MIDI standard, note-off velocity is almost never used explicitly to reduce the number of bytes that must be transmitted. For each note-on and note-off event, three bytes must be sent: the first byte indicates the event (note-on or note off), the second byte is the note number, and the third byte is the velocity value. Suppose there are three notes being played. If both note-on and note-off messages must be sent, there are a total of six bytes to send. The recommended practice in the General MIDI standard is that note-on velocity of value 0 is equivalent to a note-off message. Therefore, the improved message format is to only send the note-on byte once, and the remaining bytes are evaluated in pairs (every two bytes) as note-on messages until a new event type byte is received. By reducing the number of bytes to be sent, the messages become more compact, and communications between MIDI devices become more efficient. This is important to note as to take account for MIDI input files that may or may not use the General MIDI where note-off information may not always appear. When no note-off value is provided, the MIDI standard gives the default value of 64, which is about in the middle of the velocity range.

There is a confusion as to what physical quantity the note-on velocity actually correspond to. According to the MIDI specification, note-on and note-off velocities determine the *volume* the note should be played at strike and release, respectively [*ref to MIDI spec*]. However, it has been reported that the actual output volume on which the note is played varies depending on the master volume and/or design of the MIDI output device [*ref to MIDI systems and control*]. Therefore, it seems reasonable to treat the velocity value [0,127] as some kind of a relative scaling coefficient instead of absolute volume values. Treating the note-on/off velocity value as a scaling coefficient makes it convenient to use as a modifying parameter for acceleration, which will be shown later in this section and in Section 4.3.3.3. Moreover, the term 'loudness' is considered to be a subjective psychological quality that differ from person to person [*ref to Signal, Sound, and Sensation*]. For this reason, 'loudness' is normally measured in dB which does not give an absolute value of loudness (but may be inferred), but instead gives the change in magnitude from a reference level of 'loudness'. To avoid confusion and improper use of the term 'loudness', in this thesis note-on and note-off velocities are associated to the *amplitude* of the sound signal.

Associating the note-on and note-off velocities with amplitude has several advantages. First, now there is a direct association of note events with a physical quantity in actual sound signals, instead of subjective quality of 'loudness'. Consequently, suppose the sound input is a sampled sound (e.g. .mp3, .wav) instead of MIDI, the amplitude of the sampled sound can directly be read – hence the

amplitude information needed by the proposed system in this thesis. Second, the motion data of the robot can be thought of as a set of signals, each for one degree of freedom. These so-called 'motion signals' represent motion as position or joint angles (i.e. range of motion) over time [*ref to Bruderlin and Williams*]. Since the motion can be represented as signals with the amplitude representing the range of motion, the amplitude of a sound input may be directly correlated to the amplitude of the sound signal proportionally, for example.

Lastly, the note-on and note-off velocities are directly related to the shape of the attack and release phase of the MIDI attack-decay-sustain-release (ADSR) envelope. In particular, note-on velocity determines the 'attack' rate (i.e. how fast to reach its maximum peak amplitude), and note-off velocity determines the 'release' rate (i.e. how fast to reach amplitude of zero). A sound in MIDI is created from a library of sampled sound. The sound is played by looping the sampled sound according to the duration defined by the user or MIDI controller (Figure 4.5). In Figure 4.6, the ADSR envelope is used to synthesize the MIDI events into sound by modulating the note signal with the ADSR envelope, thus creating a sound which runs in four phases: from zero to the initial peak amplitude (attack), followed by a decrease in amplitude to the desired level (decay), maintain the desired amplitude level for some time (sustain), and finally reduce the amplitude back to zero (release).

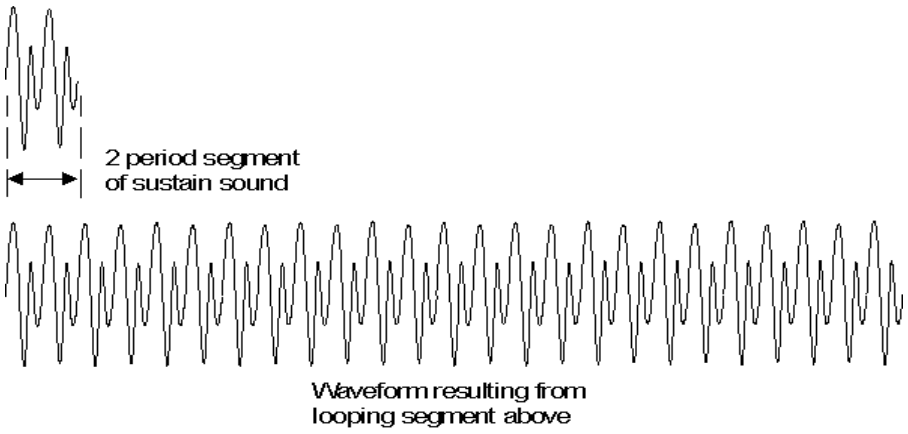

Figure 4.5 A sampled sound segment and the same segment looped
(image source: *http://www.harmony-central.com/MIDI/Doc/tutorial.html*)

Assuming that the MIDI velocity parameter corresponds to amplitude, then the change of velocity over time in increasing (louder) or decreasing (quieter) manner in musical terminology are called *crescendo* and *decrescendo*, respectively. Therefore, to capture crescendo or decrescendo in the music, the amplitude information is recorded as: the instantaneous amplitude at note strike ($v_{note\text{-}on}$), and the amplitude gradient:

$$A^i(t) = \frac{v^i_{note-off}(t + t^i_{len}) - v^i_{note-on}(t)}{t^i_{len}} \tag{4.10}$$

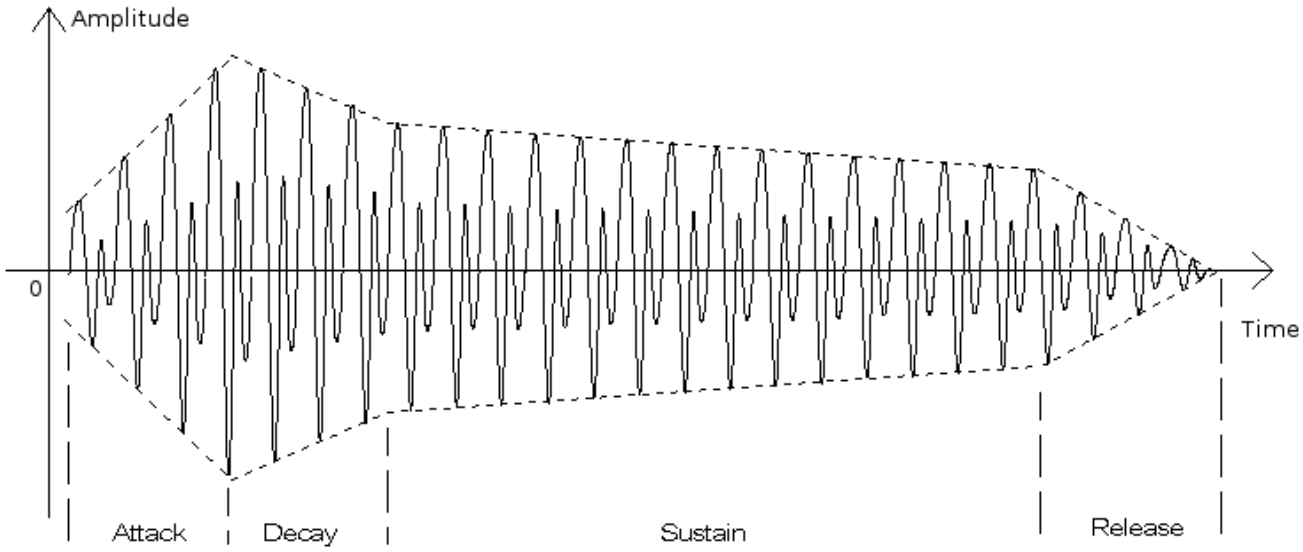

Figure 4.6 The looped sample from Figure 4.5 modulated with the ADSR envelope
(image source: *http://www.harmony-central.com/MIDI/Doc/tutorial.html*).

Where $A^i(t)$ is the gradient (rate of change) of the amplitude of note *i* which starts at time *t,* over the duration of note *i* (t^i_{len}). $v^i_{note-off}$ is the note-off velocity of note *i*, and $v^i_{note-on}$ is the note on velocity of note *i*. When $A^i(t) > 1$ then the amplitude increases over time (crescendo), $A^i(t) < 1$ is decrescendo, and $A^i(t) = 1$ means the amplitude level stays the same.

4.3.1.4 Density

Borrowing the term used by Rowe [*ref to Rowe's Interactive Music Systems*], 'density' indicates how many note-on events happen simultaneously in one beat. For example: from the event list above, the last five events has ticks: [215690, 215692, 215694, 215696, 215700]. This indicates that between beats 561 (= 215690/384) and 562 (= 215808/384), there are five notes being struck at the same time. The density is represented as *D(b),* which simply states the number of note-on events that occurs between the beat *b* and *b+1*. Currently, simultaneous note events are consolidated into one event. One note is selected to represent the event based on the note-on velocity as mentioned in Section 4.3.1.6, and the 'energy' is the average of the note-on velocities as shown in Section 4.3.1.5.

4.3.1.5 Musical 'Energy'

Musical energy refers to the average intensity or spectral power per unit time. In musical interpretation, the term 'energy' is a misnomer of the physical unit of energy (Joule) to describe the expressive part of a music. For example: in contemporary music, a song is often structured as intro, verse, refrain, verse, refrain, bridge, refrain. Often, artists (musician, dancers) describes the 'energy' of the music to be 'building up' towards the refrain. At the end of the refrain, the 'energy' climaxes (i.e. reach its highest point), then the 'energy' level drops down ('crashes') back to the level at the beginning of the first verse, the second verse is started, and the pattern is repeated. Notice the usage of the term 'energy', for the lack of better term to describe the expressive qualities in the music. However, there is an observable indication and characteristics in physical quantities as to what is referred as 'energy' here.

A dance teacher and a music teacher gave similar descriptions of what can be interpreted as a 'high energy' music, or part of a music score. A music with fast tempo does not necessarily translates to 'high energy'. Instead, the *busy-ness* in the music is an indication of the level of energy. 'Busy-ness' is the amount of musical elements (e.g. instruments), activities, or events that happen at a given point in the music. Thus, by this definition, density is one of the musical energy indicators. For example: at a certain part of a music, many musical instruments are being played at the same time. Or, in other cases, when a section of the music is much louder than the rest (i.e. average) of the music. To the best of my knowledge, currently there is no consensus on what physical quantity does this quality correspond to. One candidate measure that seems appropriate is *sound intensity* ($I(t)$). Intensity is measured in units of work per area or Watts/m^2. Intensity can be calculated as the square of acoustic pressure levels (in units of Newton/m^2), but more often measured in decibels (dB) with the reference intensity (I_0) = 10^{-12} Watts/m^2. In this sense, the sound intensity is analogous to power in electrical signal measurements [*ref to Hartmann Signal, Sound, and Sensation*].

Another candidate of musical 'energy' measurement is by power spectral density (PSD), which is defined as:

$$S(f) = \int_{f_1}^{f_2} R(\tau) e^{-2\pi i f \tau} d\tau \qquad (4.11)$$

Where $R(\tau)$ is the autocorrelation function of signal $s(t)$. Thus, the PSD is equivalent to the Fourier transform of $R(\tau)$:

$$S(f) = F(R(\tau)) \tag{4.12}$$

What makes PSD a good measure for musical 'energy' is that PSD is the measure of power in the signal as a function of the frequency components in the signal. Since musical notes are related to pitch which is closely related to frequency, and timbre – sound quality that distinguishes sounds of different musical instruments, male from female voices – also depends on frequency [*ref to Hartmann's Signal, Sound, and Sensation*], PSD is a good indication for content of notes and timbre in a segment of music. Both intensity and PSD are good indicators for musical 'energy' but on different domains; intensity is in the spatial domain (Watts/m^2) while PSD is in the frequency domain (Watts/Hz). Therefore, if the input signal is analog, the musical 'energy' can be calculated as the *power* at a time window (e.g. measure, phrase).

In the case of using MIDI, the 'energy' is derived directly from the value of the note-on velocity. When there are multiple notes played at the same time in a beat, the musical energy E in that beat is calculated as the average note-on velocity energy:

$$E = \frac{1}{N_{notes}} \sum_{i=0}^{N_{notes}} (v^{i}_{note-on})^2 \tag{4.13}$$

4.3.1.6 Melodic Surface

Melody information involves the changes of pitch and duration/transition of the notes in the measure or phrase. Melucci and Orio presented a method to classify and search musical data based on the melodic contents in the music using a feature called *melodic surface* [*ref to Melucci*]. A melodic surface is a short melodic segment in a music, which Melucci and Orio analogize with a keyword in a sentence. Thus, a melodic surface is a short melody segment that listeners can use to identify the musical piece.

To segment the music piece into melodic surfaces, a segmentation method is required. Melucci and Orio used a weighting method proposed by Cambouropoulos called Local Boundaries Detection Model (LBDM) [*ref to Cambouropoulos*]. LBDM identifies the *boundaries* between two melodic surfaces by evaluating each note and give weights on the note transitions based on the relationships of *musical intervals*, *note durations*, and *musical rests* between the note with the notes before and after it. The weighting scheme is as follows:

- Musical intervals:
  - The weight of the transition with the larger interval is added by 2, and the smaller interval by 1. Otherwise, if the interval does not change, the weight does not change.
- Note durations:
  - If the duration of the previous note is longer than the duration of the next note, then the transitions with the previous and next note is added by 4 and 1, respectively.
  - If the duration of the previous note is shorter than the duration of the next note, then the transitions with the previous and next note is added by 3 and 2, respectively.
  - If the duration are the same, no weight is added.

- Musical rests:
  - If there is a musical rest (rest), a weight of 4 is added to the transition between the note and the musical rest.

The boundaries are detected by examining the trend of the weights and looking for the local maxima.

In this thesis, a simplified form of Cambouropoulos' method is used. A melodic surface boundary is detected based only on the rest events between melodic surfaces. That is, if there exists a rest event following a series of note-on events, then the start of the rest event marks the end of a melodic surface. Consequently, the end of a rest event is signaled by a new note-on event following the rest period, which also marks the beginning of a new melodic surface. For each melodic surface, the following information is recorded:

- Beginning of the surface
- End of the surface
- Duration of the surface
- The melodic surface amplitudes
- The timing of the melodic surface
- The energy of the melodic surface

In addition to the melodic surface, the rest information is also preserved. Melucci and Orio ignored the rest phases since rest information is of no use in their proposed application. However, in this thesis, rests information are used to determine when to start or stop the robot's motion, and hence are preserved. The rest information consists of:

- Beginning of the rest period
- End of rest period
- Duration of rest period

The melodic surface is created by evaluating each beat in the surface. For each beat, the value of the prominent note in the beat is taken as the surface amplitude. In the case that there are multiple notes simultaneously occur in a beat, the note with the highest note-on velocity value is selected as the surface amplitude. The time of the note-on events of the selected notes is recorded to be the timing of the melodic surface. For the case of multiple simultaneous note events, a better approach may be to analyze the possible *chords* created by such event. Chords create various harmonics between the notes, and often used as the basic structure in musical notation. Although currently not covered in this thesis, further study can be done to explore the use of chords with the proposed system here.

So overall, the following information from the above event listing can be extracted:

- Timing:
  - The start time of a note (t^i_{on})
  - The duration of a note (t^i_{len})
  - The end time of a note ($t^i_{off} = t^i_{on} + t^i_{len}$)
  - The start time of a rest (t^{rest}_{on})
  - The duration of a rest (t^{rest}_{len})
  - The end of a rest ($t^{rest}_{off} = t^{rest}_{on} + t^{rest}_{len}$)
- Dynamics:
  - How loud a note is being played (i.e. *piano* vs. *forte*) ($v^i_{note-on}(t)$)

  - How loud a note at the point of release ($v^{i}_{note-off}(t)$)
  - The amplitude of a note over time (i.e. *crescendo* vs. *decrescendo*) ($A^{i}(t)$)
  - Energy per beat (E)
- Melody:
  - Melodic surfaces
  - How many notes are being played at a given beat (density) ($D(b)$)

In addition, other information about the song is also listed explicitly in the file:
- Timing (number of beats per bar)
- Tempo (number of beats per minute)

The sound information is stored in a Python *dictionary* data structure where each entry (i.e. event) is indexed by the ticks for each beat. Therefore, the sound information is now arranged in beats (i.e. events per beat). The application of the above information to motion control is explained in Section 4.3.3.

The sound information is stored in a structure, indexed by the time unit of each beat:

Tick	beat#	prev_note	target_note	delta	sustain	rest_count	energy	Density

Figure 4.7 Beat information

A second structure records the occurrence of melodic surface and rests. For the melodic surface:

Phrase	Phrase_start	Phrase_end	Phrase_length	Contour_amplitudes	Contour_time	Contour_energy	Notes

Figure 4.8 Melodic surface information

And for the rests:

Phrase	Rest_start	Rest_end	Rest_duration	Rest_energy

Figure 4.9 Rest information

The following is the description of each field:
- In the beat structure:
  - Tick: the time (in units of ticks) of the beginning of the beat
  - Beat#: the beat number
  - Prev_note: the selected note in the previous beat
  - Target_note: the selected note for the current beat
  - Delta: the difference between the previous note and the target note
  - Sustain: True if there is a note being sustained in the beat, False otherwise
  - Rest_count: the number of beats a rest happens before the current beat
  - Energy: the energy (E) value in the current beat
  - Density: the number of note events in the current beat

- In the melodic phrase structure:
  - Phrase: Melodic phrase/Rest flag - True if this structure is a melodic phrase, False if it is a rest.
  - Phrase_start: time of the first note-on event in the phrase
  - Phrase_end: time of the last note-off event in the phrase (i.e. start of rest segment)
  - Phrase_length: length of the melodic phrase
  - Contour_amplitudes: the amplitudes of the melodic phrase surface
  - Contour_timing: the time positions of the contour amplitudes
  - Contour_energy: the energy of each contour points
    - If there is only one note: the energy is the note-on velocity of the note
    - If there are more than one notes: $E = \frac{1}{N}\sum_{i=0}^{N}(v^{i}_{note-on})^{2}$, where N = number of simultaneous notes, $v^{i}_{note\text{-}on}$ = note-on velocity of note i.
  - Notes: list of the notes in the phrase
- In the rest structure:
  - Phrase: same as in the melodic phrase structure
  - Rest_start: time of the start of the rest segment
  - Rest_end: time of the end of the rest segment (i.e. beginning of new melodic phrase)
  - Rest_duration: duration of the rest segment, in number of beats.
  - Rest_energy: the note-off velocity at the end of the last melodic phrase

In the melodic surface example in Figure 4.10, only five pieces of information are shown: two melodic surface information, and three rest information. The first line shows a rest information, indicated by the value of `'phrase'` equals `False`. The rest segment starts at tick 768 (`'rest_start': 768`), ends at tick 1536 (`'rest_end': 1536`), the duration is two beats (`'rest_duration': 2`), and the energy is zero (`'rest_energy': 0`). The rest energy is equal zero because this segment is at the beginning of the song, and the segment is not preceded by any note, hence no note-off velocity information.

The next line is the first melodic surface information. This is indicated by `'phrase': True`. Similar with the rest information above it, the first thing to notice is the start of the melodic surface at tick 1152 ('phrase_start': 1152), it ends at tick 2292 ('phrase_end': 2292), and the duration of the surface is 2292 – 1152 = 1140 ticks ('phrase_duration': 1140). Next, the melodic surface is defined by three data points: `'contour_amplitudes': [92, 55, 92]`. The location in time for the surface data points are defined in: `'contour_time': [1532, 1540, 2196]`, and the energy for the surface data points are: `'contour_energy': [10010, 8250, 8954]`. The final bit of information is the `'notes'` attribute. In this particular melodic surface, there is a total of ten notes involved. The ten notes do not happen at the same time, of course. Instead, they happen in three beats that constitutes the melodic surface. In the first beat, there are two notes, three notes in the second beat, and five notes in the third beat. Notice the value `'notes'` attribute is a three-dimensional list as shown in Figure 4.11.

```
[{'rest_end': 1536, 'phrase': False, 'rest_duration': 2, 'rest_energy': 0, 'rest_start': 768},

{'contour_energy': [10010, 8250, 8954], 'phrase_length': 1140, 'contour_amplitudes': [92, 55, 92],
'phrase': True, 'notes': [[[1532, 568, 92, 110, 64], [1534, 568, 80, 89, 64]], [[1540, 558, 56, 85,
64], [1542, 550, 55, 98, 64], [1542, 554, 60, 89, 64]], [[2196, 96, 92, 110, 64], [2196, 44, 80, 89,
64], [2198, 40, 56, 89, 64], [2200, 26, 60, 85, 64], [2204, 22, 55, 98, 64]]], 'phrase_end': 2292,
'contour_time': [1532, 1540, 2196], 'phrase_start': 1152},

{'rest_end': 2688, 'phrase': False, 'rest_duration': 1, 'rest_energy': 8954, 'rest_start': 2304},

{'contour_energy': [9568, 6525, 7929, 7779], 'phrase_length': 1856, 'contour_amplitudes': [53, 89, 52,
80], 'phrase': True, 'notes': [[[2970, 120, 83, 110, 64], [2972, 90, 89, 98, 64], [2972, 84, 53, 93,
64], [2974, 80, 59, 89, 64]], [[3076, 64, 84, 51, 64], [3092, 136, 89, 73, 64], [3168, 88, 84, 76,
64], [3242, 64, 83, 98, 64], [3346, 32, 77, 79, 64], [3446, 102, 82, 98, 64]], [[3530, 142, 83, 98,
64], [3732, 428, 82, 93, 64], [3736, 410, 58, 79, 64], [3740, 394, 52, 85, 64]], [[4230, 254, 80, 98,
64], [4502, 42, 77, 98, 64], [4506, 32, 56, 82, 64], [4508, 22, 60, 79, 64], [4512, 18, 55, 82, 64]]],
'phrase_end': 4544, 'contour_time': [2970, 3076, 3530, 4230], 'phrase_start': 2688},

{'rest_end': 4992, 'phrase': False, 'rest_duration': 1, 'rest_energy': 7779, 'rest_start': 4608},
... ]
```

Figure 4.10 Melodic surface and rest information

The 'notes' attribute is only used as an indicator to keep track of the notes that contribute to the melodic surface. Observing the following melodic surface information, it can be seen that the length of the melodic surface corresponds to the number of beats in the surface, and not to the number of notes. This is because simultaneous note events in one note is consolidated into one value per beat. Although some note information is lost by the consolidation, the approach captures the timing information in the 'contour_time' attribute with enough detail. Because of the time it takes to send the data to the ASC16, and the time for ASC16 to process the commands, the timing information cannot be too fine (e.g. to the miliseconds), but should still be enough to capture musical timings such as eighth note (half beat), or sixteenth note (quarter beat). This issue is discussed further in next chapter on the experiments. The subsequent rest and melodic surface information can be read in the same manner as the examples above.

```
'notes': [
(first beat)
[[1532, 568, 92, 110, 64],
[1534, 568, 80, 89, 64]],

(second beat)
[[1540, 558, 56, 85, 64],
 [1542, 550, 55, 98, 64],
 [1542, 554, 60, 89, 64]],

(third beat)
[[2196, 96, 92, 110, 64],
 [2196, 44, 80, 89, 64],
 [2198, 40, 56, 89, 64],
 [2200, 26, 60, 85, 64],
  [2204, 22, 55, 98, 64]]
]
```

Figure 4.11 Contents of the 'notes' attribute of the first melodic surface from the example in Figure 4.10

4.3.2 Scenario Creation

First of all, let's define the term 'Scenario'. In the context of this thesis, a 'Scenario' is defined as;

"A sequence of gestures arranged in a particular order of execution"

The purpose of the Scenario is to facilitate the Robot Theatre project. A Scenario is analogous to a directed, planned scene such as the one in a theatrical play, which is a particular order of actions of the actor/s to convey a story. Note that a Scenario can also consists only of one gesture or motion. The sound information will be used to control the execution of the Scenario and each gesture in the Scenario. For example: timing (start/end) of each gesture. A gesture in the Scenario may be executed in one musical phrase, one beat, half of a phrase, and so forth depending on the dynamics information, for instance. Again, more details on how this is done is explained in Section 4.3.3.

A user-interface (UI) is provided to users to create and arrange a Scenario. The Scenario creation process is straightforward:

1. Open up the Scenario Editor UI.
2. The UI automatically loads all the available gestures from the Gesture Library from a default location (if the user created another gesture library, he/she can change the location of the desired Gesture Library).
3. Select a gesture from the Gesture Library.
4. Click the '+' (add) button to add the selected gesture to the Scenario list.
5. The user is provided with some options:
    1. Repeat the gesture in the list
    2. Invert the motion sequence of the gesture
    3. Re-arrange the sequence of gestures in the Scenario list
6. Once the user is satisfied with the Scenario, save the Scenario

4.3.3 Planning Scenario Execution (Scenario Mode)

At the heart of the whole system, is the planning-Scenario-execution process. Here, all the information extracted from the sound input is used to control the timing and execution of the three options: a direct translation from song information to motion, a Scenario without re-arranging the sequence of gestures in the Scenario, or an automatic gesture selection from the Gesture Library with corresponding transformations from the sound information. The user has control over which mode the robot is in at all times.

When a Scenario is provided as input, the sound information will be used as follows:

1. The execution of a gesture begins at the start of a *bar/measure*.
2. The execution of the preparation phase of a gesture is determined by the timing of the first note-on event in the bar/measure, or the beginning of a melodic phrase.
3. The range of motion of any phase of the gesture may be modified depending on:
    1. The number of notes being played at the same time (Density) at the start of the phase execution.
    2. The combined amplitudes of the note(s) being played at the start of the phase execution

3. The melody 'envelope'

4. The acceleration of the motion may be modified by the combined note-on velocity of the note(s) being played at the start of the phase execution
5. The speed of the motion is determined by the sustain duration of the note(s)
6. In the event of 'rest', the motion is stopped and the recovery phase (return to home position) is executed at the *lowest speed possible*.
7. In the event of 'sustain', which is the duration of a note being sounded, the stroke is to be executed starting on the time of the note-on, and finish at the end of the sustain (at release).

In order to 'translate' the sound information (e.g. timing, pitch, velocity) into motion information (i.e. acceleration, speed, range of motion), some conventions must be explained. Part of the convention is defined by the MIDI protocol, others are from the ASC16 servo controller board, and a few others specific to the system presented here. The source of convention will be mentioned when encountered in the explanation below.

4.3.3.1 Timing

The tempo/beat information is used to determine the execution of a gesture, or a part of a gesture. Therefore, the start of a gesture is always aligned with a beat. Specifically, if at all possible, the start of a gesture is always at the start of a bar (or *measure*). A measure consists of a number of beats, determined by the timing of the song or *time signature*. The time signature is written in the form of two numbers, one above the other, or sometimes also written like a fraction: 2/4, ¾, 4/4, 5/8, and so on. For example: timing signature 4/4. The 'nominator' value indicates that there are four note units per measure, and the 'denominator' value indicates that the note unit of the 'nominator' is quarter note. A timing signature of 5/8 indicates that there are five eighth notes per measure. A quarter note is normally played for one-beat duration.

Although the start of a gesture is aligned to the first beat of a measure, the time of execution of the first movement is determined by the time of the *note on* in that first beat. For example: a note may happen right on the beat (the note is a quarter note), or halfway into the beat (the note is an eighth note). Thus, the gesture will start *within* the beat, but does not necessarily always with the beat, but to the *note-on* in the beat. The duration of each beat is given by:

$$t_{beat} = \frac{60}{(beats\ per\ minute)}\ seconds \tag{4.14}$$

Each beat is represented in *ticks* as 384 ticks/beat per the convention used in the MIDI protocol. Suppose a song is clocked at 100 beats per minute (bpm). Thus, the duration of each beat is:

$$t_{beat} = 60/100 = 0.6\ seconds/beat$$

This is the first use of the time information: to determine the start/end of a gesture. Let's suppose the first event recorded from the song occurs at tick=1920 or beat 5. Therefore, the execution of the first gesture happens after 0.6 * 5 = 3 seconds.

There are two ways to time the gesture execution: tell the ASC16 to wait before executing the next move command, or tell the program to wait before sending a move command to ASC16. The ASC16 has a 'wait' value range [0,255], where each unit correspond to 10mS. Therefore, the the range of wait period of the ASC16 is 0 to 2550mS. On the other hand, using the `sleep` function in the program, the delay accuracy can be within 1mS. Therefore, the timing is controlled by the program, instead of by ASC16.

4.3.3.2 Velocity

Next, the duration of the event (i.e. the 'len' parameter) determines the velocity of the motion (i.e. *speed* parameter of ASC16). For example: from the event list above, at tick = 1532, the duration of the note (len) is 568, or roughly 1.5 beats, which is: 1.5 * 0.6 seconds = 0.9 seconds or 900 milliseconds. Suppose the first motion is to move the servo from position 60 to 90 degrees, then the offset motion (30 degrees) must be achieved/finished within 900 milliseconds. In other words, the top velocity of the servo will be *approximately* 33 degrees/sec. The velocity of the servo for a particular motion event is then calculated as:

$$v = \frac{\Delta p}{t_{note}} = \frac{(p_{i+1} - p_i)}{t_{note}} \tag{4.15}$$

Where v is the velocity for the servo, p_i and p_{i+1} are current position and target (next) position of the servo, respectively, and t_{note} is the duration of the note (len). The ASC16 is designed with speed value of range [0,255]. Each speed unit equals to 50 motion counts/sec, 1/160 revolution/sec, 2.25 degrees/sec, or 3/8 rotation per minute (RPM). Therefore, if the desired velocity is 33 degrees/sec like in the example above, the speed value for the ASC16 v_{ASC16} is:

$$v_{ASC16} = int(v/2.25) = int(33/2.25) = int(14.67) = 14 \quad \text{or about 31.5 degrees/sec.}$$

Notice that the rounded down integer value is chosen. If the velocity value is first rounded up, the above example gives the speed for the ASC16 as 15, or 33.75 degrees/sec. The difference of the resulting motion between the two options are too subtle to be distinguished by the naked eye. Since there are no strong reason to choose one of the options over the other, the former is chosen. More details on the speed settings of ASC16 is described in the ASC16 manual [ref to ASC16 manual (*appendix maybe?*)]. The above velocity equation is 'approximate' because it did not take into account the *acceleration* of the servo; which is the time required for the servo to reach that velocity.

4.3.3.3 Acceleration

The value for acceleration is determined by the value of the *note-on velocity* parameter, which

corresponds to how much force was applied to the key for the note. The note-on velocity is associated with the loudness of note being played. In MIDI, the value of note-on velocity ranges from 0 to 127, from the lowest to highest, respectively. The acceleration is then determined as the product of the maximum acceleration value with the normalized value of the note-on velocity.

$$a = v_{note-on} * a_{max} \tag{4.16}$$

Where a is the final acceleration value, $v_{note\text{-}on}$ is the normalized note-on velocity value [0,1], and a_{max} is the maximum possible acceleration value. For ASC16, the range for acceleration is [1,255]. The ASC16 manual suggests to set the maximum acceleration to 100 for a standard servo such as the ones used in the test robot [ref ASC16 manual]. The tests show that the difference in the movements between acceleration=50 through 100 is indiscernible to the naked eye, at least for the type of servo used. For values [1,50] there are more noticeable difference in the acceleration. Thus, a little modification to the acceleration formula (in Eq. 4.16):

$$a = (v_{note-on} * 50) + 1 \tag{4.17}$$

4.3.3.4 Range of Motion

The range of motion is affected by the amplitude level of the melodic surface (`'contour_amplitudes'` attribute). Notice that the content of `'contour_amplitudes'` is a list of selected notes, which corresponds to MIDI note codes. There are two ways melodic surface amplitudes applied to range of motion: directly to position values, and relative position changes. Amplitude information is directly converted to position values when the robot is in Free mode, where the movement of the robot is not dictated by a Scenario. In this mode, the robot always starts at a HOME position, which is mapped to the middle C or C4 note. The movement that follows is calculated as the relative note difference of the next note from the previous note. For example: the range of motion of a degree of freedom is 4000 (i.e. in ASC16 convention). Suppose the first note that occurs is A4. The distance of the note A4 from C4 is +9 (see appendix … for the scale). The displacement is then calculated as the product of the pitch distance with the energy E at the moment (i.e. beat).

$$d_{pos} = \Delta note * E$$

Only the integer part of d_{pos} is converted into position value for ASC16. Note that if the note changes from a higher note to a lower note (e.g. C4 to C3), the $\Delta note$ will be negative, thus creating a negative displacement d_{pos}. In other words, the change in pitch dictates which direction the movement will go. For example: if $\Delta note > 0$ then the joint will turn left, otherwise if $\Delta note < 0$, the joint turn right. Naturally, $\Delta note = 0$ cause no displacements.

In the Scenario mode, the same displacement d_{pos} is used, but scaled by a factor *s,* where $s < 1.0$. The scaling is necessary to make sure that the original gesture is preserved, and still recognizable.

4.3.4 Free Mode

In Free Mode, the music data is directly converted into motion data. Specifically, the melodic surface amplitude values (`'contour_amplitudes'`), which are MIDI note numbers, are multiplied with the 'energy' of the melodic surface (`'contour_energy'`), and directly translated into joint angle data. The timing of the motion is dictated by the melodic surface time (`'contour_time'`).

To convert to joint angle data, the product of the amplitude and energy, P is a normalized value [0,1]. P is multiplied with the user-defined range of motion $R = [R_{high}, R_{low}]$. The R_{high} and R_{low} are the upper and lower limit of the range of motion, respectively. The limits in R may depend on the configuration of the robot itself, as desired by the user, or interpreted from the melodic information. Since the timing of the melodic surface already corresponds one-to-one with the values of the melodic surface amplitude, no additional process is necessary to determine the start and end of the produced motion.

Taking the data from the example in Figure 4.10, Figure 4.12 shows the resulting motion data. As previously mentioned in section 4.2.2.2, every gesture starts at the HOME position. Since in the Free Mode the motion data is directly derived from the notes in the music information, a 'HOME note' is defined to be 60 or note C4 in the musical scale. Notice that during rests, no movement is made and the last position is maintained.

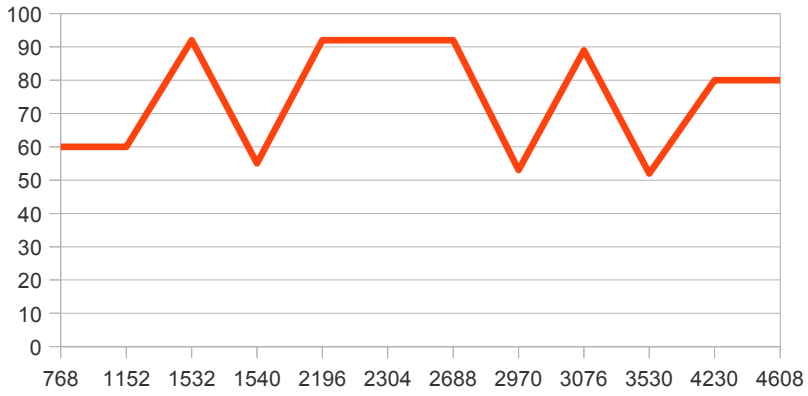

Figure 4.12 Motion data from melodic surface amplitude from example in Figure 4.10

4.3.6 Gesture Mode

Gesture Mode works similar to Scenario Mode, where the music information is used to modify how selected gestures are being executed. The main difference, in Gesture Mode the sequence of gestures is selected based on the musical events, and not pre-selected as a Scenario by a user. In other words, the system selects its own sequence of gestures to be executed.

Similar to the Scenario Mode, the execution of a gesture is associated with a melodic surface. Therefore, in Gesture Mode, the execution of the gesture is influenced by the qualities in the melodic surface (timing, dynamics), and the gesture to be executed is selected based on the characteristic of the melodic surface. For example: … (to be continued)

4.3.7 Putting Everything Together...

The system has three modes of operation: Free mode, Scenario mode, and Gestural mode. In the Free mode, the sound information is converted directly into position values. In Scenario mode, the sound information is used to arrange the timing execution of the gestures in the Scenario, and apply some modifications. In Gestural mode, the system selects gestures based on events in the sound information – essentially, creating its own Scenario, and applying modifications to the selected gestures as in Scenario mode.

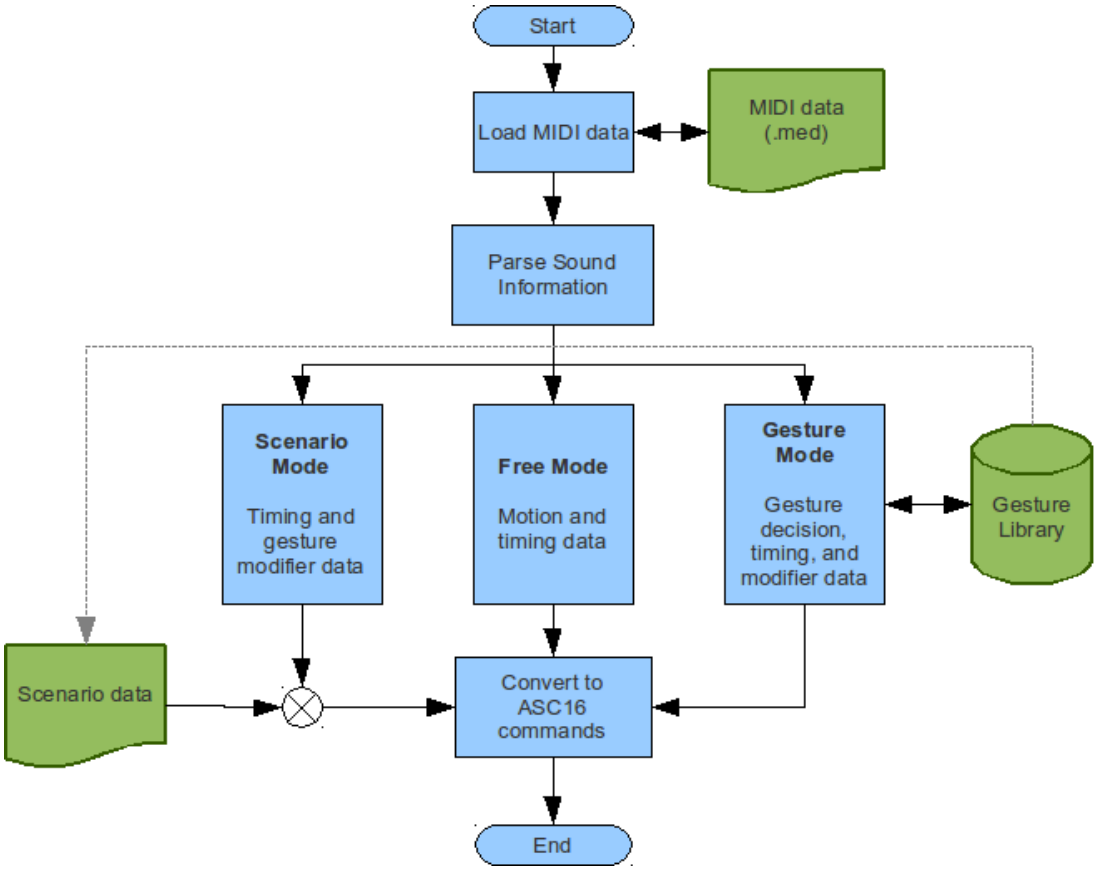

Figure 4.13 The complete system

Chapter 5 – Experiments

(to be written)

Chapter 6 – Conclusions and Future Works

6.1. Conclusions

A method is presented to map musical qualities to the control and synthesis of expressive motions for robot actors. By using music as input, the structure of melodic surfaces can be used to time the execution of the sequence of gestures in a Scenario. Also, the contour of the melodic surface can be used to modulate the gesture motion data to give the motion similar dynamics (velocity, acceleration, range of motion) as the melody, or to synthesize motion data directly from the surface amplitudes. The result is a more dynamic-looking motion which dynamics are not random, but defined by man-made structured signals which are known to have affective effects (music). The kind of affect varies depending of the kind of music used as input, and how observers interpret the motion dynamics.

The approach presented here was able to show that music data can be used as characteristic function that can be applied to different motions to give similar expressions as suggested by Unuma. Whereas events in speech information has been used to execute beat gestures, the rich melodic information is shown to allow selection of other types of gestures besides beat, such as deictic and iconic gestures. However, there is not one single rule that is correct for the mapping between the melodic events to the gesture selection, only guidelines. For example: a melody which pitch trend is increasing, deictic gestures that moves upwards such as pointing up is appropriate. A melody that has repeating intervals is suited for looping gestures such as a circle, or beat gestures.

Because of the restrictions in hardware used, such as robot configuration, servo specifications, and communication delays between the servo controller board and PC, some adjustments must be made. Although ASC16 supports acceleration values [1,255], in the main program the acceleration value is limited to [1,50], simply because values above 50 do not give noticeable difference than 50 in the acceleration of the movements. Such adjustments must be examined for different robots and/or hardware used.

6.2. Future Works

There are several future studies emerged from this thesis. First, apparently there is still no reliable method to extract rich melodic information from sampled music signals. This is the main reason MIDI was used as input in this thesis. Sampled music signals are difficult to parse since the sound of different instruments and vocals are often already intermixed into one signal. Even more difficult is to 'listen' to the music using a microphone in real-time because of the accompanying noise. There exists several different methods, each to extract specific type of information from the music data, such as beats, or tempo. But often the methods only works well for certain types of music and not for other types. Better methods need to be investigated to enable extracting rich melodic information while listening to the music sound in real-time.

Second, there are several MIDI control events that are not being evaluated in the proposed method. The contributions of those events as additional motion control parameter may be explored. When fully explored, the MIDI Show Control (MSC) protocol may be used to control the whole Robot Theatre.

Third, simultaneous note events are consolidated into one event in the proposed method. Most likely, the simultaneous note events represents a musical chord. Musical chords have interesting properties such as creating harmonics in the sound frequency. A study can be done to analyze simultaneous note events into chords, and then analyze the chords information to be used as a control parameter for motion data.