1-1-2011

Universal Event and Motion Editor for Robots' Theatre

Aditya Bhutada
Portland State University

Universal Event and Motion Editor for Robots' Theatre

by

Aditya Bhutada

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Electrical and Computer Engineering

Thesis Committee:
Marek Perkowski, Chair
Douglas Hall
Roy Kravitz

Portland State University

Abstract

Most of work on motion of mobile robots is to generate plans for avoiding obstacles or perform some meaningful and useful actions.

In modern robot theatres and entertainment robots the motions of the robot are scripted and thus the performance or behavior of the robot is always the same.

In this work we want to propose a new approach to robot motion generation. We want our robot to behave more like real people. People do not move in mechanical way like robots. When a human is supposed to execute some motion, these motions are similar to one another but always slightly or not so slightly different. We want to reproduce this property based on the introduced by us new concept of probabilistic regular expression, a method to describe sets of interrelated similar actions instead of single actions. Our goal is not only to create motions for humanoid robots that will look more naturally and less mechanically, but also to program robots that will combine basic movements from certain library in many different and partially random ways. While the basic motions were created ahead of time, their combinations are specified in our new language. Although now our method is only for motions and does not take inputs from sensors into account, in future the language can be extended to input/output sequences, thus the robot will be able to adapt the motion in different ways, to some sets of sequences of input stimuli. The inputs will come from sensors, possibly attached to limbs of controlling humans from whom the patterns of motion will be acquired.

Dedication

This work is dedicated to contribute a little towards a big dream of Dr Marek Perkowski.

Acknowledgements

I must appreciate Dr David Bullock who supported me financially for my graduation.

I would also like to thank to Electrical & Computer Engineering department in PSU for availing me all resources timely required for my project.

During the implementation phase of the project, I had great help from my friends Clint Lieser, Charlie, Amit Bhat, Mandar Patil, Padmashri Gargesa, Sreeram. The guidance from Ernest Burghardt, Arunashri Swapna, Paul Kovitz while interning at Logitech was also commendable. I also thank to Vincent Vuarnoz for his moral support.

Last but not least to mention is my family. Thanks to every single element in my family for sending me to US to complete my dream of getting Masters.

Table of Contents

List of Tables

List of Figures

Chapter 1 Introduction to Robot Theatre and Motion

Goals of this thesis will be explained briefly in the introduction:

1. Interface to iSOBOT
2. Motion edition for iSOBOT.
3. Language of generalized regular expressions.
4. Genetic algorithm for motions.
5. Editing motions, modifying motions and evolving motions in unified way.
6. Control of motors, robots, lights and whole stage with robots. Ideas.
7. Practical experiments with the editor and robots on stage.

The main idea of this M.S thesis is that the current digital, computer, robotics, and interfacing technologies allow integrating many areas of activity together, to create complex interrelated systems of events, measurements, motions and their processing involving various correlations. These possibilities will be used in soon in all kinds of home automation systems. You will be able to call from highway to your computer to set the automatic kitchen to cook dinner for you and the home robot to close windows when the storm is coming. My idea is to apply current robotics technologies in robot theatre.

From the point of view of used technologies, this thesis is a practical application of embedded systems and software/hardware co-design. I use an Arduino board to transmit infrared signal to a robot to be able to control wirelessly. Although not used as part of

final project, in the research work, I had designed hardware for speech synthesis chip interface.

Now a day, robots do a lot of different tasks in many fields; and this number of jobs entrusted to robots is growing steadily. Application point of view, their types are: industrial robots, domestic robots, medical robots, service robots, military robots, entertainment robots, and exploration robots. We are interested in entertainment humanoid biped robots. Figure 1-1 shows many of such interesting entertainment robots, among which I chose iSobot for the practical demo. The price tag on them tells why!

With 15 ~ 25 degrees of freedom, most of these robots are capable of executing a wide range of movements (walking, sitting, standing up, dancing, avoiding obstacles, kicking, seizing objects etc.). Some of them come with their own software editor. This thesis introduces a concept to make a universal editor capable of editing motions for all such beautiful robots.

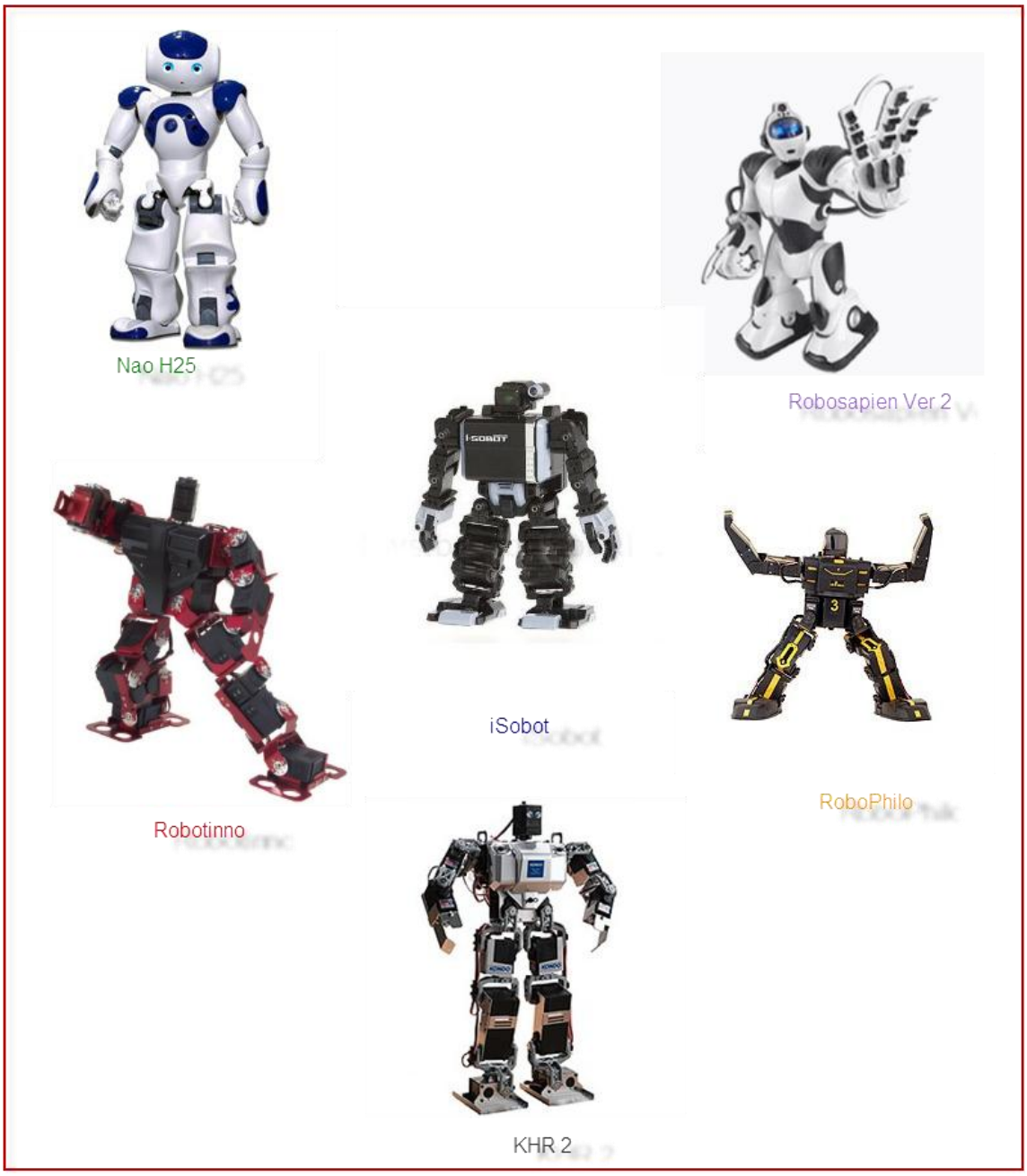


Figure 1:1 Humanoid Robots

This thesis is formatted in chapters as below.

Chapter 2 introduces a formal model of motion and behavior for a complete robot theater. This chapter is the main theory part of chosen topic. It talks about need of such editor, what was motivation behind it, and include some background research work on it.

In Chapter 3 discuss about the design of hardware to control the iSobot remotely. The first half section of the chapter details about hardware design & the second half section describe basic software designed in Python that works with the hardware and control iSobot ultimately.

In Chapter 4 the thesis will present…the speech synthesis technology. The first half section describes how a hardware part was built to interface a 'Sound Synthesizer' chip to be controlled serially. The second half part of the chapter talks about better alternative and software implementation of the same method.

Chapter 5 is about Music & Light controller. This chapter enlightens some ideas about controlling light for the robot theater. Based on those, what was chosen and implemented. The chapter also includes more details about MIDI & its interface.

Chapter 6 is complete description of graphical user interface right from the beginning how it was built in Microsoft Visual Studio 2010.

Chapter 7 is a manual teaching how to use the software, and setup the hardware. It also mentions total hardware cost, notes & instructions what to do, and what one must avoid doing.

Chapter 8 has some comparisons with other similar editors, pros and cons and feature development ideas.

Chapter 9 concludes the Results.

Literature section will present all positions and web pages that were used in the process of writing this thesis and are related to editors and robot theatre, followed by Appendix.

Chapter 2 Systematic Design of a Probabilistic Motion Generator for biped robot motions based on Multiple-Valued Logic and Event Expressions

Introduction

In this chapter we introduce a formal model of motion and behavior for a complete robot theatre. Formally in our model a behavior is certain input/output mapping between sequences of input symbols and output symbols. Input symbols are formed based on sensors' and cameras' readings, output symbols represent elementary robot/theatre behaviors such as humanoid robot's arm-up or a rotation of the stage.

This input/output mapping describes an intelligent motion controller with arbitrary perceptions on inputs and motions on outputs. The perceptions can be very complex, like recognizing an old man who is smiling. The output actions can be also very complex, like walking toward recognized target in a dancing way [37, 39, 40].

Our model of robot architecture is the concatenation of three types of machines:

- perception machines, which transform vectors of basic features (obtained from sensors and cameras) to certain symbolic recognized objects,
- behavior machines, which use stored state information and patterns of objects to transform robot's perceptions and memories to robot's actions,

- motion machines, which describe in general what the audience members perceive in the theatre – lights, sounds, music, texts, motions of robots and other artifacts, motions of lights and other sources of visual effects.

In this chapter we are interested mostly, but not inclusively, in the motion machine design. Motion in our understanding in this chapter is a very broad concept that includes (partially) coordinated motion of many robots, automated furniture, sounds, music, lights, smoke machines, curtain rising and dropping, etc all theatrical motions and effects that are synchronized and automated. This synchronization may be deterministic or probabilistic, partial or complete.

We propose here the so-called Theory of Event Expressions as a tool to design motions directly from symbols. This theory has the following properties:

(1) It is general enough to allow arbitrary motion to be symbolically described on a high level. This is the level used by artists, stage directors, writers – they do not resort to full details of programming the velocities and angles of degrees of freedom of a humanoid arm movement – they would just give the following direction – "raise the right hand energetically up".

(2) The theory is also detailed enough to allow the robot programmer, or the robot itself, to create a precise generated behavior with the most basic, atomic details of DOF angles, velocities, or light-dimming parameters.

(3) Our main concept of the presented theory is that the motion is a sequence of symbols, each symbol corresponding to an elementary action, such as shaking a head for answering "yes".

(4) The elementary action on this level can be further decomposed to angles, velocities and accelerations of every servo in every degree of freedom of the theatrical robotic system of motors, servos, lights and valves, or to other minuscule action/event.

(5) We will define some atomic, basic – so-called - primitive motions. Each will have a separate symbol.

(6) Combining primitive motions creates the complex motions. We will use some special operators for combining these motions.

(7) The complex motions are based on some general structures of basic motions and operators on them. These operators may be deterministic or use some (partially programmable) randomness.

Although our theory of event expressions is general, we illustrate a particular application: humanoid biped robots KHR-1 and iSOBOT with speech, sounds, music and lights in Portland Cyber Theatre. Literature presents many interesting small robots that can be used in a theatre. [45, 46, 47, 48, 49]

The created here theory of event expressions can be applied to simulated agents in movie animation and computer games and to physical robots with material bodies. It is a general theory to generate generalized events/motions.

We are interested here only in physical robots, not software robots or graphical animations of robots. What are the factors and properties of motions that are important to generation of symbolic motions, especially in dance and pantomime? The research issues are:

(1) What should be the primitive, basic or atomic motions? How to describe them? How to select them from a large library in our motion editor?

(2) What should be the operations on motions that combine primitive motions to complex motions?

(3) How symbolic motions are reflected/ realized as physical motions of real robots?

How to create existing and new symbolic motions automatically, using some kind of knowledge-based algebraic software editor.

This chapter presents an approach used in a graphic editor for humanoid biped robots that uses concepts of Finite State Machines, Event Expressions, MV logic, Probabilistic Motion Generators and algebra of motions. A subset of this theory was programmed in my actual editor developed for the applications of Portland Cyber Theatre – see chapter 6 [Unified Editor].

Working on developing the theory, I found that there are two semantics for my theory:

(1) **set-theoretical**, in which we only create sets of motions from basic motions (basic motions cannot be created, they remain the same),

(2) **hybrid-automata based**, in which, additionally, new basic motions are created by some calculations in a hybrid automaton.

Only the first theory is programmed in the final motion editor presented in the thesis, as we start from a finite library of motions. But I present in this chapter all my attempts at creating the most general theory, and initial ideas of its realization. These considerations may help the next students to design a more advanced version of my editor.

Symbolic Motions and Notations for them

Dr. Perkowski and RAS built a Portland Cyber Theatre at PSU. This will be a theatre of two types of robots:

- Small humanoid bipeds, such as Robosapiens V2, KHR-1, Bioloid, Futaba and others. Many technical fairs and shows demonstrate behaviors of such robots, dances and performances. These are all walking robots of small size.

- Medium size humanoid wheeled robots, such as Schroedinger Cat, Einstein, Niels Bohr or Marie Curie. These are robots on wheels or tracks, but they have heads or partial human/animal-like bodies with arms and hands to manipulate and to gesticulate. Their faces are able to demonstrate certain simple emotions.

Both types of robots have sensors and actuators. Both can be subject to the methodology developed in this thesis and presented in this chapter. Let us first analyze some motives.

Reasons of high interest of public in humanoid robots

1. Robots have physical bodies, so non-experts can see and appreciate practical usefulness of theories that stand behind their construction. These robots are quite different physically than industrial or service robots. The audience of this type of research is therefore numerously larger than audiences at other technical fairs. There is simply more people interested in entertainment than technology and fabrication.

2. These robots relate to the communication aspects between robots and humans. But on an even deeper level, they let us analyze general principles of sign-based communication.

3. They are a truly new product on a market and thus differ from laptops, cameras or iPhones which are only improvements to products known for many years. Market studies show that most customers would like to purchase the robots for their houses when they will be less expensive.

4. There is some implicit emotional value in motion which fascinates humans for thousands years. It allows humans to identify themselves with other persons. With emotional states of other humans. But also with animals or even with nature and life in general. Therefore motion is a very important aspect of every theatre, and robot theatre in particular.

The motions of modern individual humanoid robots are unfortunately still quite limited, hence comes the idea of building "theatres" in which many robots execute some synchronized or spontaneous actions. When we have a theatre, we should also have lights, music, sounds, and speech. More elements can be added that can be controlled in a very similar way to the controls of the motions of the individual robot.

Analysis of requirements for robot motions in the theatre:

1. How well the robots can imitate humans? The little bipeds have only about 17 – 30 degrees of freedom, while a human has about 300 degrees of freedom. So their repertoire of motion is very limited. However they can realistically imitate many human gestures and behaviors. We can assume thus, that a limited, finite set of basic motions can be created to become the base of our theory.

2. Programming these robots with existing commercial editors is quite time-consuming and the motions are created by engineers or artists with a lot of effort.

The programmer does not have much initiative to develop really complex motions for the robots, as it would be very time consuming.

3. A more complex way of creating motions, which are specified in the scientific motion-generating tools is not easy, because these tools have no good interfaces and are based on kinematics or inverse kinematics theories. These more advances mechanics-based tools should be in future incorporated to the editors of the type described in this theory. These advanced editors will be able to create completely new motions in addition to combining existing motions to sequences. And the motions will use kinematics and inverse kinematics for smoothness, biological realism or advanced artistic effects.

4. The problem of creating new interesting motions within constraints of the mechanical motion-generating system and environmental constraints (obstacles, walls) will become even more difficult, when future humanoid robots will have very many degrees of freedom and will operate in less-structured theatrical stages, which will be similar to normal environments of humans. In further future, even more complicated environments, such as super-movie like battle-fields (star wars) will be created for robot theatres. It is still an open question how many DOF will be sufficient for a natural-size realistic humanoid actor in all kinds of "robot theatres". Although such robots do not exist yet, we can predict that they will have more than hundred degrees of freedom and will be based on very

complicated kinematic design, mimicking animals or purely fantastic, not constrained by anything that currently exist.

5. New theories have been developed, how to create motions. Research continues on designing complete languages to describe all motions; Labanotation [33,39], DAP (Disney Animation Principles) and CRL (Common Robot Language) [32] are only some examples. Every year brings new papers and systems related to robot languages and how to program various behaviors in animated movies or in entertainment robots. In future, these theories should be somehow related to the theory presented here. At least, the editor should be able to incorporate some elements of these theories.

6. All above arguments taken together testify that the new research area of entertainment robotics becomes a valid and important topic of both scientific and engineering research. However, it is still not treated as a full academic area yet. Perhaps one reason is that there is so far very little fundamental science behind it, with the exception of biologically motivated motion generators for animal robots such as hexapods, dogs, snakes or fishes.

7. Another reason for not much research in motion generation for theatrical robots is that generated motion often represents an emotional state of an actor or a transition between emotional states. As the emotional mechanisms of

communication and behavior are just being elucidated, no deeper research was possible so far. Only ad hoc solutions are found by robot builders and animators. Papers that attempt to create such a theory just start to arrive recently. These theories will with no doubt influence future theories of motion generation based on principles presented in this chapter.

8. Entertainment is important in our lives. It is also one of the biggest industries in developed countries as USA or Japan. The research on theatrical robot and stage/lights editors based on computational concepts will thus have also a considerable commercial importance. A dancing robot will become a toy for all ages. An internet-linked talking partner robot with emotional gestures will be loved by the elderly. Researchers have thus to develop a new form of communication with robots. It will use gesture-based human communications that are thousands years old. All these issues are of our interest here.

The cyber theatre as a whole is a powerful communication system in which various media are used. Lights, music, smoke, fire, water etc can be all controlled uniformly from the same system and can serve artistic purposes of the performance creators. The symbolic gestures of robots, synchronize with stage lighting will transmit emotions and ideas. They should become the base of new human-robot emotion-augmented communication systems. The new extended communication media in addition to speech, will include prosody, facial gestures, hand gestures, body language (head and neck, legs,

bending of full body, muscle-only motions, etc). The gestures used in daily communication, ritual gestures of all kinds, theatric and dance motions will be captured or translated to certain algebraic notations. Descriptions in these notations will be created, edited, stored, recalled, modified and processed in various technical tools.

The idea of reversibility and uniformity of sets of symbols and accepting/generating machines

Motion recognition and motion generation software will be unified based on these new notations, as they are reversible operations, thus they may be based on the same grammar:

1. The unified notation will allow transforming human motions to robot motions through sub-languages of perceptions, motion models, and motion generation.

2. The unified notation will allow also transforming between various media. For instance, a captured motion of a growing flower (in one month) can be presented as motion of a pantomime actor (in 30 seconds). A sound pattern can be transformed to a pattern of colored laser lights. These two patterns can be played together as they are naturally perfectly synchronized.

3. Sound, light, theatric stage movements, special effects and robot motions have all the same nature of sequences of symbolic event-related atoms. Thus the theater itself becomes a super-robot with the ability to express its emotional state through various motions, lights and theater plays orchestration. These sequences can be uniformly processed.

4. The simplest model of the theatrical robot's brain is a finite state machine. Our theory allows creating decomposed machines for perception, transformation and motion; each can be finite state machine or a hybrid automaton.

In this chapter I propose to create universal editors that will be not specialized to any particular medium or robot. They will use algebraic and logic-based notations. The general concept of such editors is based on the theory presented here. One particular exemplification will be given in Chapter 6 where I present in detail the software editor for Portland Cyber Theatre that I developed.

This work is a continuation of previous research done at PSU since 2008:

1. In our previous work we used KHR-1 robots to imitate human body gestures based on vision [38]. The robot behaviors were fully scripted or edited using a commercial editor.

2. Martin Lukac created the CRL language (Common Robot Language) in his PHD work. This system was next used to hierarchically describe robot theatres of emotional humanoids [32].

3. Next, PSU researchers used and developed four biped-robot related motion editors: Heart-to-Heart from KHR-1, our own editor for KHR-1 constructed by Quay Williams [38], ESRA editor [36], and editor built by Mathias Sunardi [39]. The speed of editing motions improved considerably compared to previous, pre-CRL era, but even with these, motion editing is very time consuming.

Thus idea of creating quickly some new motions by motion algebra and evolutionary programming occurred to us. In this line of research, I started also to investigate symbolic and entertainment motions such as in dance, pantomime, ritual and oratory behaviors and sport. This leads to our design of a new editor of motions based on motion algebra, evolutionary programming, feedback from human audience and the "event expressions" that generalize regular expressions. Because our editors incorporate the ideas used in previous editors from PSU, nothing is lost and perhaps new achievements can be obtained.

Various Ideas useful to describe sets of generalized motions for Robot Theatre

The "theory of signs" has been developed in the area of semiotics with applications to theatre, pantomime and human-human communication. The concepts behind dance and

pantomime are truly complex and sophisticated [33]. In future, the owners of home robots would like perhaps to program these robots for dance, pantomime, ritual behaviors or just for daily home gestures that are typical for their cultures; gestures, like for instance, greeting guests and inviting them to enter the dining room. The human that programs such a robot becomes thus an actor and entertainer himself, but he does not want to be a "keyboard programmer" of standard software.

Professor Perkowski observed, while working with teenagers and hobbyists, that even technically talented teenagers who are good programmers do not like to spend long hours to animate the constructed by them humanoid robots [38]. On the other hand, the artistic-minded students invited to the laboratory, much unsophisticated with computers, had hard time to use even the commercial motion editors. We came to the conclusion, that future robot motion developers should be still engineers or technically-interested teens, but that the editor tools themselves should make their work easier.

Concluding these considerations, there exists therefore a need for very fast computer-aided design of long sequences of all kinds of motions for humanoids together with synchronized with them multi-media events, for particular environments and audiences.

Relation to digital speech processing

When a speech signal is created, the technical tools use phonemes, phoneme pairs, consonant-vowel concatenation, consonant--vowel-consonant concatenation (CVC), vowel-consonant-vowel concatenation (VCV) and other symbols. Similar techniques

based on speech synthesis DSP technologies, spectral methods, filtering, etc can be used to process sequences of motion symbols. Some editors allow to display a waveform from a human speech capture and next process this waveform manually to synchronize with motion. The waveform visualization is a base to create motions for degrees of freedom of the robot [ref]. Speech in a text-to-speech system is created by concatenating short waveforms that correspond to basic speech units. The waveforms that correspond to speech can be added by a human animator or created automatically by processing the speech pattern [44]

Much research has been done in speech synthesis and other areas that can be reused for our Theory of Event Expressions.

Relation to music generation and music-controlled robots

Similar ideas occurred in MIDI generated music. They will be used in this thesis. Work of Josh from 479 class in 2011 is an inspiration who created a hexapod robot that dances to the rhythm of music. Similar robots were built independently at PSU by Martin Lukac and Mathias Sunardi. They are based on various theories, different than one presented here, and they should be compared with our robots.

Relation to the theory of stage lighting

Similarly in stage lighting design, the general effect is achieved by parallel (in units) and serial (in time) connection of controls to reflectors and stage lights. The motions of the cameras are often controlled in two dimensions with pan-tilt platforms.

A total theatre is the theatre of unified events including all of the following: actors' motions and speech, sounds, music, lightning and all special effects like shaking chairs of the public. They all become symbolic sequences of generalized events in our approach. As it is known that some phonemes are more likely to follow some other phonemes (see the Hidden Markov Model) so are the typical patterns of light sequences or hand motions of a gesticulating human. For instance, the hand gesture reflects to certain extent the semantic contents of the outputted text. The face motion reflects somehow the pronounced phonemes.

Communication involves the presence of two parties – the source of information (the entertainer) and the receiver of the communication (the audience). Here we are interested in a robot entertainer and a human audience. What is the robot entertainer communicating to the spectators and how? The role of gesture in this kind of entertainment is different than in sports. While in soccer the judge raises the ball in order to physically throw it to the players, in pantomime the ball and its throwing have only symbolic meaning. The information is carried through a physical medium in sports, but in theater the information has to be simulated by the actors so that the audience can apprehend it and understand it. Thus although the theatrical robot has to execute the throwing action, the public is

interested not in the precision of his movement but only in its symbolic value. These motions are special. They are usually ritual and exaggerated. Dance and ceremonial gestures belong to this category of motions as well. The technical design of robot body and the method of motion programming must be therefore different for a soccer-playing robot and a theatrical robot that performs as a soccer player. These issues should be addressed in the methods of robot motion programming.

The motions can be recognized or generated off-line or in real-time. Although our long term goal is real-time generation of motions, here we present the off-line variant in which an editor software tool is used by a human animator to create motions. This tool uses some new theories [31, 32, 33, 39, 40] and representations of motion data.

Having very many low-level motions available, the higher levels of robotic architecture (such as the subsumption architecture that is much used in robotics nowadays) can create extremely many behaviors, as there are several ways to combine elementary motions to complex motions. We should think about the robot behaviors as state machines. In these machines:

1. The perceptions or stored rules stand for the inputs.
2. The moods, emotions and knowledge correspond to the internal states of the machine.
3. And the motions correspond to machine's outputs.

In this thesis we assume that a robot disposes a large library of motions that were created in advance. They are called, in real-time, as responses to the input stimuli. They are pre-specified ahead of the performance. They are not created dynamically. Having ten variants of actor's bowing at the end of performance and selecting one of them as a linear function of the measured sound of total clasping action of the public, we can create a feeling of emotional response of the robot where there is no real reaction, only the selection of stored motions. How far will such a robot differ from a real actor?

How to use the basic concepts of computational theory in robot theatre?

Normally, motion in robotics is presented in the framework of control theory, differential equations, optimal trajectories, PID controllers, and other similar notions. There are no papers that would link motions to the concepts of computational theory such as finite state machines, functional mappings, automata, Hidden Markov Models, diagrams, etc. Other fundamental concepts of computational theory are: Boolean and discrete functions, reversible functions, probabilistic machines, hybrid machines, stack machines and Turing machines, languages, operators on languages, etc. We want to use these concepts to robotics. Specifically, we want to apply these concepts to the design of our new innovative robot motion/behavior editor.

Probabilistic Motion Generators (PMG)

Based on the above assumptions, we can look thus at the robot-actor as a finite state machine or infinite state machine with many evoked motions. These motions are realized as probabilistic state machines called Probabilistic Motion Generators (PMG).

1. PMG is a synchronous Finite State Machine. Thus it has asynchronizing clock and probability generators as the inputs.

2. PMG has sequences of motions on outputs. Each output state is a basic motion.

3. When PMG have finite alphabets of input states, internal states and output states, and no random number generators are used, they are theoretically equivalent to classical Finite State Machines.

4. When probability generators are additionally used, the PMGs are probabilistic FSMs.

5. As the user can always extend the alphabets of basic motions, practically we allow infinite alphabets for our machines.

6. PMGs are generators of output controls to: robot actuators, speech/sound generating units, lights, MIDI interface and home appliances control system.

7. One can think about motions as waveforms. For instance, the ESRA editor [37] starts from a waveform created by a human speech captured and/or processed in special sound-editing software. It is next visualized and referenced to, while creating motions for the individual degrees of freedom of the robot [36]. In a different approach, speech in a text-to-speech system is created by concatenating short waveforms that correspond to basic speech units. The waveforms that correspond to speech can be added by a human animator or created automatically by processing the speech pattern [32]. A digitized waveform is the same as the sequence of basic numeric symbols. Numeric symbols are special cases of symbols. Thus waveforms are special cases of symbolic expressions discussed in this chapter.

8. This way, processed sound or speech signal is a symbolic sequence, which can be used for instance to control a robot action or can be used to control **robot emotional behavior.** Like a robot being in an emotional state "happy" when we praise him using certain sensor inputs (patting on back) or texts.

9. Based on the above, a waveform can be a base of a speech or physical behavior, including speaking action which involves synchronization of the motion of the mouth and the sound [37, 40].

10. This has also relation to fuzzy logic. If one treats every value of fuzzy signal from interval [30, 31] as a symbol, a motion (speech and motions of potentially many robots, whole theater) becomes a sequence of symbols (with an infinite alphabet). Similarly, one can use multiple-valued logic with any number of values for every type of signal, motion or discrete event. As an example, a ternary logic can be used to represent a position of a head: 0- neutral, 1 – turned to left, 2 – turned to right. Operators of fuzzy logic can be thus used in the framework of our theory.

11. A sequence of symbols reminds a "word" generated in generative grammars. Generating such words by grammars corresponds therefore to generating complete motions (event sequences).

12. A sequence of symbols reminds a "word" transformed in transformative grammars. Transforming such symbolic sequences of symbols is thus transforming motions to other motions. We can thus take the word (or a language) and transform it to another word (or a language). This would correspond to automatic creation of new robot behaviors from a behavior captured from a human or created using a simple motion editor.

13. Every motion generated is useful or not, for a given robot, in a given situation/environment. Thus it should be somehow evaluated. The simplest approach is when a human director evaluates the motion. If the editor tool allows generating and evaluating quickly very many motions we can quickly useful motions for the entire robot theatre. These motions can be those that exist in the world or they can be new motions, which may have some entertainment or even cognitive values. Think about motion composed from the legs' movement of a female waltz dancer and the upper body movements of a heavy-weight boxing champion.

Concluding the above discussion of motions generated by PMG, from now on, the motion will be a sequence of symbols. Motion in our understanding is an arbitrary sequence of arbitrary "elementary motions" (events). This sequence can be interpreted as sound, robot motion, stage lighting cues, or any other observable events. Observe that lights and instruments, smoke and steam generators are parts of our generalized symbolic motions on equal terms.

Let us emphasize now the following points:

1. The presented concept of a "generalized motion" or sequence of event is general.

2. Creating motions is hierarchical. There is an aspect of creating basic motions and micro-motions, the aspect of creating macro-motions and the aspect of adding flavor to the motion.

3. In general there is only a limited range of applicable motions in a given situation. The meaning depends on context, i.e. spoken text, lighting, sounds, sequence of past events. What changes motions is the flavor of them.

4. Depending on the situation some motion will have some aspects exaggerated and some inhibited. Sometimes the parts of the motion will be skipped while some other parts will be changed. This all goes towards creating template personalities for a given robot. For instance an unhappy robot template will be doing the motion not very precisely, in general a little bit late and slower. He will try to avoid looking at the audience and so on. This is partially possible through the Laban approach but we believe that the problem is more complex than just to generate a set of rules for Laban notation.

5. We believe that a more holistic approach is required that will allow to specify constraints of the overall motion generation. These constraints should have some deeper understanding of context. One of questions is “should they depend on the feedback from the audience?”

Off-line versus on-line expressions

Motion for robot theatre can be generated off-line or on-line. Off-line, means that a motion is generated that is next demonstrated on a robot. The motion is a file that can be reused by the system designer. Off-line expressions can be generated slowly, as the time of their generation is not critical. On-line means that the motion is generated which is immediately used by the robot. This means that the motion expression must be generated very quickly so the behavior of the robot is not slow down. The motion is generated with some parameters that result from the current robot environment.

Of course, real-time motion creation is more interesting as it allows the robot-actor to react to the quickly changing environment, such as public applause or other verbal reactions, or other unexpected events on a stage, such as robot falling down. These kinds of behavior creations demonstrate the adaptive characteristics of our system and should be definitely appreciated by the theatre's audience. However, the motions cannot be generated by our system, if they are first not analyzed and understood. For instance, some motions may lead to robot's falling down, or doing something else that is not desirable by the director. Therefore, creating an off-line tool is in any case a prerequisite to create a real-time motion-creating software. In this chapter we concentrate only on off-line expression creations.

The question of on-line versus off-line creation of motions is a fundamental question from the point of view of the philosophy of the robot theatre. As we know, a film is an art that does not require feedback from the audience. In film, every event presented to the movie audience is always exactly the same when the tape is played next time. The creation of the event has been done earlier, in an off-line mode. It was created by the director and his associates during the shooting of the movie and its editing. In contrast, in standard theatre with human actors, every performance is slightly different, because of different moods of actors, and some randomness of the real bodies and their motions. Also, every performance instance heavily depends on the feedback from the audience. The public may say "Actor X had a good day today". This will not happen in the case of the movie art.

From the point of view of robot theatre, the research questions in the on-line/off-line domain are the following:

1. "will the future robot theatre be more similar to films or theatrical performances?",

2. "can one create a realistic robot theatre without viewer's feedback?"

3. "What type of tools do we need to create a theater play off-line versus on-line?"

4. “Is an off-line sequence generator enough to create art?”

5. “Do we need a sequence generator and a virtual reality simulator before actuate the play on the robot”?

6. “How really important is the feedback from the audience for a human singer or a conference speaker to modify their on-stage behaviors? How it should be reflected in the on-line robot theatre?”

7. Should we have a “character” of each robot simulated together with the performance? (So that a shy robot should run from the stage if it is booed. A strong robot should remain. But maybe the shy robot will know much better how to express the meaning of the play.)

8. In general, do we need a “character” to use the feedback from audience or should be this simulated otherwise?”

Concluding, the “event expressions” can deal with the output aspect of “generating motions”, its reversible aspect of “recognizing motions” and the transformative/behavioral input/output aspect of “generating motions as responses to

motions". These languages correspond to various types of automata. In this thesis we are concerned with the off-line "motion recognition" and "motion generation" aspects only.

Event Expressions to specify languages of motions and behaviors

The introduced by me event expressions are extensions of the classical regular expression. Therefore, we will review briefly the regular expressions here.

A *regular expression* is a pattern that matches one or more strings of characters from a finite alphabet. A regular expression is composed from operators and atomic symbols – characters. They satisfy certain grammar that generates them. Individual characters are considered regular expressions that match themselves. The original regular expressions used union ($\cup$), concatenation (symbol ^{o}or nothing), and iteration (*) operators. Regular expressions were next extended by adding to them two operators: negation and intersection, known from the Boolean Algebra. Therefore, the Extended Regular Expressions can be built as Boolean Networks with flip-flops, or standard binary automata.

Observe that for two atoms, the intersection $X1 \cap X_2$is an empty set foratoms $X_1 \neq X_2$, as the meaning of intersection operator is set-theoretical. Thus, intersection, similarly to other operators of Extended Regular Expressions does not create new symbols. Such expressions can be then used to create sets of motions where atomic symbols are basic

motions. While a concatenation of atoms describes a single motion, an Extended Regular Expression describes a set of behaviors (motions) but all of them use the atomic motions from some finite library. This can be next extended to infinite library, in which the atomic motions can be enumerated like this $a_1, a_2, a_3, \ldots a_n$, for any value of n.

Similarly the interpretation of the union (set sum, or $\cup$) operator is set-theoretical in regular expressions, thus new symbols are not being created.

The base ideas of event expressions are these:

(1) Symbol (represented by sequence of characters) is a basic event or a set of basic events synchronized and executed in parallel.

(2) Symbols can be connected in parallel (for instance, *(a)* text spoken, *(b)* leg motion, and*(c)* hand motion can be created relatively independently and combined in parallel). Connecting symbols in parallel creates new symbols that can be used as macros or subroutines.

(3) Symbols can be connected in series; we call this symbol concatenation, which is the base operator of our theory. Concatenation is also the main concept of regular expressions from which we borrow to create the Event Expressions.

To create therefore a new motion, the human animator has to do the following:

(1) Know the set of basic symbols. (This can be a large library in which every basic motion is described by a name)

(2) Select the symbols and combine them in parallel. This means that these basic actions are executed in parallel, synchronized in the starting moment. Give new names to these combined actions.

(3) Concatenate the symbols.

(4) Give new names to concatenations.

(5) Proceed recursively and hierarchically, creating higher and higher behaviors in the hierarchy.

While creating the generalized events, it is however often more convenient to think in terms of sets of motions rather than single motions. For instance a greeting motion of a humanoid robot can be described as the following regular expression:

$$Greeting\_1 = (Wave\_Hand\_Up \circ Wave\_Hand\_Down)(Wave\_Hand\_Up \circ Wave\_Hand\_Down)^* \cup Wave\_Hand\_Up \circ Say\_Hello$$

Which means, to greet a person the robot should execute one of two actions:

Action 1: wave hand up; follow it by waving hand down. Execute it at least once.

Action 2: Wave hand up, next say "Hello". The same is true for any complex events.

As we see, the semantics of regular expressions [31] is used here, with atomic symbols from the terminal alphabet of basic events

{***Wave_Hand_Down, Wave_Hand_Up , Say_Hello***}.

The operators used here are:

(1) concatenation (^{o}),

(2) union ($\cup$) and

(3) iteration (*).

Each operator has one (iteration) or two (concatenation and union) arguments. So far, these expressions are the same as the regular expressions. They are now expanded to event expressions by recursively adding more deterministic and probabilistic operators on event expressions (this recursive definition is similar to the recursive definition of regular expressions [31]). For instance, if we agree that the meaning of every operator in ***Greeting_1*** is that it executes its first argument with probability ½ and its second argument with the same probability, each possible sequence from the infinite set of motions of ***Greeting_1*** will have certain probability of occurrence. One can create event expressions using both deterministic and probabilistic, single-argument and two-argument operators. The operators in classical regular expressions and extended regular expressions are deterministic. However, in Event Expressions we can give both deterministic and probabilistic interpretation to the operators.

For instance, in the first example, assuming that concatenation is a deterministic operator and union is a probabilistic operator with equal probabilities of both arguments, there is a probability of ½ that the robot will greet with motion ***Wave_Hand_Up*** ^{o} ***Say_Hello***.

In another example, assuming that all the operators are probabilistic operators with equal probabilities of selecting any of both arguments, for event expression ***Wave_Hand_Up*** ^{o} ***Say_Hello*** there is a probability of ½ that the robot will greet with one of the following four motions, each with the same probability:

(1)*Wave_Hand_Up* ^{o} ***Say_Hello*** ,

(2) *Wave_Hand_Up* ,

(3) *Say_Hello* ,

(4) Nothing will happen.

As we see, in this case for each of thetwo arguments of concatenation there is the probability of ½ that it happens and probability of ½ that it does not happen.

Several extensions can be done to the above concept. They all are based on the new semantics of operators. Now operators applied to single symbols can create new symbols. For instance operator MIN applied to arguments 3 and 23 will create value 3.

Similarly to the above operators, the user of the editor can use many additional operators, deterministic or probabilistic, to define "extended" event expressions. Several such "extended operators" can be created for any standard operator of regular expression.

Next, the user can define own new operators. These operators can have temporal, multiple-valued and probabilistic/deterministic nature.

Our system of Event Expressions and corresponding state machines uses an expanded set of operators taken from multiple-valued logic:

a. literals,
b. MAX,
c. MIN,
d. truncated-sum,
e. Module-addition
f. And others.

The symbols are interpreted as having numerical values for them.

Having arithmetical operators that create new symbols allows also for interpolation (Hermite, Spline, Radial Basis) and spectral operators based on Fast Fourier Transform (FFT).

The user can design arbitrary multiple-valued operators of this type and add them to the compiler.(this is not yet implemented). These operators can be specified by tables or by calls to subroutines (maximum, truncated sum, etc).

The one-argument operators on event expressions include:

a. mirror (reverse),
b. palindrome,
c. repeat k times,
d. shift-to-beginning,
e. shift-to-end,
f. shift-to-middle-
g. And others.

The system has also “energy operators” and “emotional operators”. Energy operators modify the energy level of the robot. For instance, by low-pass filtering in the frequency domain we can change the “energetic pushup” motion to a “dying old robot” motion [39, 40]. All these operators are specified as calls to lookup-tables or numerical “fuzzy-like” subroutines.

The emotional operators allow the user to perform hierarchical global operations on macros and subroutines to express emotions such as “sad, happy, and confused” and others (as in the research on Martin Lukac about simulating emotions of robots). These are time-related, frequency-related and other operators that are all parameter-controlled.

Spectra can include Fourier, Haar, Hadamard or any other transform. Spectra allow for realizations of filters.

The control may come from the user/animator or from the environmental feedback. The changes in controlled parameters are expected to not change the actual content of the motion but only its artistic or emotional aspects. It was found however that if the modification is high, the meaning can change – it depends on the frequency scope of filtering if the motion of the robot will look still as a "*tired old robot doing pushups*" or a "*dying robot*", [39].

The symbols can be of three types:

- input symbols,
- output symbols and
- Input/output symbols.

The ***input symbols*** are basic motions as perceived by the perception system, for instance "*raise right hands lightly*". The ***output symbols*** are basic motions as generated by the actuation system, for instance "*raise right hand slightly*" [38, 39]. They are similar to input symbols but they both depend on the type and quality of perception and actuation hardware/software. The input/output symbols are conditional pairs of motions, such as "***if*** actor smiles ***then*** raise your right hand slightly". They relate two motions/events and they include feedback and temporal aspects.

Machines to generate, accept and transform Event Expressions

Event expressions are very general and they have many interpretations. These interpretations deal with generation, acceptance and transformation of motion (event) languages. The user can create a PMG or a Hidden Markov Model (HMM) to generate all motions described by event expressions. A regular language is accepted or generated by a Finite State Machine. Thus a regular expression is in a sense the same as the state machine that generates or recognizes the language expressed by it. There is a one-to-one correspondence: for each machine there is an expression, and for each expression there exist a machine. The problem is: "how to transform one to another?".

These properties can be formalized and analyzed theoretically. Because of different structural construction of these machines, it is convenient to distinguish the generators of languages – the PMGs and the acceptors of these languages – the Event Expression Accepting Machines (EEAM). While the generators generate sets of motions, the accepting machines recognize the sets of motions. For every PMG_i that generates language $L_i$there exists one accepting machine, $EEAM_i$that accepts this language. Moreover, there is a one-to-one direct transformation from $PMG_i$to$EEAM_i$that does not require going back to the Event Expression. It will be illustrated (but not proved) below.

Next, the PMG machines and EEAM machines can be extended to a wide category of probabilistic machines, of which the HMMs are only a special case. Similarly, the concept of the set of generalized event sequences (event expression) can be considered the same as the probabilistic autonomous PMG that generates it, or the Probabilistic Machine that accepts these sets of sequences.

The simple motion is a single sequence of basic symbols generated by a PMG machine. The theatrical play is represented by a Complex State Machine(CSM). CSM is composed of many EEAMs on inputs, many PMGs on outputs and the Behavior Machine (BM) between them as the main knowledge bearer of the theatrical play or the individual robot character. Output symbols of BM revoke(in parallel) one or more of the PMG generators. Symbol inputs of the BM come(in parallel) from many Event-Expression-Accepting machines (EEAM). BM is a deterministic or probabilistic machine. Formally CSM machine is also a probabilistic state machine, but it is practically convenient to describe it as a hierarchy and sequence of machines of various types. The decomposition of CSM is done off-line by the designer. It would be not possible to describe the motion or behavior as one machine. The user can work incrementally; he designs or modifies one machine at a time. This activity of the designer is similar to designing a large computer architecture from blocks such as counters, registers, controllers, memories, etc.

The inputs of EEAMs come from: (1) the robot's sensors and (2) the speech patterns that are generated by the audience of the play [40]. Our algebraic motion-describing system extends thus the concepts of regular expressions, MV logic, HMMs, state machines and spectral theory. These expressions and machines can be built by hand by the animator, or they are evolved by using the evolutionary computing principles [32,40].

Motions have many flavors: for instance in dance there is a classic ballet, ballroom dancing, rock-and-roll and rap dancing. There is a classical and English Pantomime. There are various rites for rituals. It has been a human creativity to invent these symbolic

sign systems for motion with all their artistic, cultural and emotional nuances. Now we can have an automatic system to design these motions. The issues of gait generation for humanoid robots, dance decomposition to individual steps and analysis of ritual gestures are known. They can be all used to invent new operators for evolutionary algorithms that can be added to classical crossover and mutation operators.

We can also consider more rule-based type of behaviors with some "modification probabilities" for the operators used in expressions. One of the possibilities is to create the event expression "by hand" and next to compile the expression automatically to PMG. This will be presented in the next sections.

Regular Expressions, Deterministic Finite Automata, Event Expressions and Probabilistic Motion Generators

As presented in section 5, our system of Event Expressions and corresponding machines allows executing additional operations on atoms, such as their MIN, MAX, etc. Thus new symbols are created by these operators and the sets of symbols become potentially infinite (natural or rational numbers). For instance, the MODULO-ADD operation takes two symbols from the alphabet and creates the symbol in the alphabet, but the Divide operator or ADD operator can create new numbers, which means – new symbols. This is the main difference between regular expressions and event expressions. Event expressions allow also to create dynamically new operators, which is not possible in

regular expressions. Regular expressions are widely used in software applications such as data mining, syntax highlighting systems, and parsers for computer languages. As classical regular expressions are the special case of event expressions, the understanding of event expressions becomes easier for the reader. Paper [5] explores hardware applications of regular expressions by detailing state machine (NFA) construction directly from regular expressions using FPGAs. The reason for implementation using FPGAs, as opposed to a solely software implementation, is the speed of the string matching [34]. Event expressions can be implemented in software, hardware and mixture of them. FPGA implementation will allow for very fast processing. The NFAs, EEAMs and PMGs presented here can be all easily converted to machine formats that can be handled by standard FPGA and VLSI CAD software tools.

Regular expressions are used to generate standard NFAs. A NFA has one state per identifier in the regular expression. And a circuit can be constructed directly from a single regular expression, using one memory element per NFA. A NFA constitutes a directed graph where each node is a state and each edge is labeled with a single character. An outgoing edge having the identical label in a NFA, for circuit synthesis, translates to a specified flip-flop being connected to the input of more than a single flip-flop. An advantage of an NFA is that it can have edges labeled ϵ. An ϵ edge is a connection from the input of a source flip-flop to the input of a destination flip-flop. Thus if all edges from a source flip-flop to a destination flip-flop are ϵ, the source flip-flop can be

eliminated, thus greatly simplifying the synthesized circuit. The graph of NFA can be interpreted as acceptor, generator or transformer depending on the meaning of symbols used in it. Paper [31] was the first to convert regular expressions to NFA, skipping the step of initially deriving a DFA from the NFA, and directly implementing the NFA with FPGA hardware. Both these approaches have some merits for hardware but the approach from [31] is better for software.Our approach generalizes regular expressions to event expressions and NFA to PMG, BM and EEAM. While the Brzozowski's method [31] creates acceptors (NFA), our method can be used to generate both acceptors (EEAMs), behavioral transformers (BM) and generators (PMGs), with both probabilistic and deterministic properties. Here we will discuss only the generation of deterministic acceptors and probabilistic generators.

Hardware vs Software Implementation of NFA and PMG generation from Event Expressions

As shown below in Fig. 1, a single flip-flop is associated with each state and at a given time, exactly one flip-flop stores a 1-bit signifying the active state. The combinational logic ensures the 1-bit from the active flip-flop is transferred in the next clock cycle to only a single flip-flop. A node with multiple outgoing edges with the same label is realized as a flip-flop output that is connected to the input of multiple flip-flops. Є edges

are realized by connecting the input of the source flip-flop to the input of the destination flip-flop.

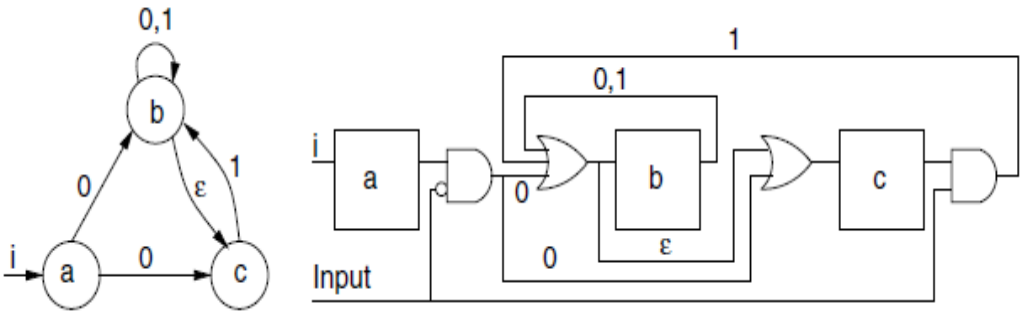


Figure 2:1 An example of a graph and corresponding hardware logic implementation of NFA. Similarly EEAM, PMG and BM can be realized in hardware using binary, fuzzy and MV gates and other operators

The NFA construction algorithm can be constructed as a program that outputs the NFA as a placed and routed netlist and uses vendor tools for generation of configuration bits. While NFA logic can be simple and fast, the bulk of the execution time is spent mapping the NFA.If the FPGA can generate configuration bits at runtime and utilize them to modify its own existing configuration, it inevitably allows improvement in performance gains and increased flexibility. This ability of self-reconfiguration greatly reduced the time allocated to the mapping of the NFA. Brzozowski's derivatives of regular expressions serve as a sound method for transformation of regular expressions to NFA. This method not only directly converts regular expressions directly to NFA bypassing a step of initial transformation of DFA, but also elegantly handles extended regular expressions. Given *E*, a regular language over alphabet *X*, a NFA is derived that accepts all words from the language, and only those words. All words of *E*, starting from letter X_j

ϵ X. If the letter is removed from the front of each word from the set, a new language is created, referred to as left-side-derivative of language *E* by letter X_j. This new language is now denoted by E/X_j.A derivative for word $s = X_{i1}, X_{i2}\ldots X_{in}$ is defined as follows: $E/X_j = \{s \epsilon X^* : X_j s \epsilon E\}$.

As inherent laws of Brzozowski's derivative method, the following properties $\mathbf{P_i}$ always hold true.

P1. Xi/Xj =e for i = j

= ø for i ≠ j

P2. (E1∪ E2)/Xi = E1/Xi∪∈ E2/Xi

P3. ∈(E) = e when e ∈ E

= ø when e ∉ E

P4.E1E2/Xi=(E1/Xi)E2∪∈ (E1)(E2/Xi)

P5. E/s=E1/Xi1Xi2…Xin=[[E/Xi1]/Xi2]…]/Xin

P6. E*/Xi = (E/Xi)E*

P7. E/e = E

P8. (E1 ∩ E2)/Xi = (E1/Xi) ∩ (E2/Xi)

P9. (-E)/Xi = -(E/Xi)

P10. (E1max E2)/Xi = (E1/Xi) max (E2/Xi)

Observe that the machine realized according to these rules can be realized both in hardware (FPGA) or software.

Extensions of operators

The derivation-based system can include many other rules similar to rule P10 for MIN, MAX and other MV operators.

P10A. $(E_1 \min E_2)/X_i = (E_1/X_i) \min (E_2/X_i)$

P10B. $(E_1 \text{truncated-sum } E_2)/X_i = (E_1/X_i) \text{ truncated-sum } (E_2/X_i)$

P10C. $(E_1 \text{parallel } E_2)/X_i = (E_1/X_i) \text{ parallel } (E_2/X_i)$

P10D. $(E_1 \text{modulo-add } E_2)/X_i = (E_1/X_i) \text{ modulo-add } (E_2/X_i)$

P10E. $(E_1 \text{divide } E_2)/X_i = (E_1/X_i) \text{ divide } (E_2/X_i)$

There are many rules similar to P9 for literals and rules similar to P8, P2, P6 and P4 for probabilistic variants of operators ∩,∪,and concatenation, respectively (their meaning has been explained in section 3).

With these properties at our disposal, an example with the following language is given= $(X_2X_1^* \cup X_1X_2)$.

Applying the left-side-derivative with respect to first character in string, X_1

$E_1/X_1 = (X_2X_1^* \cup X_1X_2)/X_1 = (X_2X_1^*)/X_1 \cup (X_1X_2)/X_1$

by P2= $(X_2/X_1)X_1^* \cup \in (X_2)X_1^* \cup (X_1/X_1)/X_2 \cup \in (X_1)(X_2/X_1)$

by P4= $\emptyset\, X_1 \cup \emptyset(X_1/X_2) \cup eX_2 \cup \emptyset\, \emptyset$

by P1= X_2

This corresponds with a state or node in the NFA, in this case we label it E_2. Thus there is an edge in the NFA labeled X_1 going from node E_1 to node E_2. Applying the properties of Brzozowski's derivatives to the regular expression $E_1 = (X_2X_1^* \cup X_1X_2)$, result in the NFA shown in Fig. 2. Observe that this graph can be interpreted as an acceptor, when symbols Xi are inputs. It can be interpreted as a generator when symbols Xi are outputs. The graph can be thus used to recognize if some motion belongs to some language and can generate a motion belonging to the language. This graph is realized in software in our approach. All, PMG, BM and EEAM machines can be created from it, depending on the types of the symbols used in the graph. Each of these machines can be realized in software (our approach) or in hardware. As some of our operators are fuzzy or MV, respective fuzzy or MV hardware can be used to realize them.

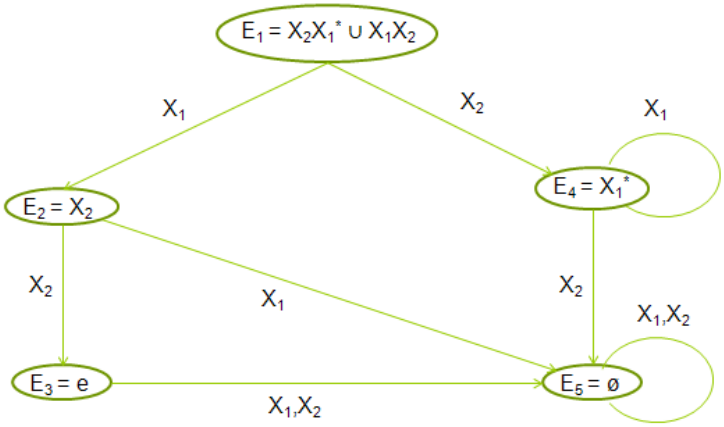


Figure 2:2Graph for regular language E1 = (X2X1*∪ X1X2). It can be interpreted as PMG, EEAM or BM depending on meaning of symbols Xi

.

Conclusions

In this chapter I proposed a new concept of describing very general sets of events for new types of robot theatre. I introduced a notation and theory of symbolic motion to be used in designing motions and events. The concepts of motion atom, basic and complex motions were introduced. I introduced also the new concepts of event expressions (EE) and explained them in a state machine notation. A set of operators on motions, i.e. operations on machines that generate these event sequences were also defined. In future the user of my editor can define new basic motions and basic operators on them, he will be also able to redefine probabilities.

The methods outlined in this chapter allow to automatically create the motion-accepting and motion-generating machines from EEs. These machines can be realized in both software and hardware. The approach from this thesis emphasizes software editor. However it is possible that in future there will be a hardware (FPGA-based) system [34, 35] that will create sets of behaviors in real time and then select one most adapted to the state of the environment or action.

When I played with the editor that I built, in which a simplified version of the theory presented in this chapters was implemented, I found that many new motions, motion genres and input/output behaviors can be created automatically by our simplified "universal event editor". Therefore, it is very probable that the editor, which will have all

possibilities of the theory from this chapter, will be even more powerful by creating new and unpredictable behaviors.

Using automatic behavior scripting / music/ lights editing tools extends the set of possible motions and thus stimulates also the human creativity of the stage directors, music directors and stage designers. In future, individual people will be such directors – teenagers will be able to create complete new theatre performances of robots. We already see the first attempts at such performances on YouTube and Internet where many short movies of humanoid robot behaviors are shown by young people and robot hobbyists.

An interesting open problem is how to connect various motions, how to structure more complex scripts, how to make smooth transitions from scripts to scripts. In the simplest variant the robot has to come back always to the initial neutral position. But we can create few intermediate positions and have rules that every $motion_1$ must end with the same position as the concatenated to it $motion_2$starts. This can be added to the evolutionary algorithm that evolves behaviors. A software demo for the committee will demonstrate the use of the editor to create new motions and the video will present several emotional and theatric biped motions generated by the first variant of our editor.

Chapter 3 Programming of Arduino board for iSobot

The primary purpose of the motion editor for robot is to be able to control the robot. Controlling robot means to send the command strings (HEX codes) through such physical medium. We chose iSobot robot for this project whichcomes with rich set of command pool.

Up until now owning a robot with true humanoid flexibility, balance, and movement has been outside the price range of the average consumer. Typically costing $1000 or more for an entry level package, the popular [20] robot series and others like it have been the sole domain of the hard-core hobby roboticist. Tomy [21] decided to change all that with its terrifically fun, talented, and tiny **i-Sobot** robot.

According to the **Guinness Book of World Records**, standing only 6.5 inches tall (16.5 cm), i-Sobot[Figure 3-1] is the **world's smallest mass produced humanoid robot**. It sings, dances, rolls, kick, and do just about everything you would expect a miniature robotic man to do. It also responds to 10 preprogrammed voice commands and comes with a hilarious collection of special action commands, which include animal and celebrity impressions. Finally, it can be controlled in proper fashion for robot soccer tournaments, races, and other friendly competitions via its powerful infrared remote control. The iSobot comes fully assembled and ready to play right out of the box.

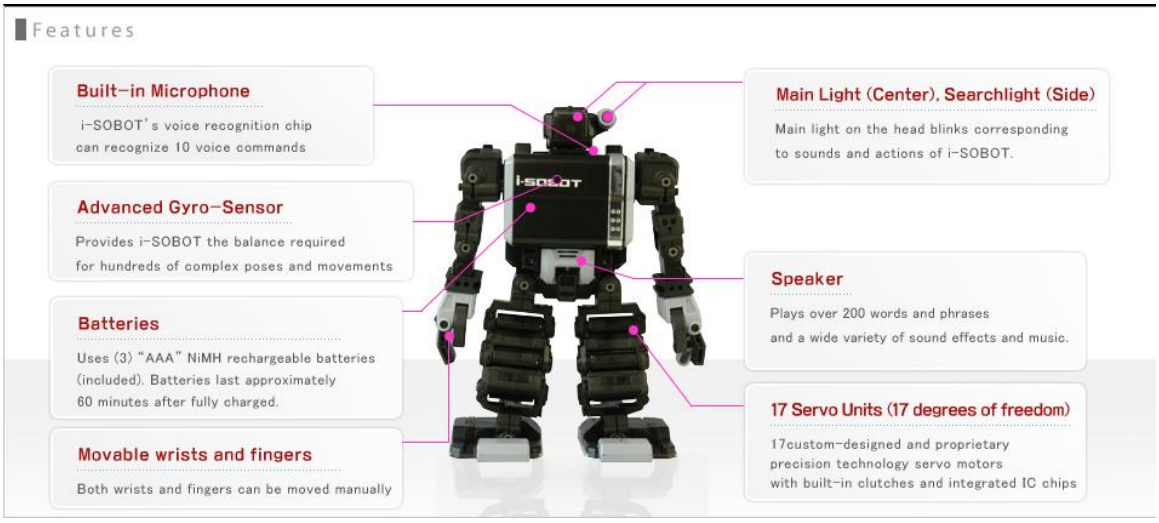


Figure 3:1iSobot Features

Using the iSobot remote control and its manual [appendix A] for commands list, user can push combination of buttons on the remote and send one command to the iSobot. The iSobot performs that single action at a time and can take the next command only after finishing the current command execution.

Procedure to send one command using remote:

1. Search for the command in the manual & pick one.
2. Read the command number from the manual & memorize.
3. Hold the remote right in front of IR sensor of iSobot and push the combination of buttons just memorized. Most commands require 3 to 4 buttons in the right sequence.
4. Watch & enjoy the action and wait to send next command until it finishes first command execution.

This little toy is absolutely fun with its nice action set but can quickly get boring when user has to struggle to follow the above procedure every time to send each command. An automaton script can be useful to take care of these repetitious works and let user have more fun with ease of access to the commands. Hence an idea of editor generated which does job of this remote, have some kind of interface to send command to robot, provides GUI and let the user pick commands quickly and send the same with a button click.

Abstract

The purpose of this chapter is to capture the requirements and related theoretical and implementation details for a unit that enables an iSobot to be wirelessly controlled via a personal computer. The later section of the chapter details briefly about PC controller software built in Python.

Requirements and Objectives

The following list of requirements and objectives are meant to guide the development of the PC-based i-SOBOT Controller (PC-IC).

1. The PC-IC must be capable of transmitting all of the messages generated by the i-SOBOT hand held controller.
2. PC-IC must be hardware compatible with one or more standard computer I/O interfaces (e.g. serial port, USB port, etc.).
3. PC-IC must be software compatible with standard PC operating systems: Windows, Linux, and MAC-OS.

4. Wireless range of PC-IC shall be no less than that of i-SOBOT hand held controller.
5. PC-IC shall be easily replicated. Extensive soldering or construction of custom printed circuit boards shall not be required. PC-IC hardware shall be obtainable commercially and require minimal modification (e.g. addition of infrared LED).
6. PC-IC will draw power from host PC and will not require its own power supply or battery.

Proposed solution: "Miles Moody" Hack

An existing design, which is capable of enabling communication from a PC to the i-SOBOT, can be found on the web [19]. The solution's designer, "Miles Moody", employs an "Arduino board" to both acquire the i-SOBOT IR command codes and retransmit them on demand via an IR LED.

The Arduino board can be commercially obtained for less than $30. The solution's firmware source code is available for download and is "free and open to the public" with no other restrictions such as the GNU Public License.

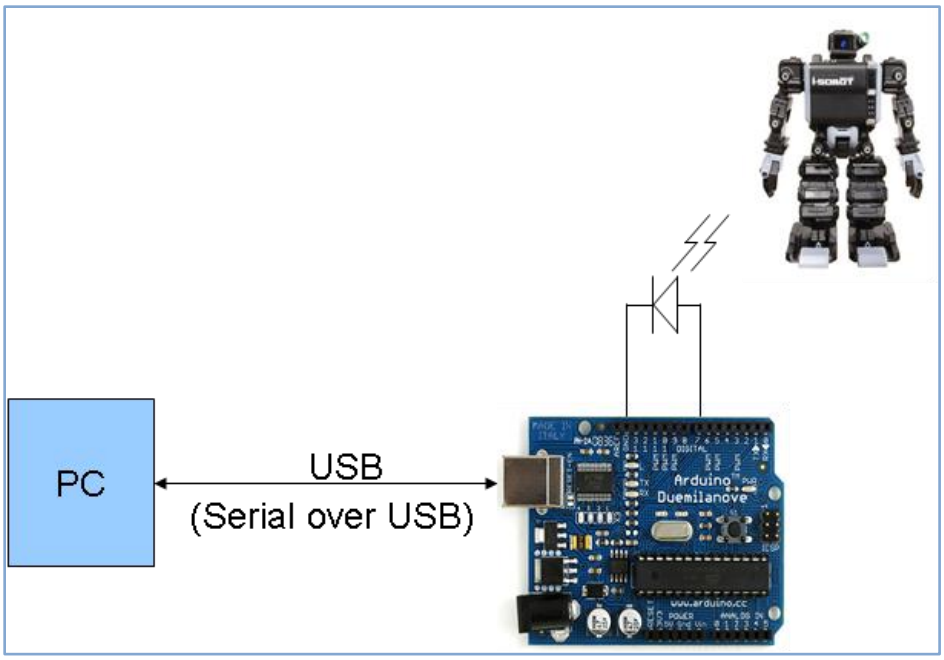


Figure 3:2Hardware configuration for iSobot Controller

Figure 3.3 shows the envisioned hardware configuration. The Arduino board communicates with and draws power from the host PC via USB cable. During normal operation, the PC sends commands to the Arduino board which relays them to the iSOBOT by modulating an infrared LED.

iSOBOT IR PROTOCOL

Knowledge of the iSOBOT IR protocol comes from the work of Miles Moody who has been effective at deciphering the communication interface.

The protocol utilizes a modulated 38 kHz carrier (which is common for IR remotes). It is believed that there are two different types of commands which can originate from the iSOBOT remote: 22-bit commands and 30-bit commands. Most commands are of the 22-bit type while those allowing fine grain control of the robot's arms include an additional byte to specify servo position.

Both commands begin with a 2.5ms long burst at 38 kHz, presumably a start of frame (SOF) delimiter. The SOF is followed by a series of alternating silence and 38 kHz bursts as shown in Figure 3.3.

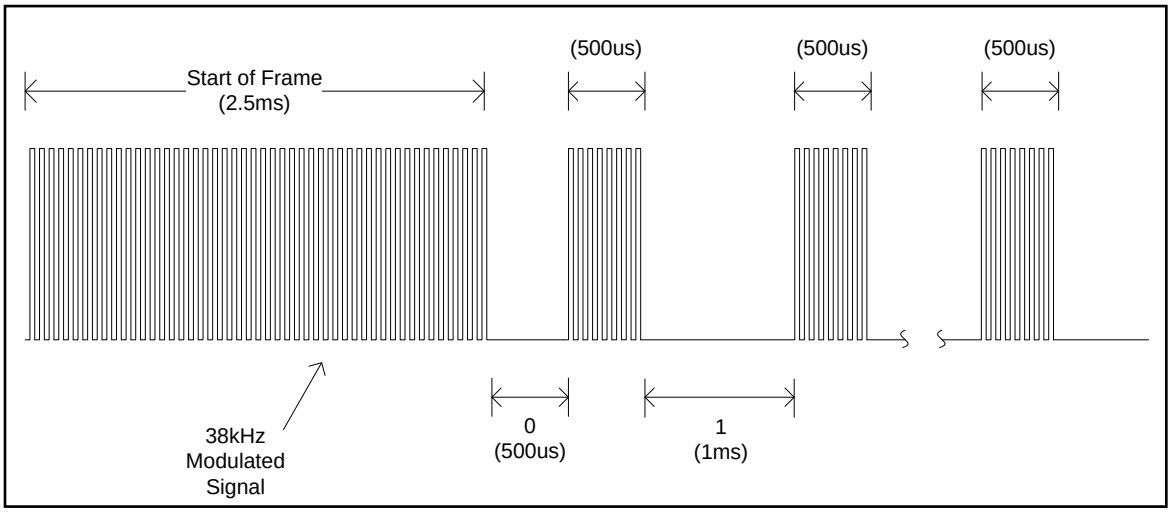


Figure 3:3iSobot IR Protocol Timing

The bursts are approximately 500us long. The data is encoded in the length of the gap between bursts which is either 500us or 1ms long. Per standard IR remote convention, 500us silence corresponds to logic 1 and 500us corresponds to logic 0.

Interfacing Arduino to IR LED

A circuit is required in order to enable the Arduino microcontroller to modulate an IR LED.

Testing has shown that a relatively high power (80mW or more) IR LED is required in order to achieve reasonable range (4' or more) when sending commands to the iSOBOT. IR LEDs with this kind of power output generally require 100mA or more of operating current.

Unfortunately, the AtMega328 microcontroller [26] on the Arduino board has a maximum current output of 40mA per I/O pin and, more importantly, a rated current output of only 20mA at an output voltage of 4V. This current is insufficient to drive the LED.

Ganging multiple I/Os together is possible, however, each I/O should have its own current limiting resistor in order to assure load sharing. In order to achieve a reliable connection from the microcontroller's I/Os through four or more limiting resistors to the IR LED, one should probably consider using a small prototype perforated PCB. We used small dot matrix board.

Once the Rubicon of adding a PCB has been crossed, it is of little additional cost to add a transistor in order to make a more general LED driver circuit.

Driver Circuit

Figure 3 shows the simple circuit used to drive the IR LED.

Target current for the NTE 3027 IR LED is 150mA. Assuming a nominal forward operating voltage of about 1.2V, a supply voltage of 5V and negligible Vce across the BJT gives:

$$R_{lim} = (V_s - V_f)/I_f = (5 - 1.2)/150mA = 25ohms$$

But 150mA through 25ohms is:

$$I_f^2 R_{lim} = (150mA)^2(25) = 0.5625W$$

Which is greater than the capacity of readily available 1/4 W resistors. To address this issue, four 100ohm resistors are employed in order to dissipate the required power.

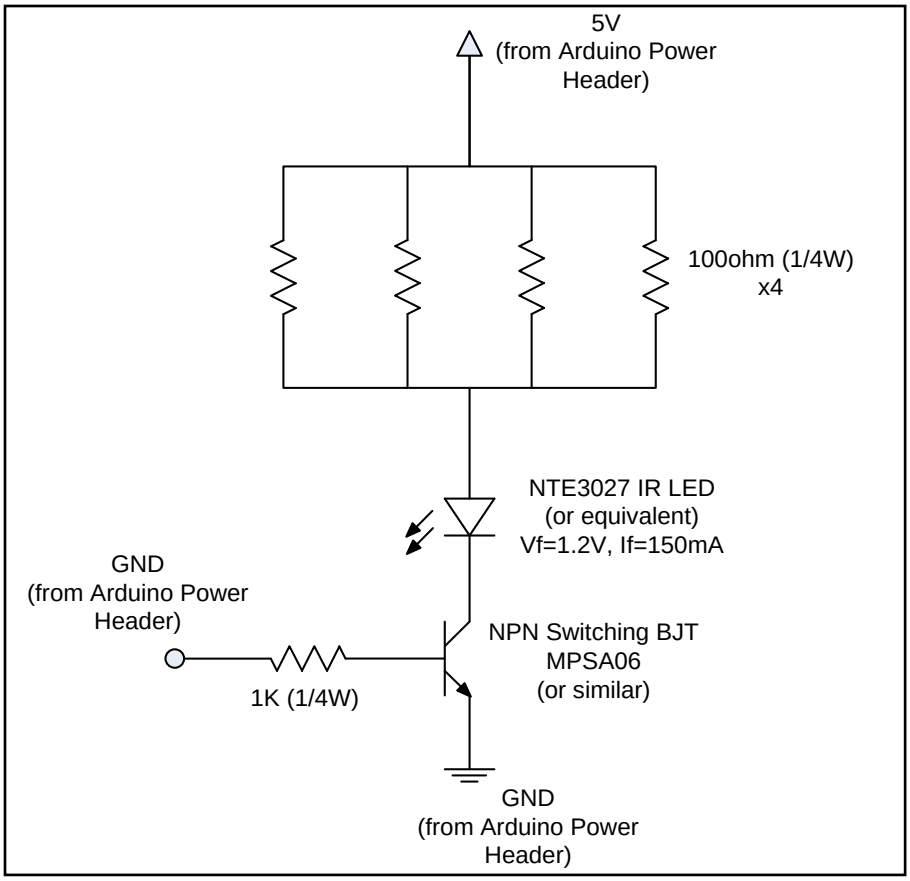


Figure 3:4Circuit used to drive the IR LED

PC-to-ARDUINO LINK: SERIAL OVER USB

The serial over USB link between the PC and the Arduino board serves two purposes:

1. Normal Operation: The Arduino board has on board USB-to-serial chip. The chip's manufacturer, FTDI, supplies and supports drivers for all of the popular operating systems (Windows, MAC OS, Linux). During normal operation, the FTDI chip provides a virtual COM port. From the perspective of the host PC and

the Arduino's microcontroller, they are effectively connected by an RS-232 serial link.

2. Flash Programming: The Arduino's on board microcontroller has internal flash memory which is loaded over the USB link using the Arduino Integrated Development Environment (IDE). The protocol used for this firmware update process is the STK500 protocol from Atmel. The Arduino boot loader briefly enters STK500 mode upon reset but quickly times out if it does not receive the expected STK500 commands. After the timeout, the Arduino begins to normal operation. (Note that this mode is only required during the development of the PC based controller hardware. The iSOBOT control software should not enter it.)

During normal operation, ASCII messages are passed from software on the host PC to the Arduino microcontroller via this virtual RS-232 link. Firmware on the Arduino microcontroller receives these messages converts them from ASCII to binary and modulates the IR LED in order to send the corresponding bit pattern to the iSOBOT as described in the previous section.

The settings for this serial port are given as in Table 3.1:

SETTING	VALUE
BAUD RATE	38400
DATA BITS	8
STOP BITS	1
PARITY	NONE
HANDSHAKING	OFF

Table 3-1Serial Port Settings

Note that the host PC should disable the DTR hardware handshaking signal because it is used by the Arduino framework to allow the PC to reset the microcontroller.

The PC-to-Arduino protocol shall permit the Arduino board to be easily controlled via a terminal emulation program. The Arduino shall echo all received characters and should support the backspace operation.

The envisioned protocol consists of an ASCII string that contains either 6 (for 2 byte commands) or 8 (for 3 byte commands) hexadecimal characters.

Reception of a carriage return shall cause the Arduino board to initiate the process of checking the received string for validity, converting the ASCII characters to binary and transmitting the appropriate 2 or 3 command byte bit code via the IR LED.

The Arduino shall send an error message to the PC if the string isn't the right length or contains non-hexadecimal characters. The Arduino should transmit an appropriate user prompt such as '>' following every error message and every successful transmission.

iSobot Command Structure

Miles Moody speculates that the message can be partitioned into a header followed by two or three bytes. The following table shows the presumed command format:

Bit(s)	0	1-2	3-5	6-13	14-21	22-29 (Optional)
Description	CH	TYPE	CHECKSUM	CMD1	CMD2	CMD3

Table 3-2iSobot Command Structure

Where:

CH	0 = iSOBOT A 1 = iSOBOT B
TYPE	00 = 3 Byte Message (for robot arms control)

	01 = 2 Byte Message (all other commands)
CHECKSUM	Three bit checksum for data integrity. Calculated by summing the header and command bytes and then summing the result three bits at a time: s=hdr+cmd1+cmd2 (+cmd3) chksum=s[0..2]+s[3..5]+s[6..7]
CMD1 CMD2	Required command codes – See Appendix A
CMD3	Optional command byte (presumably contains servo setting for arms).

Table 3-3iSobot Command Structure Details

Note that a delay of 200ms is inserted between successive commands from the iSOBOT's hand held remote.

Motion Editor Controller Software

This sectionis about the 'motion editor', the controller software for iSobot built in Python language.The editor lets user pick the command and send through Arduino board in more sophisticated and user-friendly way replacing the iSobot remote controller.

Objectives

The objectives of the research on iSOBOT interface and editor were the following:

- Design iSobot Motion editor UI in Linux Environment
- Integrate iSobot UI with KHR1 Motion Editor
- Develop a serial communication between the Linux PC and the Arduino Board
- Develop a script editor that allows user to create or modify scripts of ISOBOT command codes.
- Create a PC-based remote control that sends command codes through IR to the ISOBOT.

Software

Used the following software for this design:

- Python 2.5, PyQt 4.0 and QtDesigner
- Arduino USB-SERIAL driver

The iSobot motion editor is implemented in Python and PyQt which is a Python binding of the cross-platform graphical user interface toolkit Qt. The design and layout of motion editor form is actually accomplished by QtDesignerwhich is a graphical tool that simplifies the design of the user interface by enabling the drag-and-drop of control buttons onto the form. Chosen Python as the base language in the implementation of iSobot motion editor because the editor needed to be integrated with KHR1 motion editor that extends from MathiasSunardi motion editor's program[39], which was previously developed under Python. The advantage of using Python is the ability to re-use the source codes written by MathiasSunardi& a quick start up.

iSobot and PC Communications

The communication between the iSobot and the PC is through the Arduino board which receives binary command codes from the PC and sends the command codes as IR signals to the iSobot. The Arduino board has on board USB-to-serial chip (FTDI)which provides a virtual COM port. After the driver of the FTDI is installed on the PC, the PC and the Arduino's microcontroller can communicate serially as if they are connected by an RS-232 serial link.

During communication, iSobot command codes are treated as ASCII messages, which are passed from software on the host PC to the Arduino microcontroller via the virtual RS-232 link. Firmware on the Arduino microcontroller receives these messages converts them from ASCII to binary and modulates the IR LED to send the signals to the ISOBOT.

iSobot Commands Structures

iSobot commands consist of 22-bits or 30-bits commands. This iSobot motion editor project only interprets and processes 22-bit commands. The hacking 30-bit commands are fairly complex. A table of 22-bit commands is included in Appendix A.

When a user wants to execute a motion on the editor such as the “walk forward” motion, the corresponding 22-bit command code “0x0DB703” is retrieved from the command database and sent as one ASCII character at a time to the serial port, and ended with a

‘\r’, a carriage return, which indicates the end of message. Arduino Microcontroller receives ASCII Hex codes convert to binary, sends bit patterns to ISOBOT through IR. Following is an example of the executions steps of the “walk forward” command.

Example:

Command to send: “walk forward”

Command code: 0x0DB703

Device Setup: Port = /dev/ttyUSB0, baud rate = 38400, Data Bit =8, Parity=None

1. Serial.Open(Port, baudrate, DataBit, Parity)
2. Serial.Write(“0”)
3. Serial.Write(“D”)
4. Serial.Write(“B”)
5. Serial.Write(“7”)
6. Serial.Write (“0”)
7. Serial.Write (“3”)
8. Serial.Write(“\r”)
9. Serial.Close()

Since the Arduino board handles the conversion of ASCII characters to binary data as well as all the IR communication protocol, the execution of an iSobot command from the PC is fairly simple. Notice that when executing a group of commands in sequence, a time

delay (1 to 5 seconds) must be specified between the commands, so that the iSobot has enough time to complete the motion before moving to the next one. Otherwise, only one command may be executed by the iSobot from the sequence of commands because the other command signals will be lost.

Motion Editor Design Model

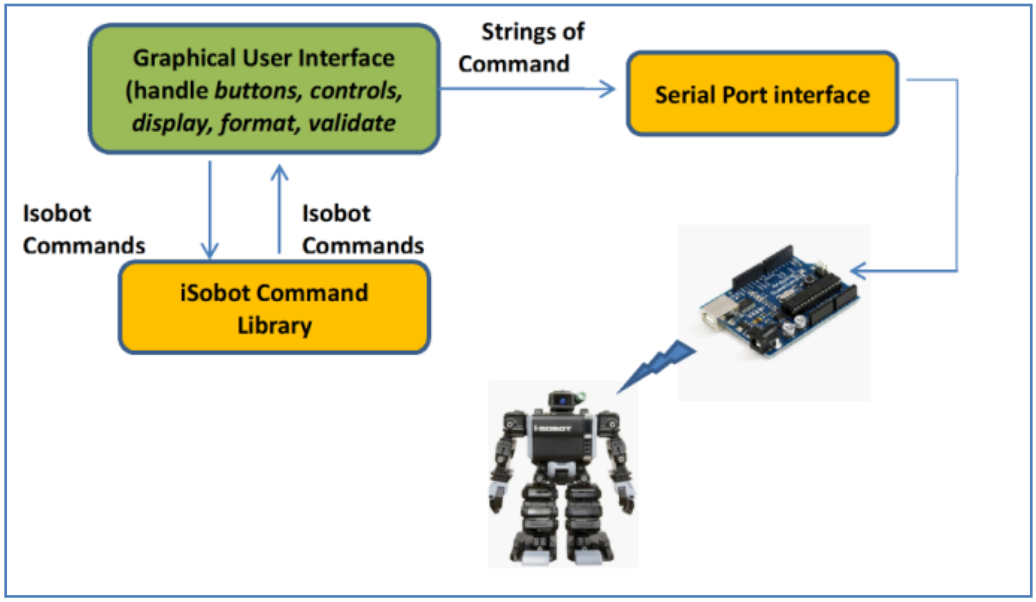


Figure 3:5Design Model of iSobot Motion Editor in Python

The implementation of the iSobot motion Editor consists of the following main file modules:

1. **iSobot Command Library**

   This module stores the 22-bit iSobot command in a dictionary (hash table) key-values pair format. The descriptions of the motions are stored as keys while their

corresponding codes are stored as values. When a user specifies a motion name from the user interface, the command code are retrieved from the dictionary and sent to the serial ports as ASCII characters.

2. Serial Port Interface

This module handles the serial communication between the PC and the Arduino board. It consists of the following functions:

1. Open_serial(): open a serial port connection
2. Close_serial(): close the serial port connection
3. Send_command(): send command codes are ascii characters through the serial ports to the Arduino board
4. FlushInput_serial(): flush the input buffer of the serial ports
5. FlushOutput_serial(): flush the output buffer of the serial ports

3. **Graphical User Interface**

This module handles all the signals and events generated from the user interface controls. It integrates all the KHR1 and iSobot modules to create a universal motion editor.

iSobot Motion Editor User Interface

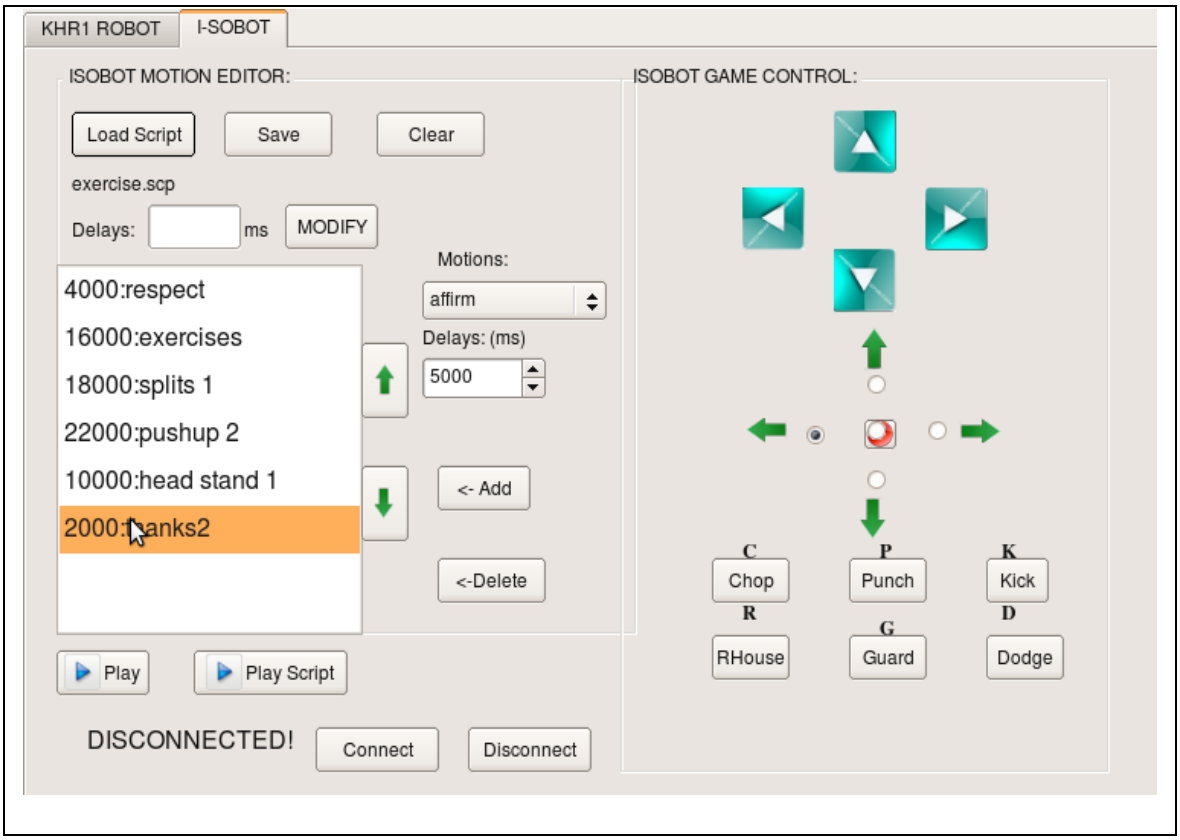


Figure 3:6iSobot Motion Editor Interface

ISOBOT Motion Editor Features

1. Open a script file of iSobot motions and display the motions on the editor list
2. Create new group of motions by adding motions from an existing list of iSobot motions from the ComboBox control to the editor list.
3. Modify a group iSobot motions and their time delays and save them to a script file.
4. Remove an iSobot motion from the script editor list.
5. Move motion in the script editor up and down to switch their order of executions.
6. Play an iSobot motion from the editor by sending a command code as IR signal to iSobot.

7. Play a group of iSobot motions from the script by sending IR commands to iSobot.
8. Include PC-based remote control that sends martial arts motions through IR to the iSobot.
9. The PC-based remote control enables the iSobot to walk forward, walk backward, turn left, turn right, kick, chop, punch, guard, and dodge.

PC-based Remote Control Features

There are more than a hundred action commands which iSobot can perform. Hence, creating individual buttons for all action command would not be a good idea. We selected few commands which we would like the robot to perform more frequently and on the fly when we click it's action button.

For e.g.: kick, punch, chop, RHouse, Guard, dodge.

ISobot remote controller can perform the above actions with either hands or legs. Hence we kept 4 direction toggle buttons: left, right, forward, backward. For e.g. if left direction toggle is selected & user hits the kick button the iSobot kicks the left leg. In this way, the 6 pushbuttons & 4 direction buttons cover 24 actions.

Moving the iSobot forward, backward, left & right is also done handy by providing special action buttons for them.

Buttons	Actions
Load script	• Opens a box of folder where previously saved scripts are stored. • Let the user select the saved script file. (file format: .scp) • After hitting “open” button, loads the set of actions
Save	Let the user save the set of actions.
Clear	Clears all the actions in the display.
Motion	This is list of commands. When user clicks on the list, the drop box open up showing hundreds of action modes among which user can select one.
Delay	Every action has some delay by default. Different actions have different time lengths. This provision of this delay button, user can increase or decrease the default delay period.
Add	When user selects the motion from the motions list, the add button is takes that selected action and puts in the display.
Play	Plays the single highlighted action commands for given delay.
Play Script	Play all the actions in the display sequentially.
Up Arrow	Swaps the highlighted action with action above the selected one.
Down Arrow	Swaps the highlighted action with action below the selected one.
Connect	Opens the serial communication port to connect Arduino board through USB.
Disconnect	Close the connection to Arduino board.

Table 3-4iSobot Motion Editor GUI Controls

IMPROVEMENTS for this editor:

The iSobot motion editor has a few drawbacks and still needs improvements. One drawback of the motion editor is that the user interface can control only one iSobot. The user interface can be improved to control multiple iSobot. This can be accomplished by modifying the source codes to work in a threading environment. In a threading environment, each iSobot will have its own process and runs separately in each own thread. Assigning each iSobot a separate running thread also solves the problem of the user interface being temporary locked up when a executing a sequence of commands. Due to the time delays being inserted between the commands, the user interface does not response to user inputs (because the system is put into sleep) when the script of command is executing. If the codes that execute the command codes are turned into a thread program, this problem will not occur.

In the improved motion editor all above drawbacks are taken care of. There are much more additional functionalities as well. All those are discussed in chapter 6.

Chapter 4 Adding Sound With Sound Synthesizer

It's fun to watch a robot playing, walking, moving, dancing, and expressing numerous of its skills, especially those with its own sound or with some music. This particular iSobot has got its voice for almost all of his actions but not all robots are capable to produce sound. In such case, even the great mechanical work for that humanoid robot may also fail to convey the right expression through just playing action itself. Rather, the viewers are sometimes puzzled thinking what actually robot did?

A speech announcement prior to playing action may give some idea to viewer, and sometime set their mind to see the announced action play and when they actually watch the robot action, they quickly recognize it and such action gets appreciation rather than a big question mark on viewers' faces. Hence addition of sound along with robot action play makes it livelier.

Some of the robotic actions are easy to recognize, for eg a robot waiving his hands. It clearly indicates he wants to greet / say Hi. But what if he bends down, moves his head twice and then gets back straight? Japanese may interpret as robot greeting in their style, giving respect; whereas Americans know that robot seems frustrated. A simple speech announcement in the background saying "Oh No! I'm so annoyed" would more clear his action.

Such speech announcement can be recorded and played as music. While another better approach is to use speech synthesis technique which converts text strings and speaks those in humanoid voice. Hence, this chapter is targeted to detail about such speech synthesis chip "SpeakJet" and its hardware implementation.

Later part of the chapter explains alternative solution, it implemented in the software.

SpeakJet Interface Project

Aim

To add voice to a mute robot using Speech Synthesis chip.

Introduction

This is to describe how to interface a SpeakJet[22] with a MAX232 chip and LM381 Audio Power Amplifier chip to generate speech or sounds using the Speakjet chip. The Speakjet is a completely self contained, single chip voice and complex sound synthesizer. The SpeakJet has a build-in library of 72 speech elements (allophones), 43 sound effects, and 12 DTMF touch tones. The SpeakJet can be controlled by a serial data line from a MAX232 chip. Other features include an internal 64 byte input buffer, internal programmable EEPROM, three programmable outputs, and direct user access to the internal five channel sound synthesizer. There are other more capable speech synthesizers in the market but they are much more expensive and are restricted to speech only. The SpeakJet represents a reasonable compromise of voice/sound function and price

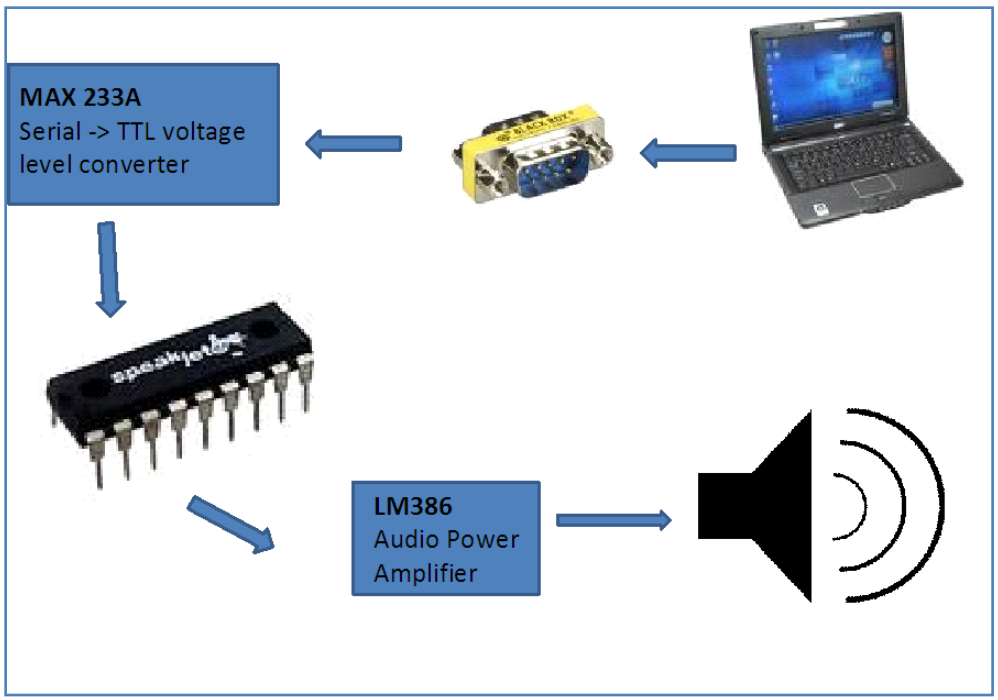


Figure 4:1SpeakJet Interface Configuration Design

SpeakJet Core Features

1. DTMF and other sound effects.
2. Programmable control of pitch, rate, bend and volume.
3. Programmable, 5 channel synthesizer
4. Natural phonetic speech synthesis.
5. Programmable power – up or reset announcements.
6. Multiple modes of operation
7. Simple interface to microcontrollers.
8. Simple “Stand Alone” operation … {Which is used in this project}
9. Three programmable digital outputs.
10. Internal 64 byte input buffer.

11. Internal programmable EEPROM.
12. Extremely low power consumption.

Special Features of SpeakJet

1. An internal clock oscillator provides for a truly self contained sound system. Simply connect the SpeakJet to a power supply and a speaker to hear it speak.
2. AN internal user programmable EEPROM allows for programming of up to 16 complex phrases or sound sequences. These may be played back once or looped many times in response to events.
3. Three multipurpose, programmable digital outputs allow the SpeakJet to control external devices based on timing of the sound output. Control of devices may include lights, motors, or even launch model rocket after a count down sequence.
4. Phrases may call other phrases, sounds or controls, with nesting up to 3 deep.
5. No special equipment is required to program the internal EEPROM, only a serial connection is required.

Hardware for SpeakJet

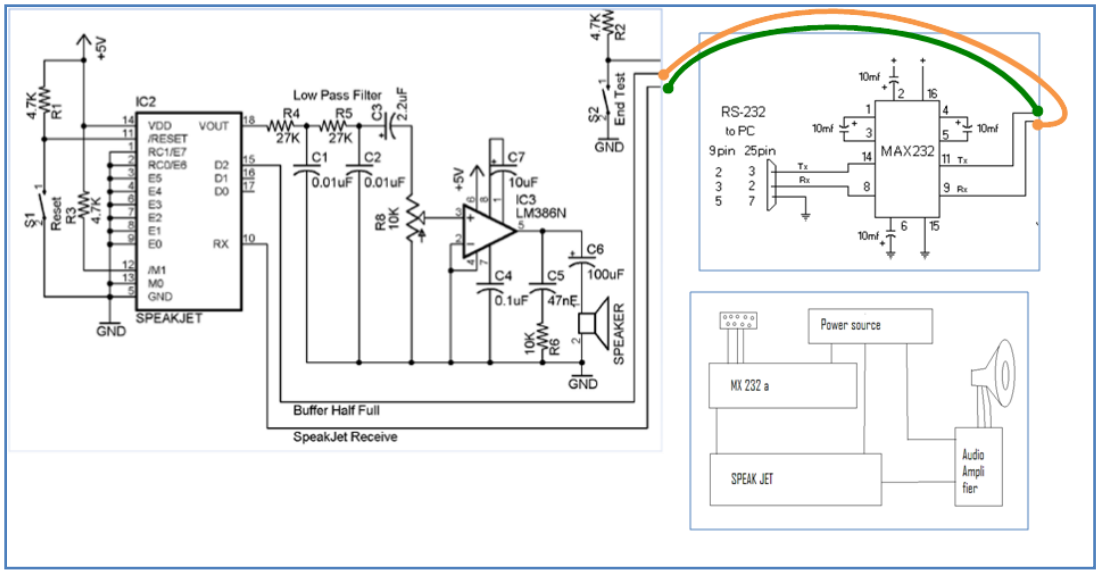


Figure 4:2SpeakJet Hardware Connection Circuit Diagram

Circuit Assembly

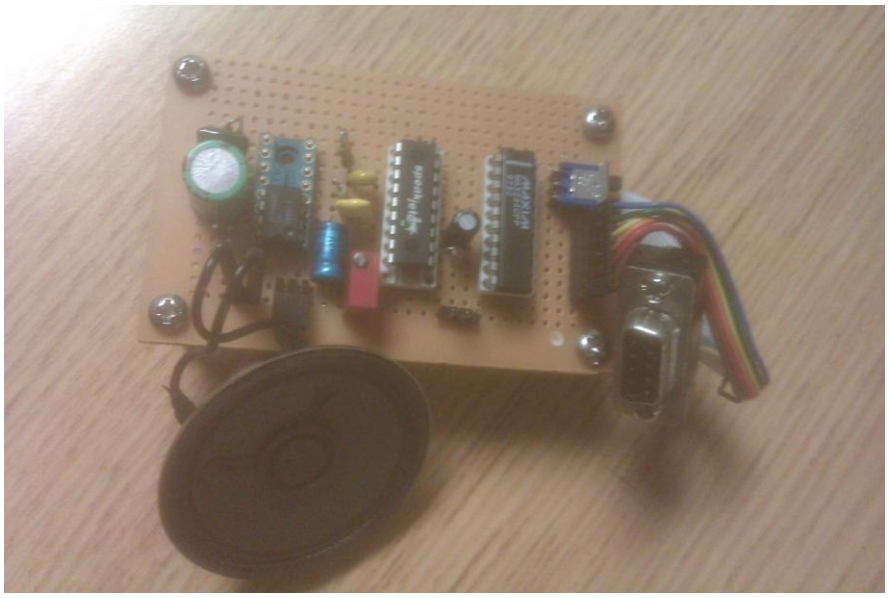


Figure 4:3SpeakJet Hardware built on Dot Matrix board

More details about the SpeakJet can be found on the site[23], yet Figure 4.3 is the main architectural diagram of SpeakJet followed by a brief description to understand it's working.

SpeakJet Working

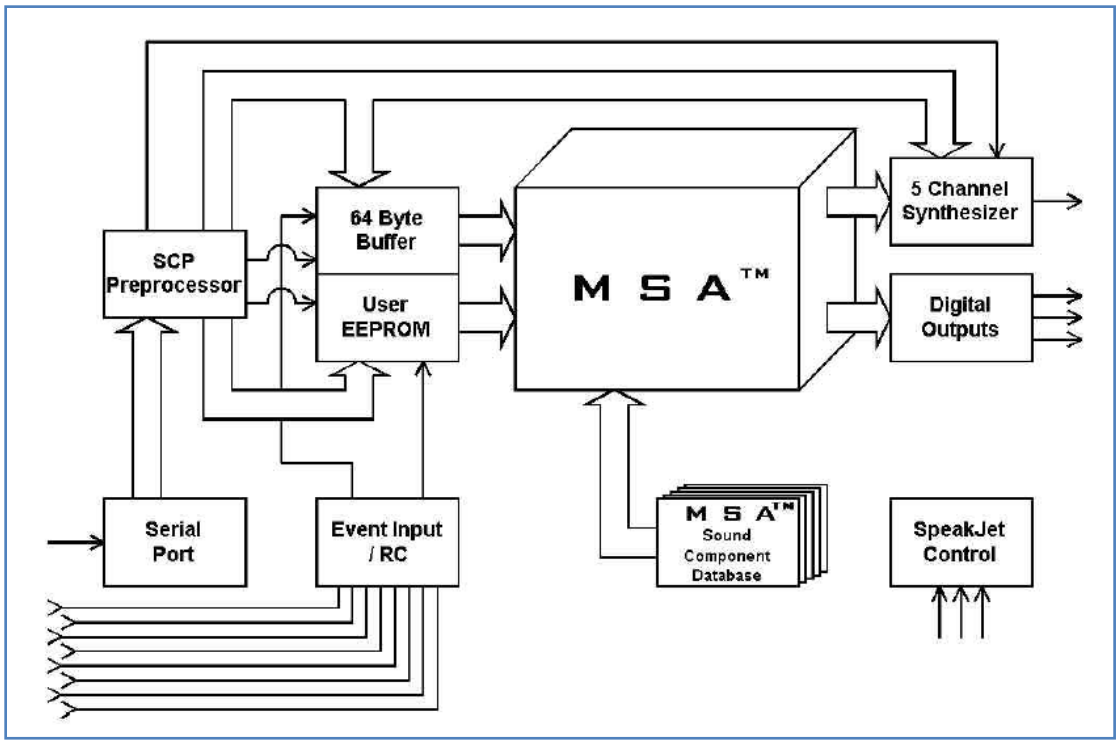


Figure 4:4Block Diagram of Internal Architecture of SpeakJect

Description of each block in the Figure 4.4:

Serial Port

Serial data is the main method of communicating with the SpeakJet to execute commands or create voice and sounds. The serial data can also be used to program internal EEPROM.

Serial COM specification:

- Baud rate : 9600
- Bits : 8
- Stop bit : 1
- Parity : N
- Flow : Hardware

SCP {Serial Control Protocol}

SCP fetches the serial input commands and fed to either of three:

1. Internal buffer of 64 byte
2. EEPROM
3. 5 channel synthesizer

Internal buffer

This buffer stores upto 64 bytes of input serial commands.

Since the working speed of SpeakJet is slower than that of PC, all commands do not get executed on the flow. There is delay for execution of each command, since it is the time taken for speaking the text. Hence, this slower device works with its own speed storing upcoming 63 bytes from faster device.

When this buffer gets half full with 32 bytes, it raises a flag, which is nothing but high signal on "buffer half full" line. This line can be used in serial connection as the serial port's CTS line. A logical low output indicates that the buffer can accept 32 bytes and a logical high output indicates that the buffer cannot.

EEPROM

256 byte internal EEPROM allows to store 16 phrases / sounds. With a set of 16 registers, those can be accessed, over-written or fetched values from.

EEPROM access can also be manipulated for 8 event-based interrupts. In this project, they are all connected to ground.

5 channel synthesizer

The module, as shown in the figure4.5, is responsible for production of SpeakJet's voice. It is comprised of 6 oscillators, 5 mixers and a Pulse Width Modulated (PWM) digital output.

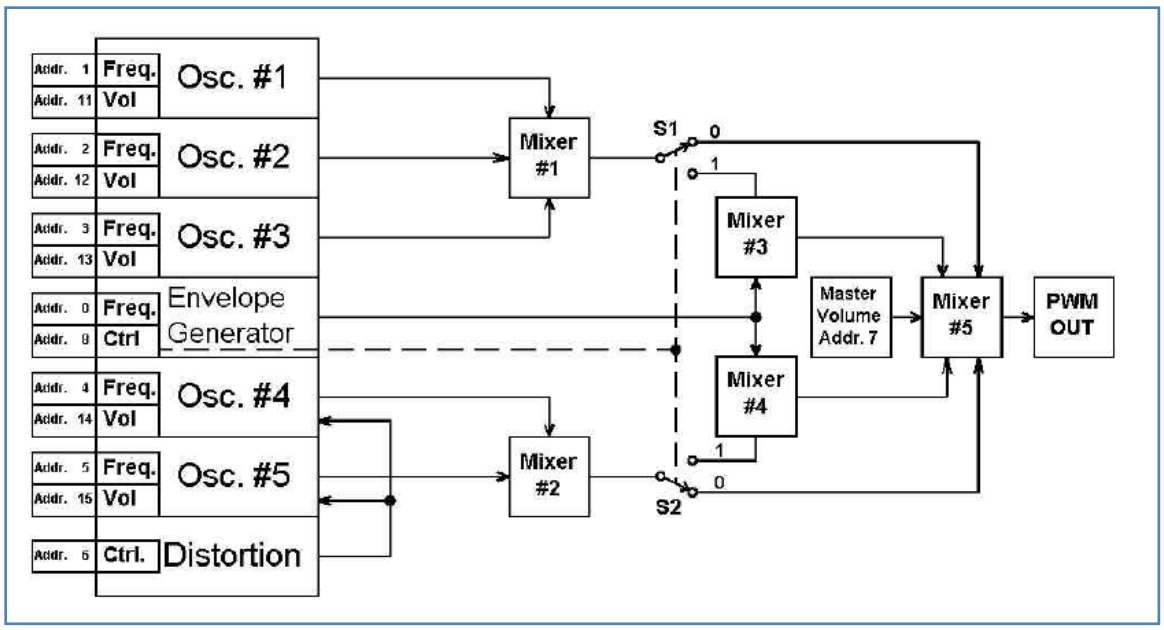


Figure 4:5 Five Channel Synthesizer

MSA {Mathematical Sound Architecture} & Sound Database

MSA engine executes the commands stored in the buffer using its sound database and directs them to 5 channel synthesizer to produce voice and sound effects output.

MSA sound components will manipulate the synthesizer's register according to its own needs. After the MSA engine is finished executing the commands in the input buffer, the values of the synthesizer's register are left in their last used state. The auto silence option determines whether or not a pause is played after the input buffer is empty.

Example below explains how it works:

Hello = \HE \FAST \EHLE \LO \OWWW

To speak the word "Hello", user has to send commands as:

\HE \FAST \EHLE \LO \OWWW

These are nothing but parameterized string of hex codes. Hex code value for the same is:

183, 7, 159, 146, 164

\HE	183
\FAST	7
\EHLE	159
\LO	146
\OWWW	164

These codes are stored in the input buffer. MSA, when executes these commands one by one, it uses its sound database to generate set of volume and frequency values to be fed to 5 channel synthesizer for single input command. This inbuilt MSA architecture is fixed and not user programmable. The 5 channel synthesizer further converts it to PWM to produce sound.

User has a way to directly program this 5 channel synthesizer to produce phoneme, micro parts of the speech. This could be helpful to produce words in the different languages, and for the grammar knowledge that language is also necessary.

Logic and algorithms

I have created a simple function “speak(char * string)”, which works just like function “printf(char * string)” in C. printf prints the string inside bracket on output window, whereas, speak function speaks it, provided: the word should be present in dictionary.

Speak(“string to be spoken”);

1. We first created a dictionary of limited words we require for speech.
2. Created a structure of dictionary which has 2 elements: (A) word (B) code

   Code is nothing but the array of hex codes to be sent serially to SpeakJet chip which has inbuilt 5 channel synthesizer to convert these codes to the Pulse Width Modulated [PWM] signal ievoice.

3. Written a function “speak(char * string)” which passes text string to be spoken.

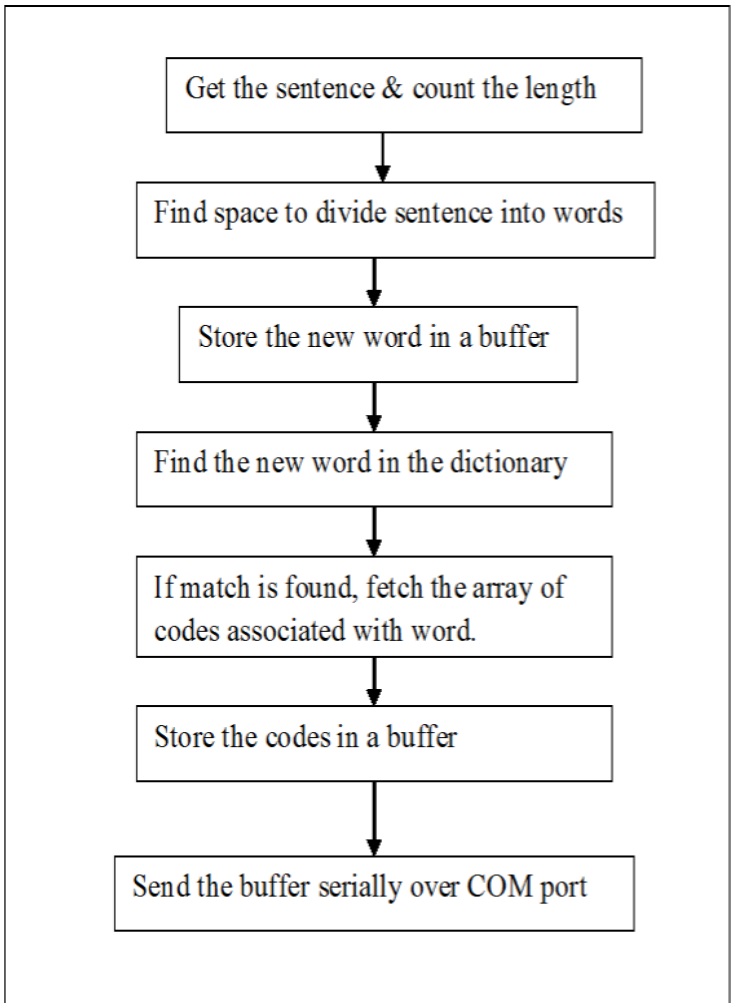


Figure 4:6Algorithm for text to speech synthesis

The speakjet comes with its own software to create new words in the dictionary.

The software already has a decent word dictionary, containing around 5000 most commonly occurring English words. User can use these readily available words and quickly create very many sentences. Then, following the same algorithm for 'Speak("your sentence")' , the human voice can be generated.

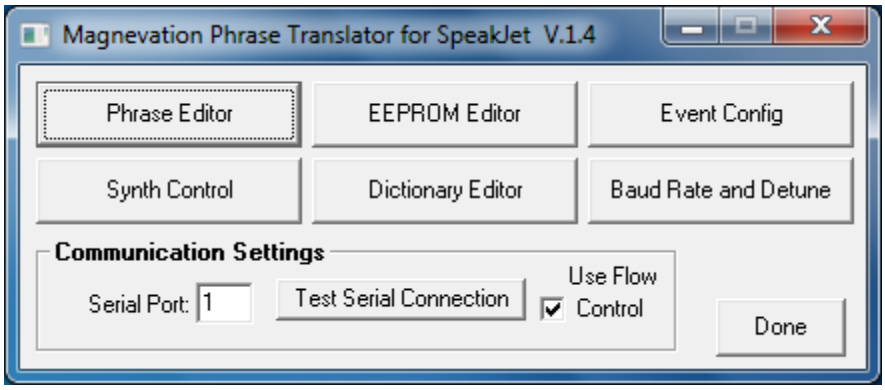


Figure 4:7Speech Synthesis software for SpeakJet

Although, user can produce meaningful text sentences using available dictionary words, he has very many limitations. Dictionary doesn't contain\names of the person which is one of the most common things in human speech / talks. For e.g. the dictionary does not contain the words like Tom, Jerry. Hence while speaking the sentence: 'Tom seems angry at Jerry, and running behind him to punish'; you listen the sound output as

' - -- seems angry at ---, and running behind him to punish'.

Are you puzzled? At least I'm.

Hence there is need that user can create own words, meaning produce the right pronunciation for those. The software avails another editor shown in Figure4.2 which does the same job.

E.g. To create a speech sound like 'Hello' pronounced, user has to select following parts of the speech in the right sequence.

Hello = H + eh + lu + oh

Sound	Speech part from Editor	Command code
H	\HE	183
Eh	\EHLE	159
Lu	\LO	146
Oh	\OWWW	164

Table 4-1Speech Edition for word 'Hello'

After given the first shot, user realizes the word sound too crude. So there is need of adjusting sound tempo / pauses. If the pause between \H and \EHLE is quickened than normal, then the pronunciation is little better.

Sound	Speech part from Editor	Command code
H	\HE	183
Quick	\FAST	7
Eh	\EHLE	159
Lu	\LO	146
Oh	\OWWW	164

Table 4-2Better version of word 'Hello'

Once, user is satisfied with this pronunciation, he can save this newly edited word in the dictionary so as to use in the future.

The third column shows the command code to be sent serially to the SpeakJet assembled hardware.

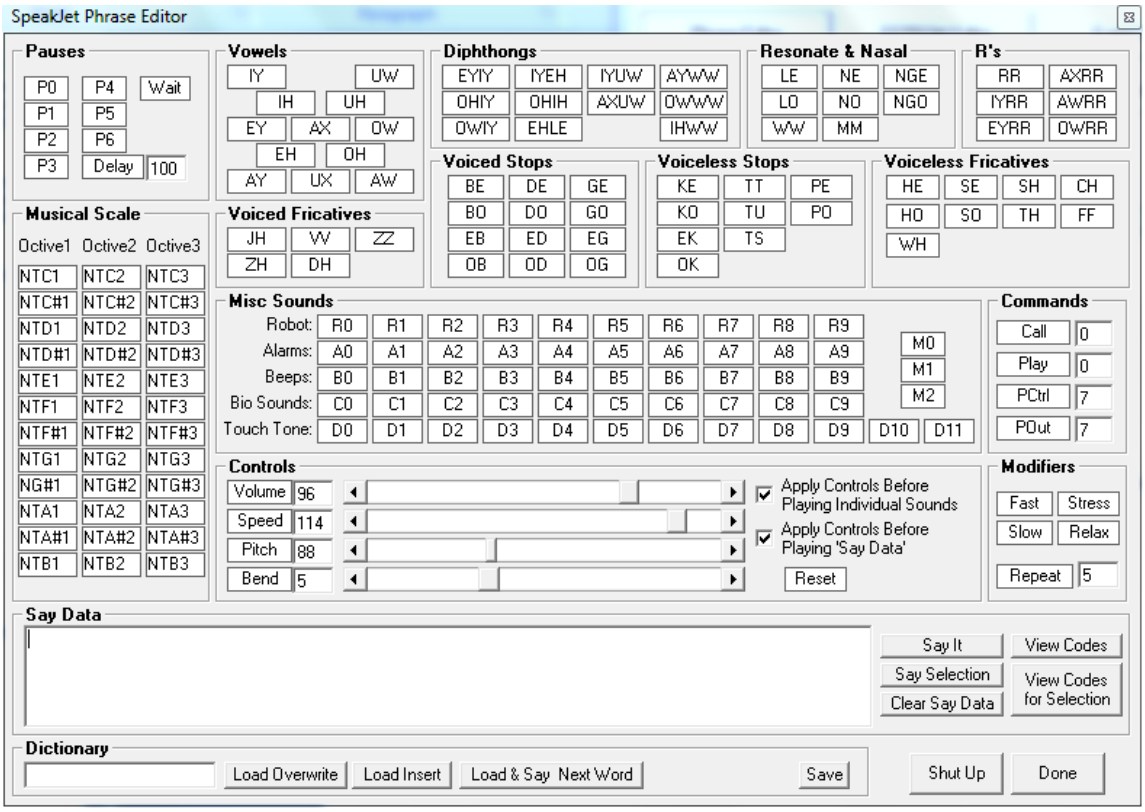


Table 4-3Word Pronunciation Editor for SpeakJet

Yes, it may quickly become boring to spend time in trial and error; just for creating one word, that too with such complex user interface of software.

Microsoft's Speech SDK

Text To Speech Synthesizer

Speech synthesis library is now very commonly available in all operating systems like windows, Macintosh, Ubuntu and even smartphone's OS like iOS (for iPhone), Android. Hence, we are using OS inbuilt speech synthesis library provided by Windows SDK.

For the mute KHR1, we had added external speech synthesizer circuit for its speech.

The new Editor uses window's inbuilt 'Sapi.dll' (Speech API), which has all headers not only for converting Text to Speech but also for speech recognition.

Simple Interface:

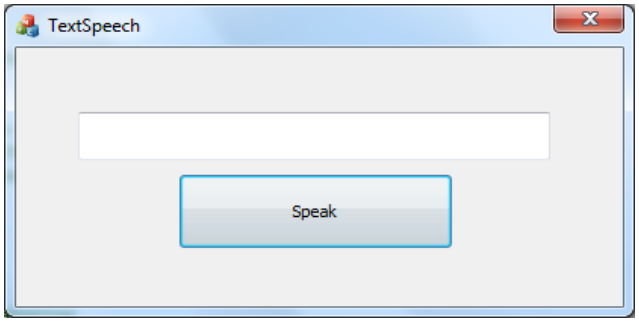


Table 4-4Basic GUI for editing text and listening it by clicking a button

The string entered in text box will be spoken after hitting the "Speak" button. This lets the user test the speech synthesis.

Important header file to be included to use speech synthesis methods:

#include<sapi.h>

The SAPI application programming interface (API) dramatically reduces the code overhead required for an application to use speech recognition and text-to-speech, making speech technology more accessible and robust for a wide range of applications.

API for Text-to-Speech

Applications can control text-to-speech (TTS) using the ISpVoice Component Object Model (COM) interface. Once an application has created an ISpVoice object (see Text-to-Speech Tutorial), the application only needs to call ISpVoice::Speak to generate speech output from some text data. In addition, the IspVoice interface also provides several methods for changing voice and synthesis properties such as speaking rate ISpVoice::SetRate, output volume ISpVoice::SetVolume and changing the current speaking voice ISpVoice::SetVoice

Special SAPI controls can also be inserted along with the input text to change real-time synthesis properties like voice, pitch, word emphasis, speaking rate and volume. This synthesis markup sapi.xsd, using standard XML format, is a simple but powerful way to customize the TTS speech, independent of the specific engine or voice currently in use.

The IspVoice::Speak method can operate either synchronously (return only when completely finished speaking) or asynchronously (return immediately and speak as a background process). When speaking asynchronously (SPF_ASYNC), real-time status information such as speaking state and current text location can polled usingISpVoice::GetStatus. Also while speaking asynchronously, new text can be spoken

by either immediately interrupting the current output (SPF_PURGEBEFORESPEAK), or by automatically appending the new text to the end of the current output.

In addition to the ISpVoice interface, SAPI also provides many utility COM interfaces for the more advanced TTS applications.

Events

SAPI communicates with applications by sending events using standard callback mechanisms (Window Message, callback proc or Win32 Event). For TTS, events are mostly used for synchronizing to the output speech. Applications can sync to real-time actions as they occur such as word boundaries, phoneme or viseme (mouth animation) boundaries or application custom bookmarks. Applications can initialize and handle these real-time events using ISpNotifySource, ISpNotifySink, ISpNotifyTranslator, ISpEventSink, ISpEventSource, and ISpNotifyCallback.

Lexicons

Applications can provide custom word pronunciations for speech synthesis engines using methods provided by ISpContainerLexicon, ISpLexicon andISpPhoneConverter.

The Only method added in the project is:

Declaration :

void speak(CStringTextToSpeak);

Definition:

```
// Speaks the input String
void CTextSpeechDlg::speak(CStringTextToSpeak)
{
        ISpVoice * pVoice = NULL;
        if (FAILED(::CoInitialize(NULL)))
return;
HRESULT hr = CoCreateInstance(CLSID_SpVoice, NULL, CLSCTX_ALL,
IID_ISpVoice, (void **)&pVoice);
if( SUCCEEDED( hr ) )
        {
                hr = pVoice->Speak(TextToSpeak, 0, NULL);
pVoice->Release();
pVoice = NULL;
        }
::CoUninitialize();
return;
}
```

Chapter 5 Controlling Stage Lights with MIDI Sequencer

A theater without light and music is like a well without water.

Theatrical lighting is certainly the next step when compared to the home lighting or any other lighting system. The purposes of theatrical lighting systems is not just lighting the theater with illumination but also to capture the attention of viewers or focus / divert their point of sight on the stage. Especially, dancing lights plays great role on stage by letting the show to be not monotonous. Lighting adds excitement, communicates mood, emotions when synchronized with sounds (like storm) or music.

A light control could be as basic as a binary control (on &off) or a little complicated with fuzzy level sync. Fuzzy level means, it has more than 2 states. For e.g. it's intensity could be of 5 different levels as: 0%, 25%, 50%, 75% & 100%. If the light intensity level is synced with music, it makes more meaningful presentation. Chameleon allows for synchronized lighting. Chameleon synchronizes music to lights in real time without the need for costly and time-consuming custom programming. Chameleon is a great source of entertainment for crowds waiting for entrance to your haunt.

Musical Light Controller: Chameleon

Figure 5:1Musical Light Controller: Chameleon

Power Model Unit (PMU) …

Figure 5:2Power Management Unit for Chameleon Musical Light Controller

User guide for chameleon is attached in the appendix F.

Synchronized Lighting Controller Chameleon Needs Music Source.

Chameleon is a highly intelligent (microprocessor-based) lighting controller that creates a professional light show without the need for expensive pre-programmed shows or time-consuming custom programming. All it needs is a music source and so can be generated using MIDI easily. Hence this chapter explains in brief the main information about MIDI: why, what, how?

MIDI

Why do I need MIDI?

Generation of sound is extremely easy with MIDI, and simple mouse click can be used as events for playing music.

What is MIDI?

MIDI (short for **M**usical **I**nstrument **D**igital **I**nterface) is a music industry standard communications protocol that lets MIDI instruments and sequencers (or computers running sequencer software) talk to each other to play and record music. Most of the music we hear every day is written with and played by MIDI sequencers.

MIDI is:

- **Compact** - Hours of music can fit on a single 3 1/2" floppy disk, thousands of songs on a CD
- **Efficient** - Just about any computer can handle it
- **Powerful** - A whole orchestra is at your command
- **Versatile** - A click of a button is all it takes to change key, tempo, instrument, etc.
- **Intuitive** - a MIDI file is just an electronic version of a player piano roll for many instruments
- **Portable Industry Standard** - Any MIDI instrument can talk to any other

At minimum, a MIDI representation of a sound includes values for the note's pitch, length, and volume. It can also include additional characteristics, such as attack and delay time.

The advantages of MIDI

There are two main advantages of MIDI -- it's an easily edited/manipulated form of data, and also it is a compact form of data (ie, produces relatively small data files).

Because MIDI is a digital signal, it is very easy to interface electronic instruments to computers, and then to equipment with that MIDI data on the computer with software.For example, software can store MIDI messages to the computer's disk drive. Also, the software can playback MIDI messages upon all 16 channels with the same rhythms as the human who originally caused the instrument(s) to generate those messages. So, a musician can digitally record his musical performance and store it on the computer (to be played back by the computer). He does this not by digitizing the actual audio coming out of all of his electronic instruments, but rather by "recording" the MIDI OUT (i.e., those MIDI messages) of all of his instruments. The MIDI messages for all of those instruments go over one run of cables, so if you put the computer at the end, it "hears" the messages from all instruments over just one incoming cable. The great advantage of MIDI is that the "notes" and other musical actions, such as moving the pitch wheel, pressing the sustain pedal, etc., are all still separated by messages on different channels. So the musician can store the messages generated by many instruments in one file, and yet the messages can be easily pulled apart on a per instrument basis because each

instrument's MIDI messages are on a different MIDI channel. In other words, when using MIDI, a musician never loses control over every single individual action that he made upon each instrument, from playing a particular note at a particular point, to pushing the sustain pedal at a certain time, etc. The data is all there, but itis combinedtogether in such a way that every single musical action can be easily examined and edited.

Contrast this with digitizing the audio output of all of those electronic instruments. If you've got a system that has 16 stereo digital audio tracks, then you can keep each instrument's output separate. But, if you have only 2 digital audio tracks (typically), then you've got to mix the audio signals together before you digitize them. Those instruments' audio outputs don't produce digital signals. They're analog. Once you mix the analog signals together, it would take massive amounts of computation to later filter out separate instruments, and the process would undoubtedly be far from perfect. So ultimately, one lose control over each instrument's output, and if you want to edit a certain note of one instrument's part, that's even less feasible.

How does MIDI work?

MIDI Data Format

The majority of MIDI communication consists of multi-byte packets beginning with a **status byte** followed by one or two **data bytes**.Bytes are packets of 8 bits (0's or 1's). Status bytes begin with a '1' e.g. 1xxx xxxx--this is call 'set.' Data bytes begin with a '0'

e.g. 0xxx xxxx--this is called 'reset.' Each byte is surrounded by a start bit and a stop bit, making each packet 10 bits long. Messages fall into the following five formats:

Channel Voice

Control the instrument's 16 voices (timbres, patches), plays notes, sends controller data, etc.

Channel Mode

Define instrument's response to Voice messages, sent over instrument's 'basic' channel

System Common

Messages intended to all networked instruments and devices

System Real-Time

Intended for all networked instruments and devices. Contain only status bytes and is used for synchronization of all devices. essentially a timing clock

System Exclusive

Originally used for manufacturer-specific codes, such as editor/librarians, has been expanded to include MIDI Time Code, MIDI Sample Dump Standard and MIDI Machine Control

Channel Voice Messages

The most common MIDI messages are **channel voice messages** listed in the chart below. They convey information about whether to turn a note on or off, what patch to change to, how much key pressure to exert (called *aftertouch),* etc.

A MIDI channel voice message consists of a statusByte followed by one or two dataBytes**.**

Status Byte	**Data Byte 1**	**Data Byte 2**	**Message**	**Legend**
1000nnnn	0kkkkkkk	0vvvvvvv	Note Off	n=channel* k=key # 0-127(60=middle C) v=velocity (0-127)
1001nnnn	0kkkkkkk	0vvvvvvv	Note On	n=channel k=key # 0-127(60=middle C) v=velocity (0-127)
1010nnnn	0kkkkkkk	0ppppppp	Poly Key Pressure	n=channel k=key # 0-127(60=middle C) p=pressure (0-127)
1011nnnn	0ccccccc	0vvvvvvv	Controller Change	n=channelc=controller v=controllervalue(0-127)
1100nnnn	0ppppppp	[none]	Program Change	n=channel p=preset number (0-127)
1101nnnn	0ppppppp	[none]	Channel	n=channel p=pressure (0-127)

			Pressure	
1110nnnn	0fffffff	0ccccccc	Pitch Bend	n=channel c=coarse f=fine (c+f = 14-bit resolution)

Table 5-1MIDI Message structure

A sample message for turning on a note (middle C) on MIDI channel #5 very loudly (with a velocity or force of 127, the maximum) is shown below in binary.

status byte	data byte	data byte
10010100	00111100	01111111

Table 5-2MIDI Message format to be sent

The first four bits of the status byte (1001) tell MIDI that the following message is a note-on command, while the last four bits tell MIDI what MIDI channel the message is for (0000=MIDI channel #1, 1111=MIDI channel #16). Note that the channel number are offset by one value, since channel 1 is set by binary '0' and channel 16 is set by binary '15.' The first data byte tells MIDI what note to play (decimal 60=middle C), while the second data byte tells MIDI how loud to play the note. In this case the maximum velocity of 127 is sent. The note will sound until a message to turn off the same note number is received.

Standard MIDI Key Assignments

The standard MIDI key assignments for percussion instruments are based on the MMA General MIDI Mode specification. The following illustration shows the standard key assignments for MIDI files authored for Windows.

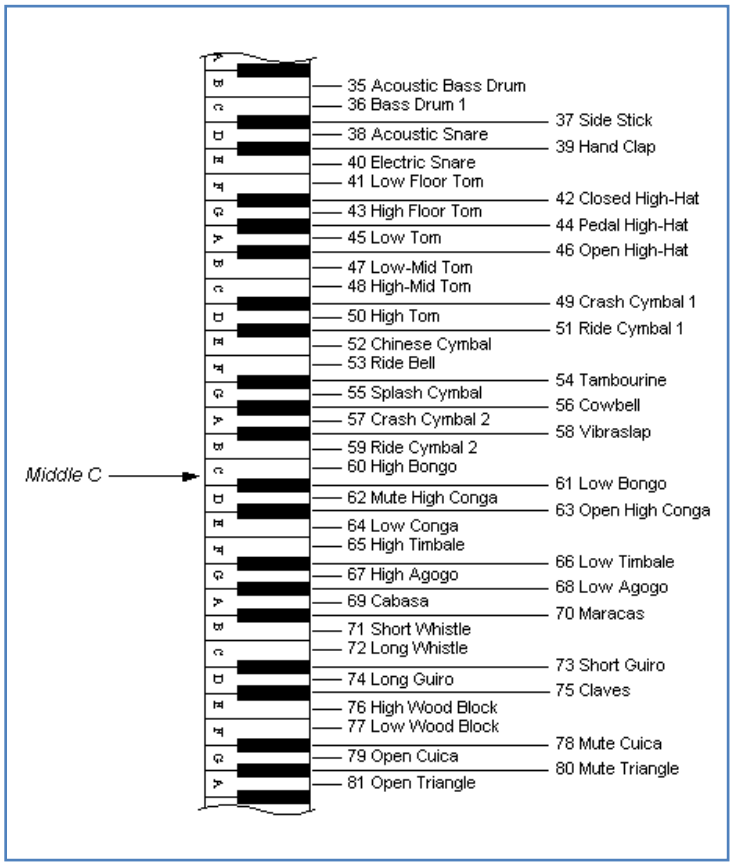


Figure 5:3Standard MIDI Key Assignment

Using midiOutShortMsg to Send Individual MIDI Messages

The following example uses the midiOutShortMsgfunction to send a specified MIDI event to a given MIDI output device:

```
UINT sendMIDIEvent(HMIDIOUT hmo, BYTE bStatus, BYTE bData1,
```

```
    BYTE bData2)
{
union {
        DWORD dwData;
        BYTE bData[4];
    } u;

    // Construct the MIDI message.

u.bData[0] = bStatus;  // MIDI status byte
u.bData[1] = bData1;   // first MIDI data byte
u.bData[2] = bData2;   // second MIDI data byte
u.bData[3] = 0;

    // Send the message.
returnmidiOutShortMsg(hmo, u.dwData);
}
```

Interface in the Editor

In order to play the different scales, and let the user generate interesting sound through the editor, an easy interface is necessary. There are numerous ideas; I came up with an idea explained below.

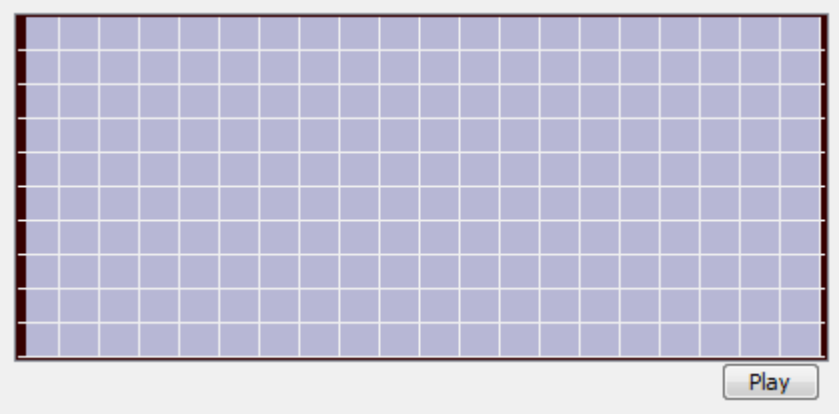


Figure 5:4Power Management Unit for Chameleon Musical Light Controller

The above diagram shows matrix of 10 rows X 20 columns. With the mouse click user can select certain nodes of matrix as shown below. He gets response (listen) for clicked note too.

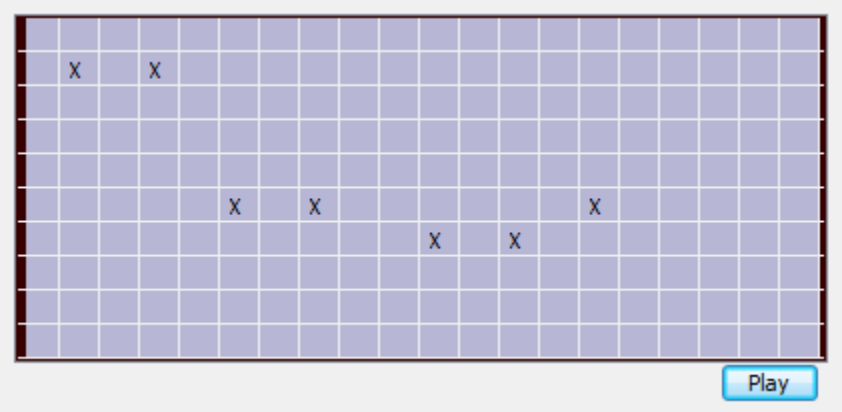


Figure 5:5With Mouse clicks, user can tick particular nodes in composer and create melodies quickly

The composed melody in Figure5.5 plays first line of "Twinkle Twinkle little Star".

Logic:

Each row in the matrix is set for an identical musical note. First 8 rows are set for musical notes: "C D E F G A B C" and last 2 are for high pitch notes.

Each column is for 1 time period, set as 200 mSec by default.

Once user hit the play button, he can hear the whole composed song.

Each column plays 10 notes at a time, and the whole pattern is played for 200 mSec X 20 = 4 seconds.

Using setTimer, onTimer functions in MFC, the timer events are generated.

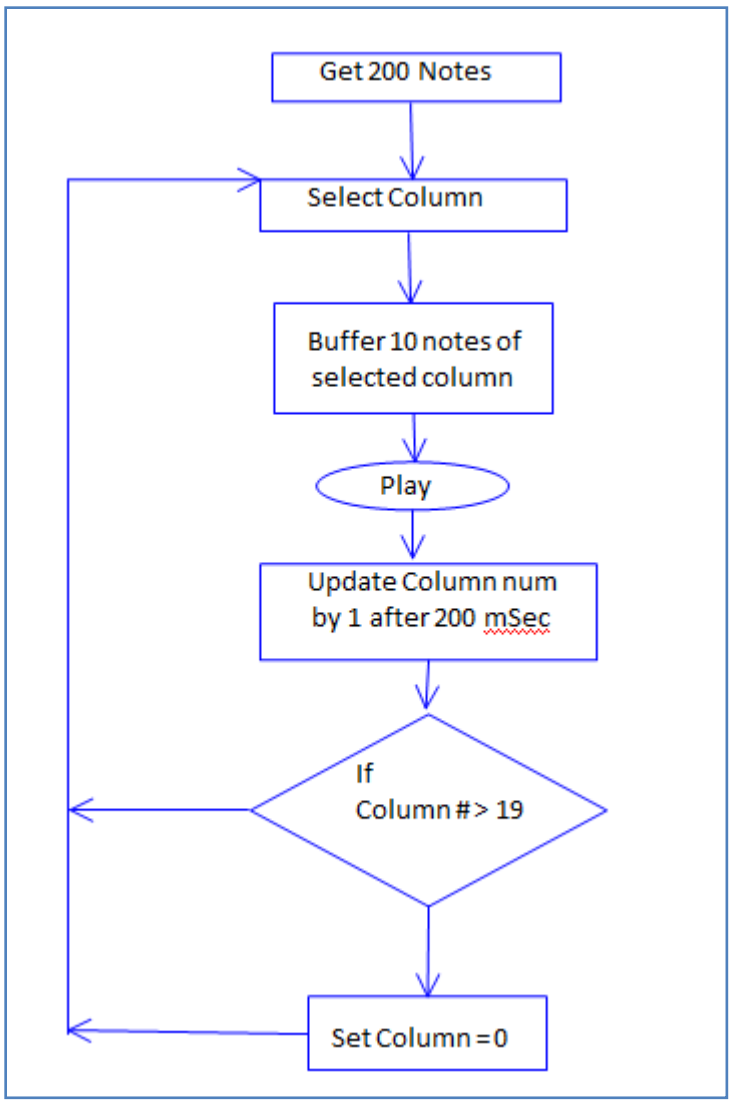


Figure 5:6Algorithm for playing melodies created by MIDI Composer

Chapter 6 Unified Editor

To control any external word peripherals, user experience is equally important along with interface. A Graphical User Interface is a type of user interface that allows user to interact with programs in more ways than typing. A *GUI* offers graphical icons, visual indicators, as opposed to text based interfaces, typed command labels or text navigation to fully represent the information and actions available to user. The actions are usually performed through direct manipulation of the graphical elements.

Visual C++ is the platform here chosen to design Windows forms & controls so as to provide sophisticated user friendly *GUI* for operating on Robots. Here in this project, the motion editor build is to operate iSobot and application calls Win32 APIs.

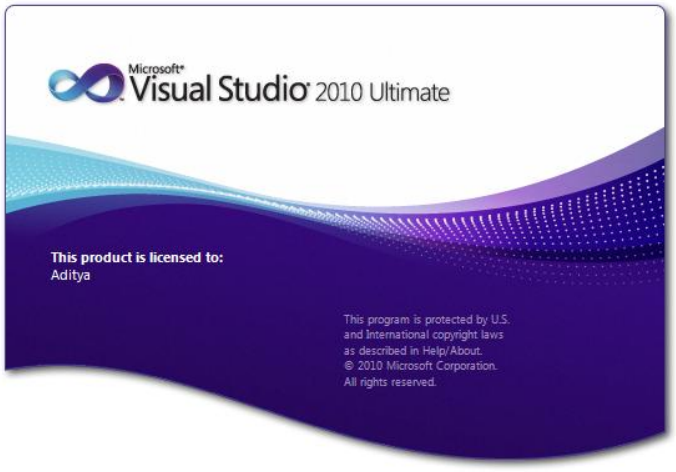


Figure 6:1Microsoft Visual Studio 2010 Ultimate Logo

Win32 API refers to Windows 32 bit Application Programming Interface. It is the interface that Microsoft Windows exposes to applications so they can run and interact with Windows. At one time the only way to write a Windows application was by using the Win32 API directly. However, as applications became more and more complex, higher level libraries and platforms were developed to ease development. Examples of these higher level platforms and libraries include the .NET Framework, Microsoft Foundation Classes commonly known as MFC. These higher level platforms and libraries speed development by including code programmers don't need to write their selves, therefore their use is generally preferred for Windows application development. These libraries sit on top of the Win32 API and in the sense that the application interface with a library, and the library in turn interfaces with the Win32 API. So application using these platforms and libraries still use the Win32 API, just not directly.

However, if one won't mind to write a bit more code, the application can use the Win32 API directly. Since it wouldn't include any higher-level libraries, such an application would generally be smaller and quite possibly faster as well.

MFC stands for Microsoft Foundation Classes. MFC is a large set of classes that simplify Windows programming and speed development. It's important to note that, not only does MFC have methods that one can call from one's application, but parts of MFC actually call one's code.

MFC is an application framework that has to be extended to customize an application. MFC is also object oriented. Programmers extend the default application behavior by

deriving their own classes from those in MFC. Not only does MFC speed development by implementing parts of application, but the application wizards will actually create substantial functionality for a MFC application before one even writes a line of code.

So, this project uses Visual Studio 2010 Ultimate to develop MFC application which requires little knowledge of GUI working & C++.

The following part shows step by step, what elements are added to create a unified motion editor that incorporate all controls' user interface.

Open the serial port and send command to iSobot.

The primary aim of the editor is to send command to the robot, iSobot. As mentioned in chapter 3, we programmed an Arduino board, which converts the serial data received from PC and sends to the iSobot and controls it over Infrared signal. In order to send command to Arduino board using serial port, there is need of opening a port & then send the command.

The simple diagram below shows an interface to let the user open / close the serial port with "ON" / "OFF" button and a "Send" button to send a single static command to the iSobot.

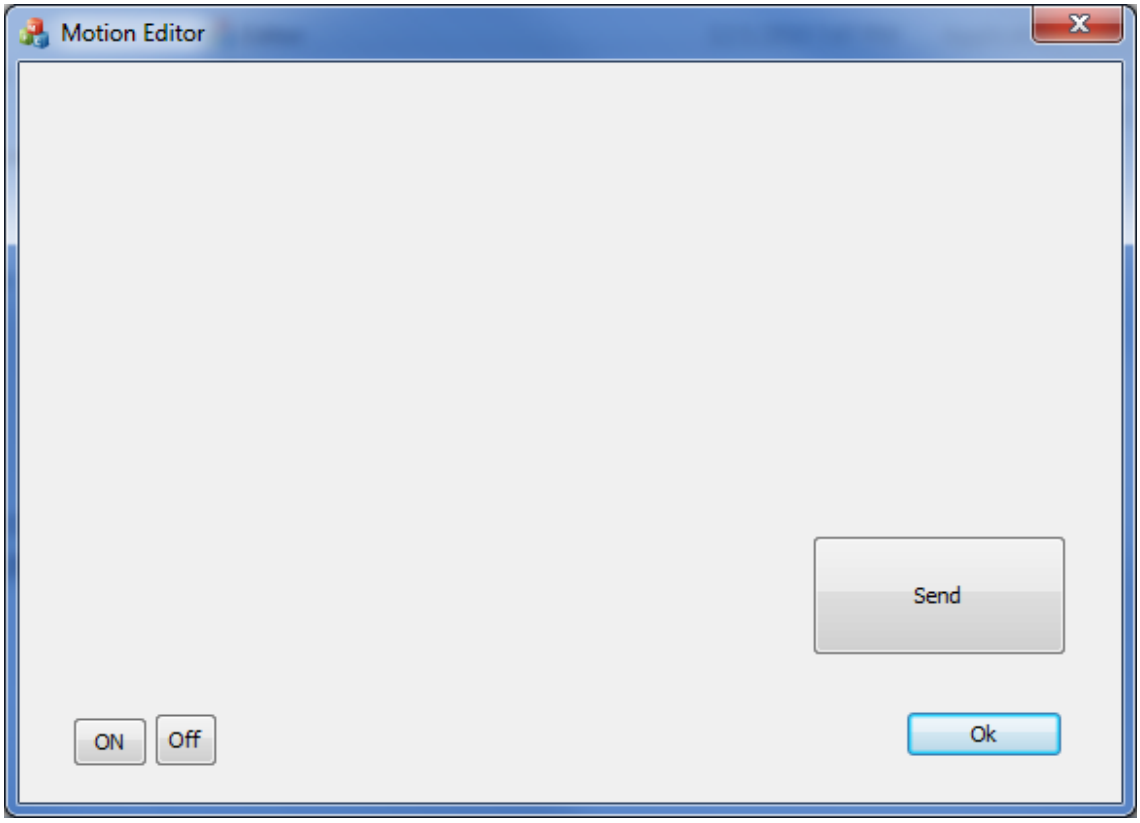


Figure 6:2Basic Interface for opening a serial port and sending command to iSobot

For serial port communication, a third party IP is used, available free on internet.[43].

An object of CSrialPort class is created which has several methods for serial port communication. The list below shows what button handler calls which method of serial port.

Button	Handles
ON	Open the Serial Port
OFF	Close the Serial Port
Send	Send a single command to iSobot

OK	**Remove (flush) all resources used by system & quit dialog window**

Table 6-1Serial port connection button interface

Managing command library for robots

Different kinds of robots have

- Different commands of actions
- Different number of motors, sensors, controls.
- Different roles, functionality
- Different speed, Action sets, Motion patterns.

But one thing they have in most common is:

They need commands through serial port either from PC or Microcontroller, if controlled remotely.

Hence, I created a list (Chart), which has 4 essential parameters:

- ID : Just a serial number (important in the program since each action has unique ID, henceforth ID can be used as a pointer)
- Command Name : The command caption, should relate to robot action
- Command code : The Hex code to be sent to robot serially
- Time : The minimum time to perform the action completely.

When user wants to add a new robot and he knows set of commands for that particular robot along with serial command code, then he can simply create an excel sheet as shown in Figure6.1 and save the list as ".csv" format.

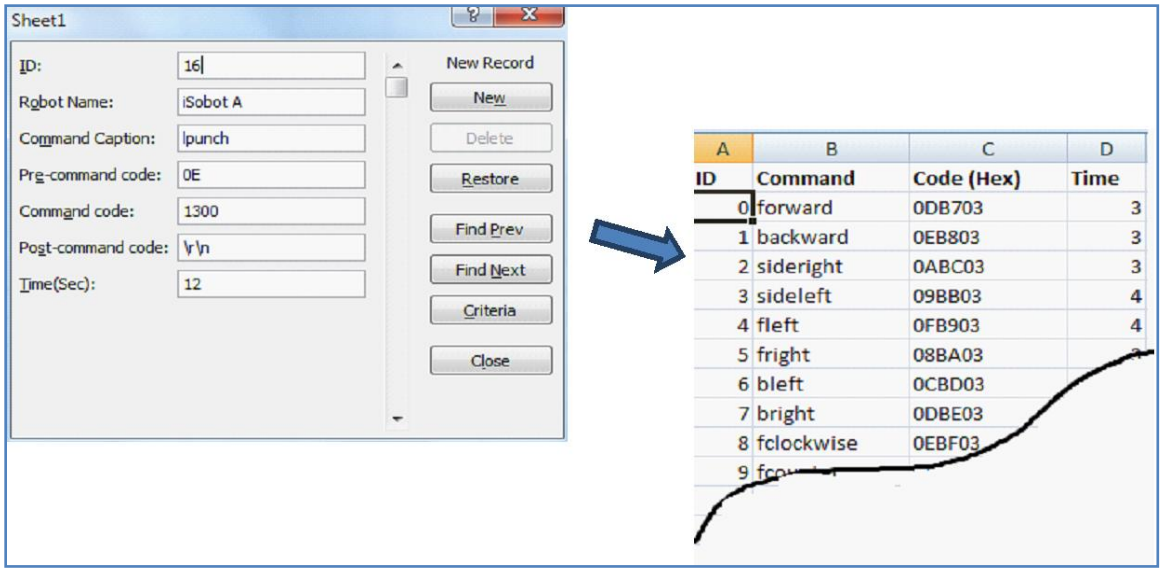


Figure 6:3Creating a table in Excel with standard format for Robot commands

Let user pick command and send to iSobot

iSobot has rich set of commands. They can be stored in an array / vector and populated in a list box where user can select particular command and send it to see robot acting. This approach makes the editor only limited to iSobot and non other robot. In order to make it universal ie good for controlling any robot for which list of command set is available, a dynamic approach is used.

1. All commands of particular robot are written / copied in a simple Excel sheet.

2. The excel sheet is saved in CSV file (Comma Separated View) list file.
3. A class created to read this CSV file.
4. Copied all the commands for that particular robot in a vector.

For the user interface, a list box is created in the dialog window. Using controller of the list box, all the commands stored in the vector are inserted in the list box.

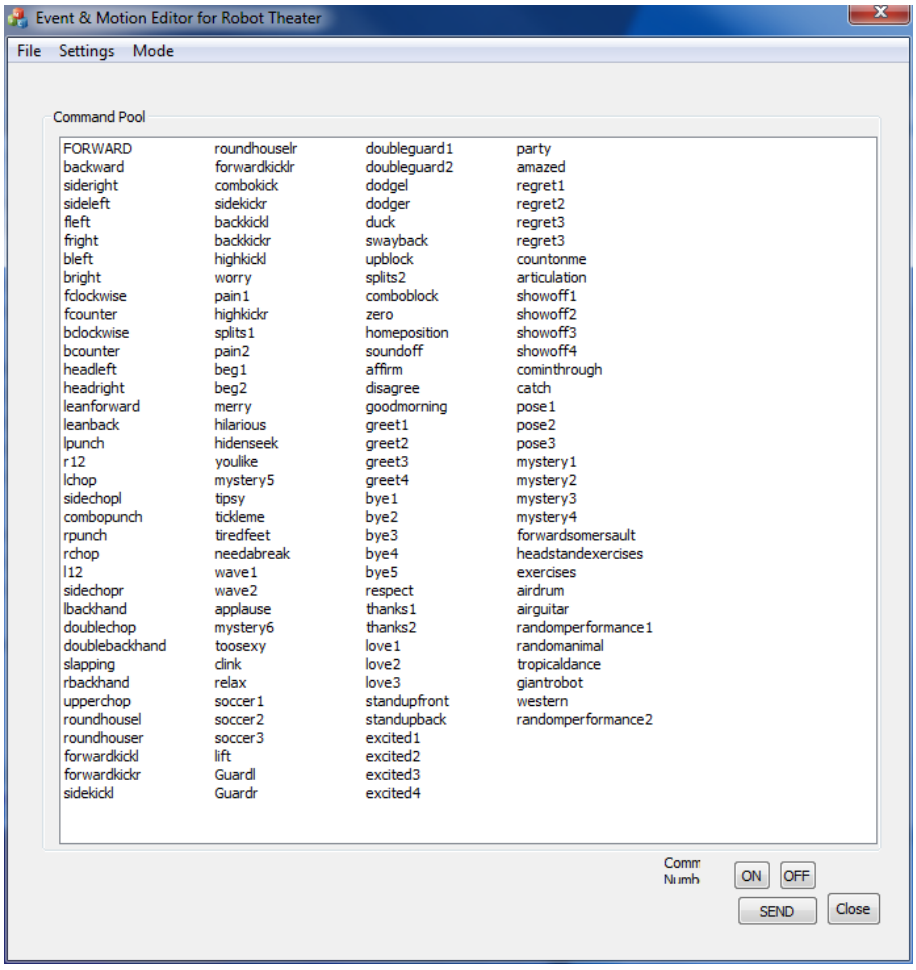


Figure 6:4List box with all commands let user pick one and send to iSobot

Yet, there is no connection between chosen command by user and the command to be sent to iSobot. Here comes the need to understand how Graphical User Interface works. Several books & MSDN help database can be a good resource for understanding those little basics [27].

Whenever user clicks on a particular command in the list box, an event is generated. A callback routine is needed to drive some action for generated event. This kind of programming is often referred to as *event-driven* programming. In the example, a button click is one such event. In event-driven programming, callback execution is *asynchronous*, that is, it is triggered by events external to the software.

Most GUIs wait for their user to manipulate a control, and then respond to each action in turn. Each control, and the GUI itself, has one or more user-written routines (executable C++ code) known as *callbacks*, named for the fact that they "call back" to dialog window to ask it to do things. The execution of each callback is triggered by a particular user action such as pressing a screen button, clicking a mouse button, selecting a menu item, typing a string or a numeric value, or passing the cursor over a component. The GUI then responds to these *events*. One, as the creator of the GUI, provides callbacks which define what the components do to handle events.

Hence, within the callback routine, steps executed are:

1. Get the item number in the listbox selected by user.

2. Find the command listed at that particular item number. Command number is different than the item number of the list box. The command number is associated with the CSV list of iSobot command list.
3. Use this command number as a pointer to find the code associated with that command. The code is nothing but the Hex code of 6 chars to be sent to iSobot. [chapter 3]

Now, if user clicks the Send button, provided the serial connection is already established followed by "ON" button, then the 6 char hex code is sent to the robot through serial port.

Select multiple commands and send all with one button click.

The approach discussed above was good to send single command to robot, but what if multiple commands have to be sent? Then, there is a need arise to add another list box in which user can accumulate all the commands and make a list of them. Once enough commands are selected, then a send button can send all commands at a time. "Report" style list box [figure 6:5] is chosen for this list, so multiple columns can be added.

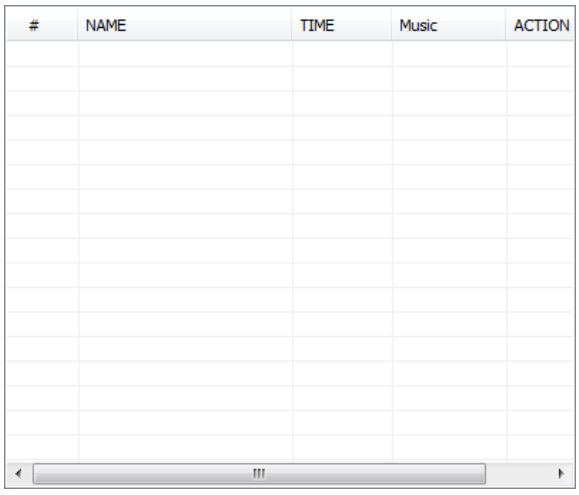


Figure 6:5Report style list box with multiple columns

Refer the MSDN site [42] for varieties of styles of list box. This list box is named and referred as “Script” list.

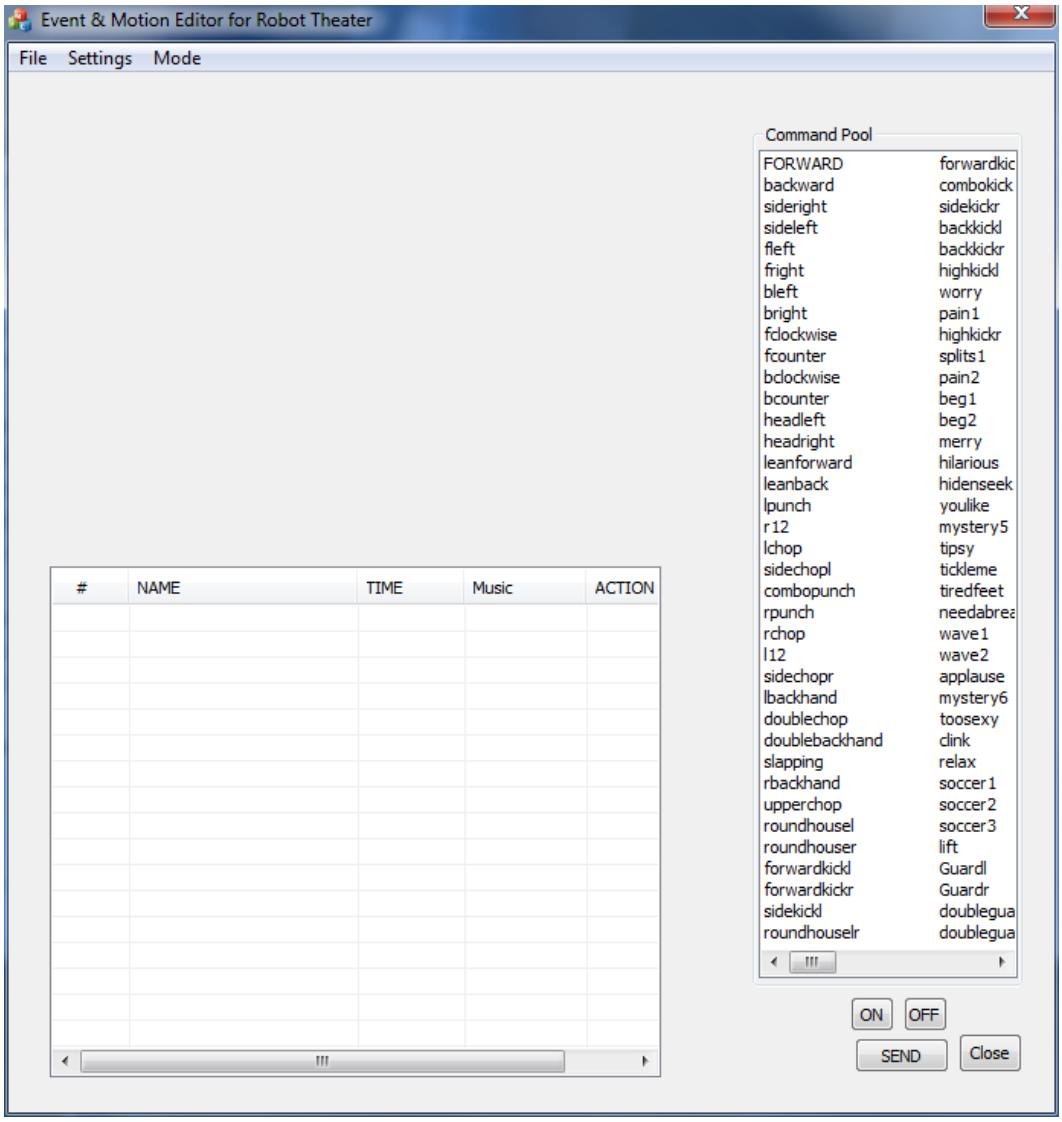


Figure 6:6GUI for two list boxes. One for all commands and another for Script

When user double clicks on command list,

1. The handler routine copies the item number selected from command list.
2. The name of the “command name” associated with that command number is copied in a string variable.
3. Also copied the “time of execution” associated with that command number in an integer variable.
4. Finally, the command number and string format of integer value of “time of execution” are inserted in a new row of Script list, as shown in Figure 6-4

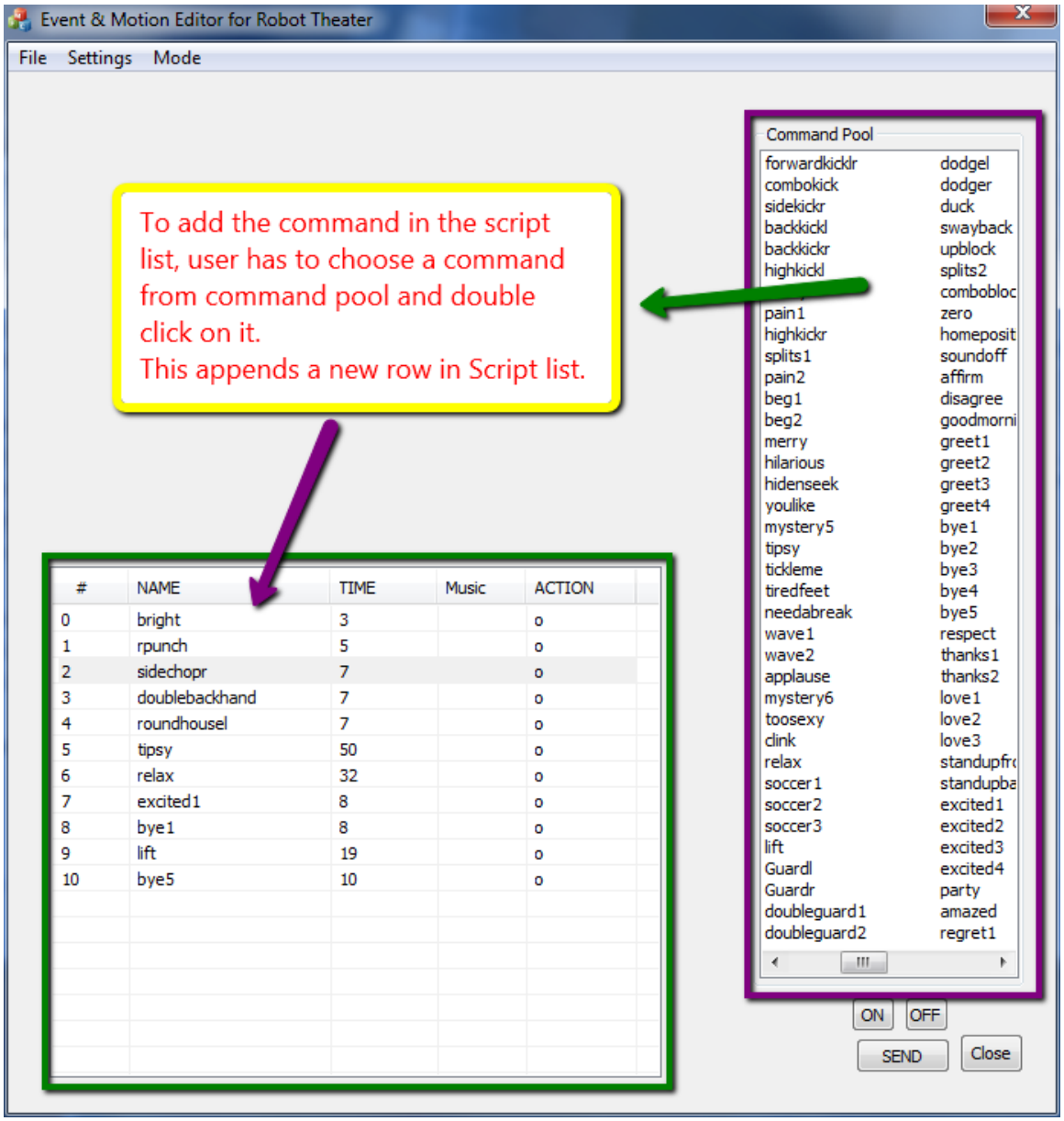


Figure 6:7GUI for adding commands in Script List from Command Pool

Need of Timer

Since, iSobot (and most other robots) executes only one command at a time, sending all commands together won't work. In that case, iSobot will execute only the first command and rest all will be sent through the serial port one by one, but won't be fetched by iSobot

since it is already busy in executing first command action and not ready to accept any other command.

Hence, PC needs to wait until iSobot finishes his first command and then send the second one. Since there is no feedback from iSobot about action completed, there is no easy way to know when it is again ready to accept new command. Therefore, execution time is calculated for each command just by running individual action. Previous editor built in Python was used for this time calculation. The execution time of each command has been given space in the Script list view. [Figure 6-4].

#	NAME	TIME	Music	ACTION
0	bright	3		o
1	rpunch	5		o
2	sidechopr	7		o
3	doublebackhand	7		o
4	roundhousel	7		o
5	tipsy	50		o
6	relax	32		o
7	excited1	8		o
8	bye1	8		o
9	lift	19		o
10	bye5	10		o

Figure 6:8A Special Column for the command execution time in the Script list.

To add a wait after each command sent, sleep(*time*) function can be used. But, this is a very lame approach. This freezes PC so badly that user can not even see the cursor moving on screen when mouse moved.

So, windows inbuilt timer functions are used which spawns the thread. How do Win32 timers work? First, create a timer, specify its elapse time, and (optionally) attach it to a window. After being created, timer sends WM_TIMER messages to the window message queue, or to the application queue if no window was specified. We can process this message to call the code that we want to be executed in regular time intervals. The timer will send WM_TIMER messages until it is destroyed.

To create a timer, Win32 function can be used as below:

UINT_PTR SetTimer(HWND hWnd, UINT_PTR nIDEvent, UINT uElapse, TIMERPROC lpTimerFunc);

Arguments:

- hWnd - The handle of the window to which the timer is associated; may be NULL, in which case nIDEvent is ignored, and the return value serves as the timer identifier.
- nIDEvent - A nonzero timer identifier.
- uElapse - Timer's time-out interval in milliseconds.

- lpTimerFunc - An application-defined callback function that processes WM_TIMER messages. May be NULL (more often than not, it is).

Using these timers, all commands in the script list can be sent one by one after specified time interval without letting the computer freeze. The command time in the forth column is the time required to execute that one command.

Manipulation within Script

As described in the above section, the user can add different commands from the command pool and create a script that can be played as a sequence of all commands. From the user's experience point of view, the user must have the flexibility to manipulate the commands within the script list view. Hence 4 buttons are as figure 6-3. Their bitmap Figures are created using inbuilt editor in visual studio.

Figure 6:9Customize Script list items with UP DOWN or DELETE button

Button	Action
UP	Move the selected command one step above in the list
Down	Move the selected command one step below in the list
Delete	Delete the selected command
Play	Plays the script (Send all commands through serial port)

Table 6-2Picture Button Controls

Logic used to move the commands:

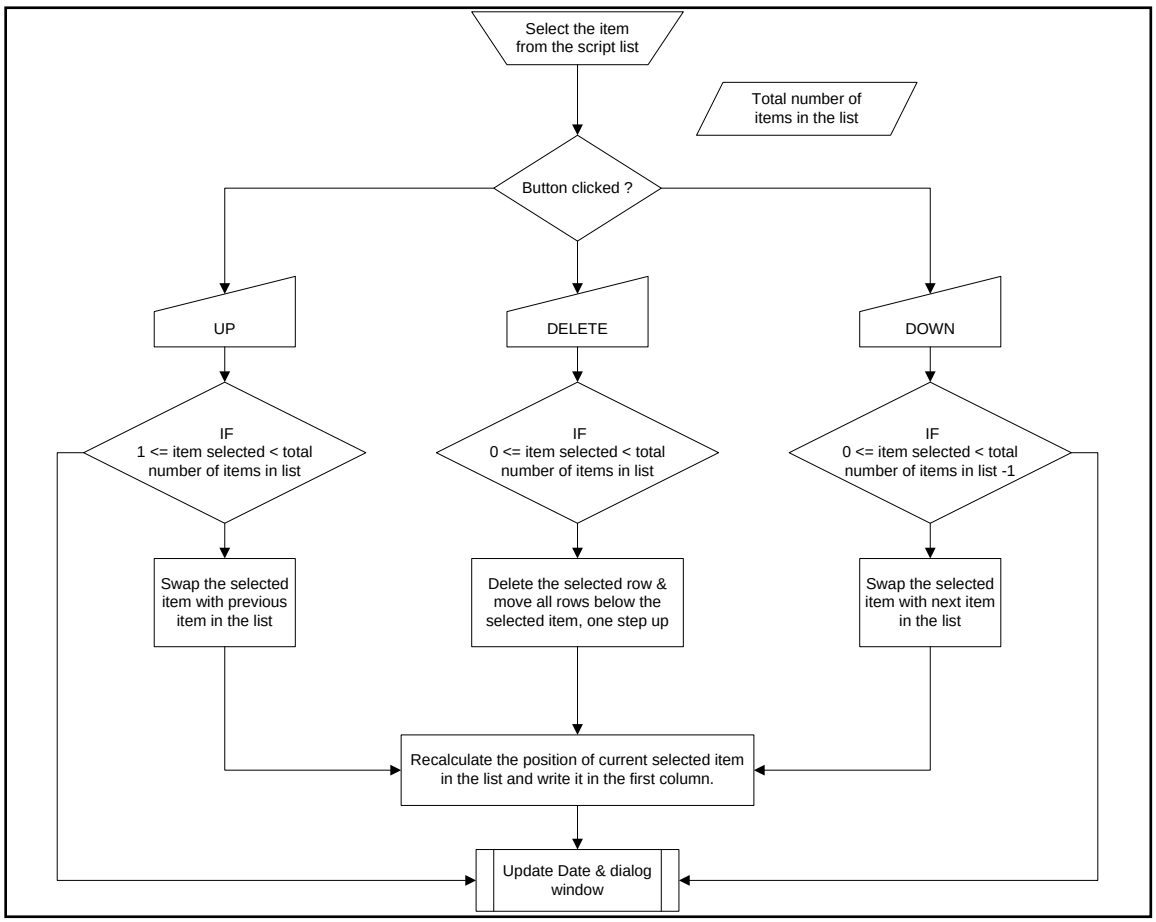


Figure 6:101Action behind Up Down & Delete button in Script List

Preview the command action

It's fun to be able to add different commands easily to the script list from command pool but to create an appropriate script, with some meaningful action, the concatenated actions should be somehow related. Appropriate caption of command names can help to understand what could be the role played for particular command but not always.

For e.g. iSobot has 3 different actions to say "bye". They all are named as bye1, bye2, bye3. The best way to remind *"which kind of "bye" action does what?"* is to see its actual preview, its video clip. Hence an active X control object of Media player is added in the editor.

Why Media player?

1. User (Robot motion movie creator) can see the preview of individual robot action.
2. Reduces hassle of checking robot action every time by sending commands through port and waiting for response.
3. No need to remind actions any more, although commands name is pretty explanatory.
4. The embedded Media player has almost all features like: Play, stop, Mute, volume control, scroll, time display.
5. Recording all action video previews didn't take very long. I did it using mobile camera and converted them in media player format.

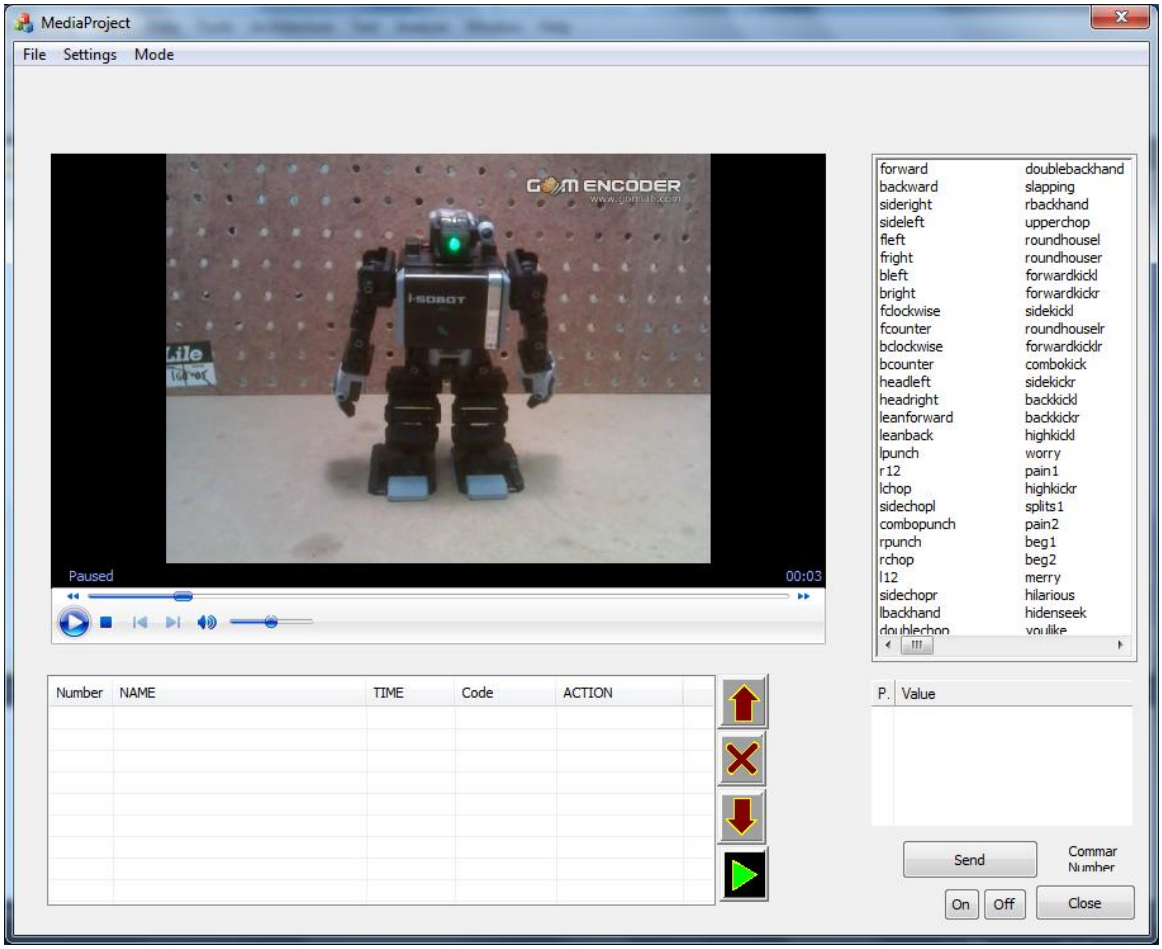


Figure 6:11Embedded Windows Media Player interface within editor

There are several different ways to use the Windows Media Player control in a C++ program. You can create an instance of the control in a console application, or you can embed the control in a Windows application. Also, you can implement interfaces that enable you to run an embedded Player control in remote mode. You can customize the user interface of an embedded control by applying a skin definition file [28].

It took a bit time to understand how to use Active X control to interface COM APIs or Windows based application into GUI, but it was worth spending time. The crucial part was to find the right ".dll" file to use methods associated with the media player.

DLL for Windows Media Player: wmp.dll

DLL Path: C:\Windows\System32

Interface: CWMPPlayer4

Interface files: CWMPPlayer4.h, CWMPPlayer4.cpp

The important methods used in the project:

- `Void put_URL(LPCTSTR newValue)`

  `This methods takes the filename to be played as an input. Be sure to add the path name too.`

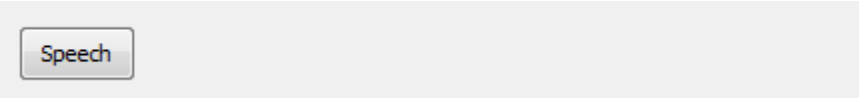
All this method takes is one simple argument "LPCTSTR *newValue*", which is nothing but the complete path name of the video file stored in the secondary memory (hard disk). This lets the video clip start playing automatically.

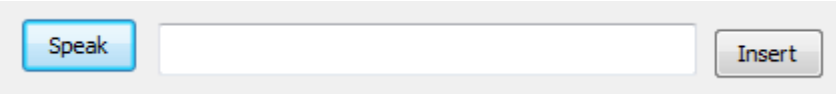
Addition of Speech

Now, the editor is able to let the user pick some commands and add to the script list. Before adding to the list, one can also see the video of command action. In order to enhance the play, some speech announcements can be added.

The details of Speech Synthesizer & MSDN Speech SDK are already mentioned in chapter 6. With available pool of methods for Speech, much can be done with speech voice like, adjust the speech volume, speed, tempo, human voice but it is equally important to keep the graphical user interface neat and avoid too many buttons which adds complexity.

Therefore, at minimum, only three graphical elements are added into the editor: two buttons and one textbox.





Element	Caption	Purpose
Button	Speech / Speak	Speech – Enable "Insert" button & text box visible, turns the own button text to "Speak" Speak – Speak the input text in the text box
Button	Insert	Insert the speech in the script list, then disable the text box

			visibility and the button “Insert” itself.
Text Box	Input here	Text	Let the user add his text to be spoken

Table 6-3GUI elements for inserting speech text

As can be seen in the table above, the same button has two captions ‘Speech’ & ‘Speak’. This way, one button can be reduced and a bool variable can be used as a mutex object between two functions to share the same button as event generator.

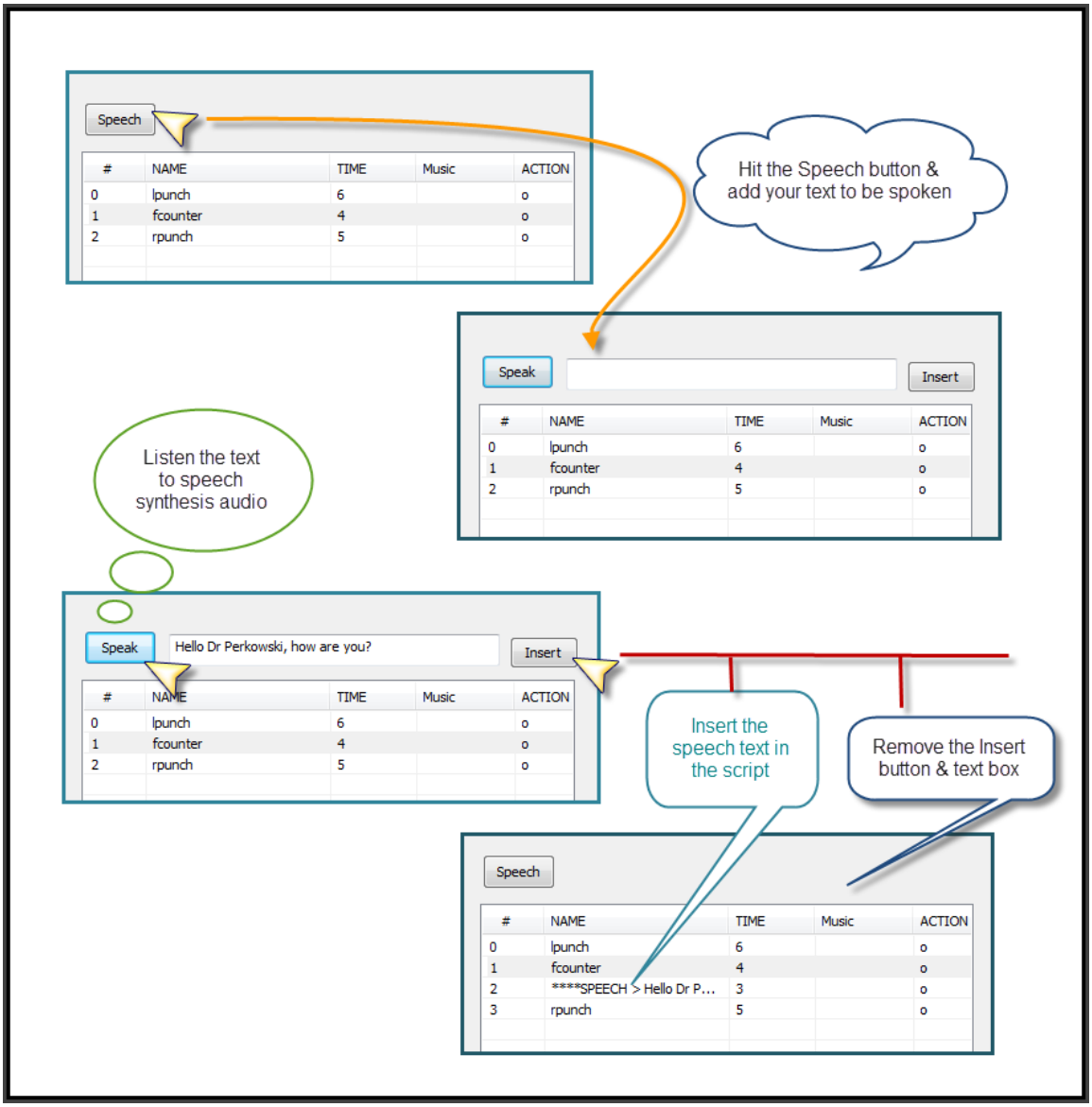


Figure 6:12Clean & Tidy interface for adding speech text in the script list.

How/ what to add / insert the speech in script list?

Since the script list is full of integers, the question is obvious how the string is inserted? Also, the user has to be able to add multiple text strings within the text strings, not the only one.

Hence, a vector of CString variable is declared. A static integer variable is also initialized to 0, named: speechCounter.

Somehow, the text pointer should be added in the script list as an integer, as like command numbers of iSobot, and at the same time, iSobot command number must be different than those of text strings. So, all text strings are given command number at & after number '1000'. The number 1000 is taken just to be far ahead from the maximum number of commands of iSobot (140), so they don't have any command number in common.

Every time a new text is inserted, a speechCounter is incremented& added to '1000' to generate the command number to be added in the script list.

Eg: to insert 3 text strings in Script list:

1. Hello, how are you?
2. I'm fine. How are you doing?
3. I'm doing well too. Thank you.

Vector <CString>	Location in the vector	Static int speechCounter	Command number after adding 1000
Hello, how are you?	0	0	1000
I'm fine. How are you doing?	1	1	1001
I'm doing well. Thank you.	2	2	1002

Table 6-4Using int values as pointer to store text strings

While inserting in the script list, the state diagram [num] is followed.

Same is the logic used while sending commands to play the script.

If (command number < 1000)

then send command to iSobot

else

Speak the string at Speech[command number – 1000].

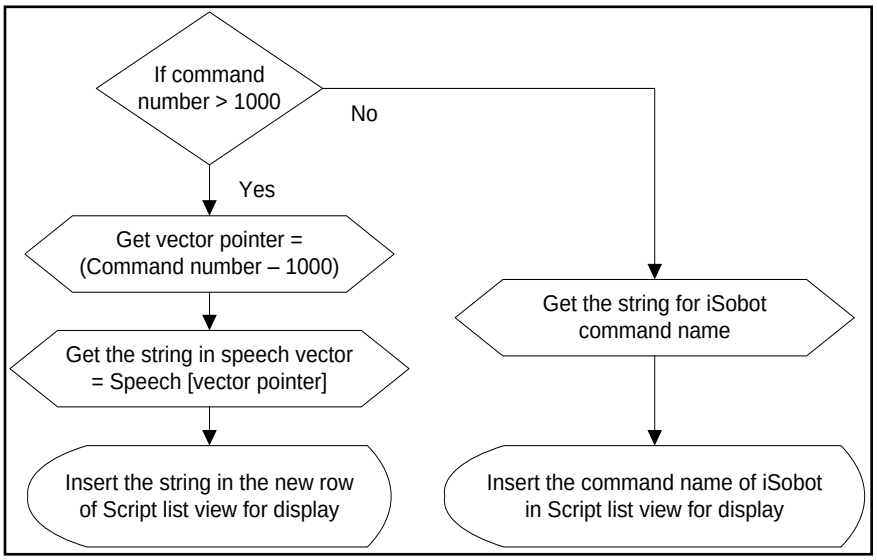


Table 6-5Simple logic to differentiate command ID number from those of String pointer location number.

Use of Vectors

As the complexity of editor grows up, it becomes difficult to manage the different arrays created for list data. Hence vectors are used for storing list data in the back end

processing. For the front end, graphical interface, the data is fetched from vectors & displayed in the list boxes.

MIDI Composer

The working of MIDI, its purpose is all discussed in chapter 7. Its basic interface is also described in there. This part below will demonstrate how rests of the graphical interface elements are added in the Motion Editor Dialog window.

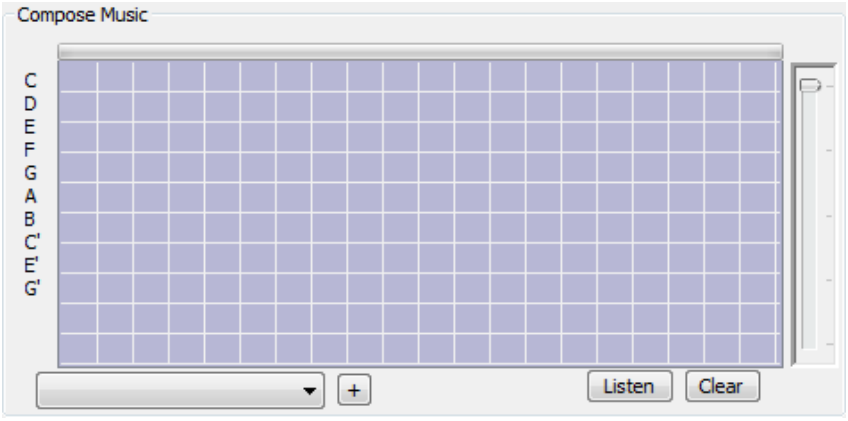


Figure 6:13GUI for MIDI Composer

As shown in the Figure 6-13, some additional graphical user interface elements are added along with the basic composer (music editor) shown in chapter 7. The list & purpose of those elements:

GUI Element	Variable used / Caption	Purpose
List Box	BOOL matrix[10][20]	Let the user add / remove 1's
Progress Bar		Shows progress of period from 0 to 4 sec
Button	Listen / Stop	Start / stop playing the midi pattern
Button	Clear	Remove all 1's from matrix, reset.
Button	+ (Save)	Add the current pattern to selected melody
Slider Bar		Change the scale of the MIDI sound
Drop down Box	Melody 1 - 100	Shows / let user select melodies out of melody pool
List Box	Instrument 1 – 128	Shows / let user select Instrument out of the list.

Table 6-6Graphical element objects used for MIDI composing user interface

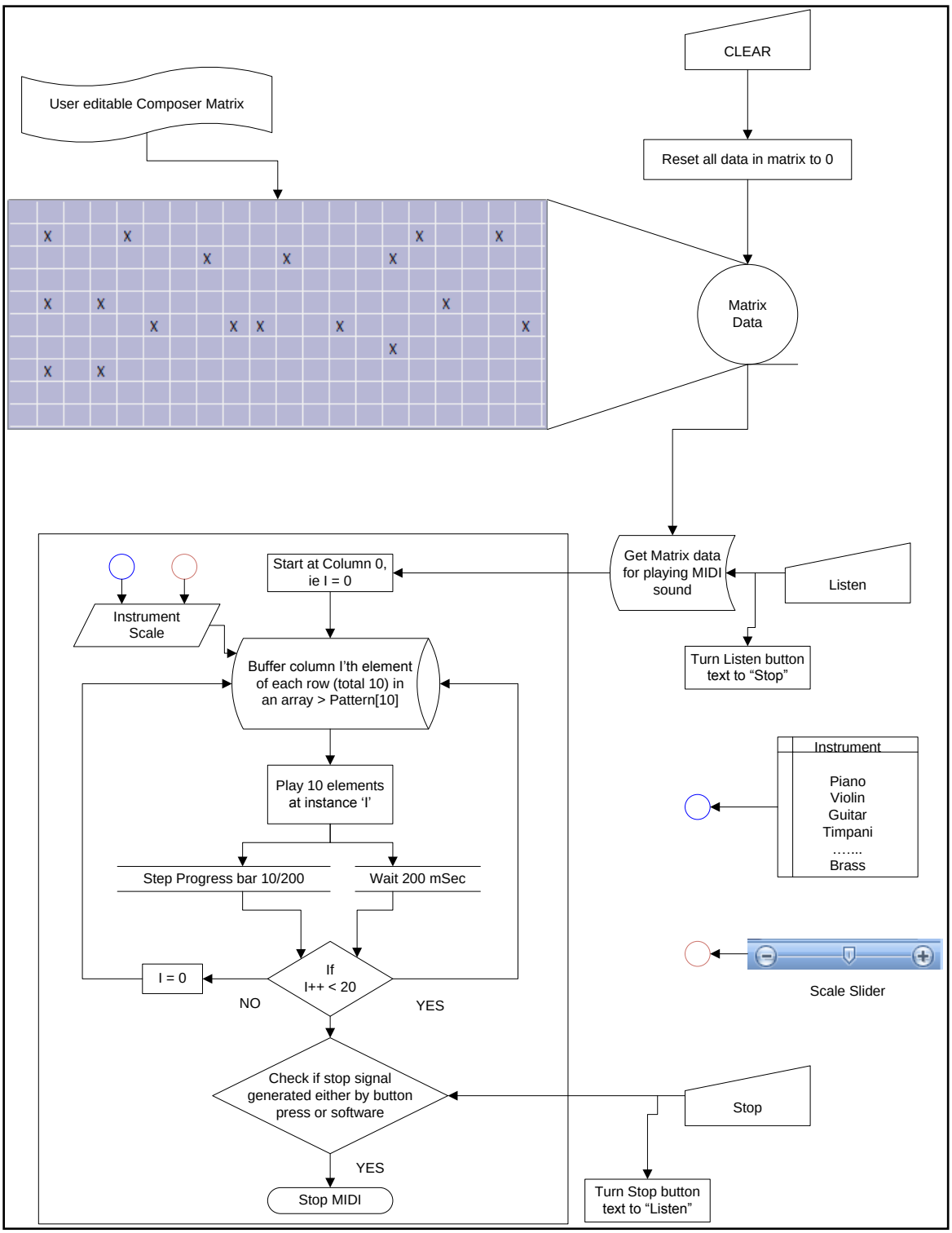


Figure 6:14MIDI Composer Interface

Description of chart:

Composer is the blue matrix appearing in the diagram above, is of size 10 rows x 20 column.

Some blocks in the matrix seem marked and remaining ones are blank. Graphical element used for this view is of “report” style list box. Inserting 20 columns in the beginning and then adding 10 rows initialize it. This list box is representing the Boolean data in matrix[10][20]. The ‘x’ mark in the block represent data “TRUE” in matrix for that particular node, and blank represent “FALSE”. Eg. matrix[1][1] = TRUE, matrix[0][3] = FALSE.

When user clicks on any block in the matrix with mouse click, change appearing in the list box as well as in the matrix is shown in the state diagram below.

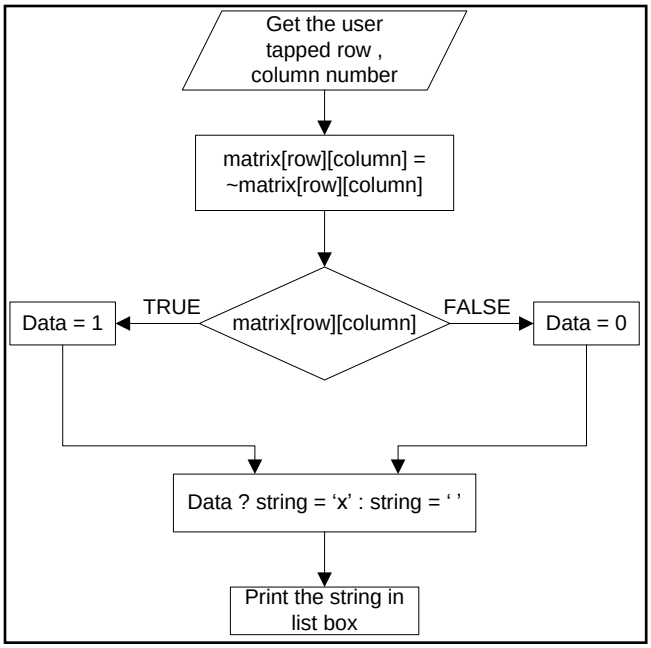


Figure 6:15Playing each bit of melody.

This is how one data of one node is displayed in the list box. The overall matrix need not be updated after every user click. Individual matrix node & its graphical view in the list box update fasten the process.

At the same time, when particular node is tapped, the midi sound is generated for that node so that user can get immediate preview of the sound.

Once the overall matrix is filled up as user require it, and he clicks on "Listen" button, the midi sound starts playing. More details are in chapter 7, but for diagram explanation, it is shown that the matrix data is copied in a buffer and lend to play. 20 columns of 10 bits data are played at interval of 200 mSec. After each interval, progress bar is stepped up by 10/200. For this, progress bar has to be initialized with following details:

ProgressBar range (min, max) = 0, 200

ProgressBar step size = 10

This setting lets the progress bar increment by 10 steps after each function call of:

progressBar.stepIt();

Slider: Moving the slider generate the event. In the callback routine for this event, current slider position can be fetched using function: slider.getPos();

Slider initialization setting:

Slider range (min, max) = 0 , 4

slider tic mark frequency = 1

For more details about slider & progress bar, refer MSDN site [29. 30].

For the value of slider position, scale is selected as:

Slider Pos	Scale base set
0	40
1	50
2	60
3	70
4	80

Table 6-7Slider value used to set MIDI scale base

Instrument Selection:

MIDI has decent library of instruments. There are total 128 instruments, among one can be selected & played sound with. To have the user select one of those, a list box is needed. Adding another list box that can contain those many items in the list will add mess in the graphical interface. The command list box is ideal for reusing it to display the whole list of Instruments. Hence, the list items in command list box are all deleted and filled up with instruments while composing midi sound. When composing is done, the display control of list box is given back to iSobot command list.

Selecting an instrument is similar to selecting a command for iSobot from that list box. After user clicks on particular item in the list box, the item number is noted and set to integer variable named “instrument”.

The ‘scale’&‘instrument’ are integer variables that can be used to set the scale & instrument of the midi playing.

Dropdown box:

The dropdown box is a similar to list box. It needs to be initialized with items as string. A vector of CString is used to contain all items to be displayed in the drop down list box. This list is filled with the names as: Melody 1 through Melody 100. Hence, user can save / load 100 melodies and use as when needed. Saving and loading data to and from the file is discussed in later section, but for brief understanding, this operation clears the current matrix[10][20] data and set it with the another loaded matrix data.

A melody is a piece of sound, that user just composed with mouse clicks in the composer list box. Adding another step, it is played in particular scale and with particular instrument. The scale and instrument are set with integer variable and the matrix is of BOOL type. Hence, a structure of melody created that contains all these 3 elements.

```
Typedef struct melodies {
BOOL matrix[10][20];
int instrument;
int scale;
}melody[100];
```

When user clicks on dropdown box and select particular Melody, the whole structure is to be fetched & made according graphical UI changes as:

1. Clear the current matrix[10][20], reset all items to 0.
2. Copy the contents of matrix stored in melody structure to current matrix.
3. Set the midi instrument with new value = melody.instrument.
4. Set the midi scale with new value = melody.scale.

This is how, the new data comes to the user view selected from dropdown box list and loaded into buffer.

Similarly, to save the current buffered data value to the selected melody structure, exactly opposite is done. The current values of instrument, scale and matrix are stored in the current selected melody structure, when user hits “+” button.

Menu bar

Most windows based graphical user interfaces have Menu bar at the top of dialog window. Keeping the menu bar in standard format makes it more users friendly. Figure 6-14shows menu bar:

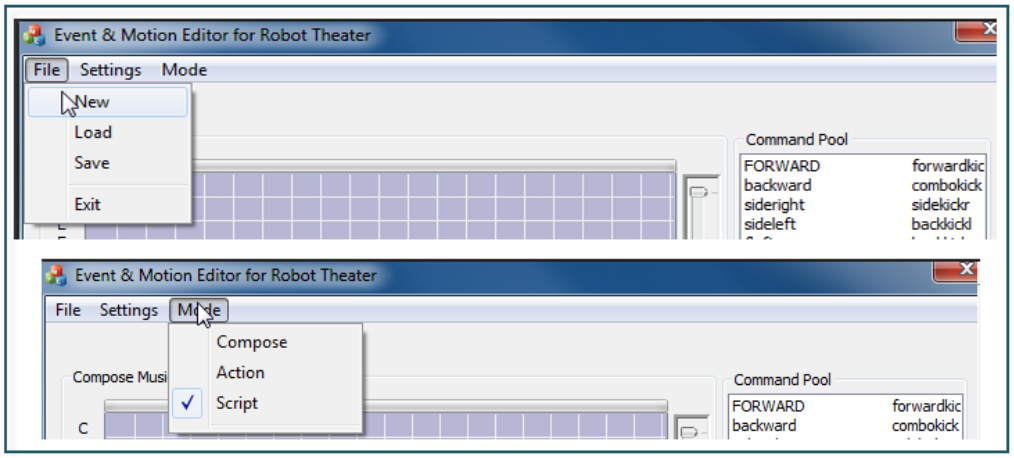


Figure 6:16Menu Bars

File	Setting	Mode
New	Connect	Compose
Load	Robot	Action
Save		Script
Exit		

Table 6-8Menu Bar Items

Menu item	Purpose
File > New	Clear all the items in Script list box & script vector
File > Load	Open the dialog window to open a file
File > Save	Open the dialog window to Save a file
File > Exit	Flush the dialog resources & close the window
Setting > connect	Connect to serial port
Setting > Robot	Select among 2 robots: iSobot A, iSobot B.
Mode > Compose	Choose Compose mode
Mode > Action	Choose Action mode
Mode > Script	Choose Script mode

Table 6-9Purpose / Action taken for Menu bar item clicks

What are Modes?

There are 3 modes among which, user can select only 1 at a time. These 3 modes are described briefly in below:

1. **Compose mode:**

   In this mode, user can compose the melody. Clicking on compose matrix won't make any change in any other modes. In that way, user can avoid messing up created melody by accidently clicking on any of the matrix node while not in compose mode.

Secondly, when user selects this mode, the command list box control is cleared and populated with list of Instruments to be used for MIDI.

2. **Action mode:**

   In this mode, command list box is first populated with list of iSobot commands. While in this mode, when user double clicks on a command, embedded Windows Media Player opens up in another pop up window & plays the video of the selected iSobot command.

3. **Script mode:**

   Command list box is populated with iSobot commands. In this mode, when user double clicks on the commands in the command list box, its item number is selected and the same is pushed in the script vector. This is how; the user can select the commands to add in script play list.

   The script can only be edited in this mode only. User can add melodies in the script & add expressions for each command action in the script.

About file > New, Load, Save:

Every time user creates the new script, there must be a way to save it, so that it can be reused.

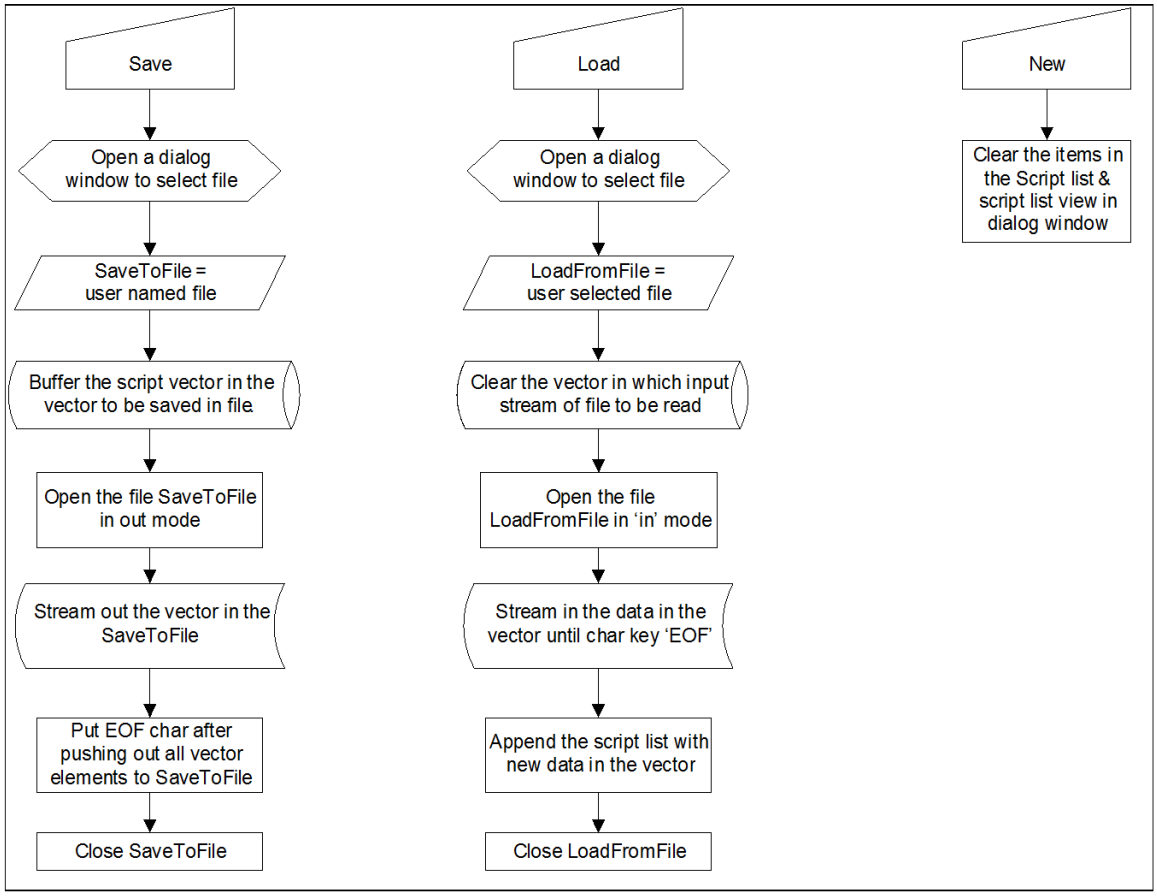


Figure 6:17Ssave / Load script. Clear Existing one script and create new.

Load file appends the previously saved script in the current script list rather than opening just the saved script as it is.

Regular Expression

The regular expressions have been given the 5^{th} column space in the script list view. To add the expression for each command, user has to click on the 5^{th} column. Then, the expression list view gets visible. Choosing one of the expressions from this list adds that expression into the script list and again closes the expression list view window.

The expression list view is just like other list box window item, just that it is invisible and activated only when user clicks in script list view with intention to add an expression.

Play Button

The play button, the final element in GUI discussion does the last &most important thing. When user clicks on this button, all commands in the script list view are sent for regular expression calculation and the generated output vector full of commands to be sent sequentially & parallel to robot, speech synthesizer, audio device & light controller. Figure 6-19, 20 shows the brief algorithm.

The regular expression library can be used for solving the expression [10]. In this implementation, simple parsing method is used.

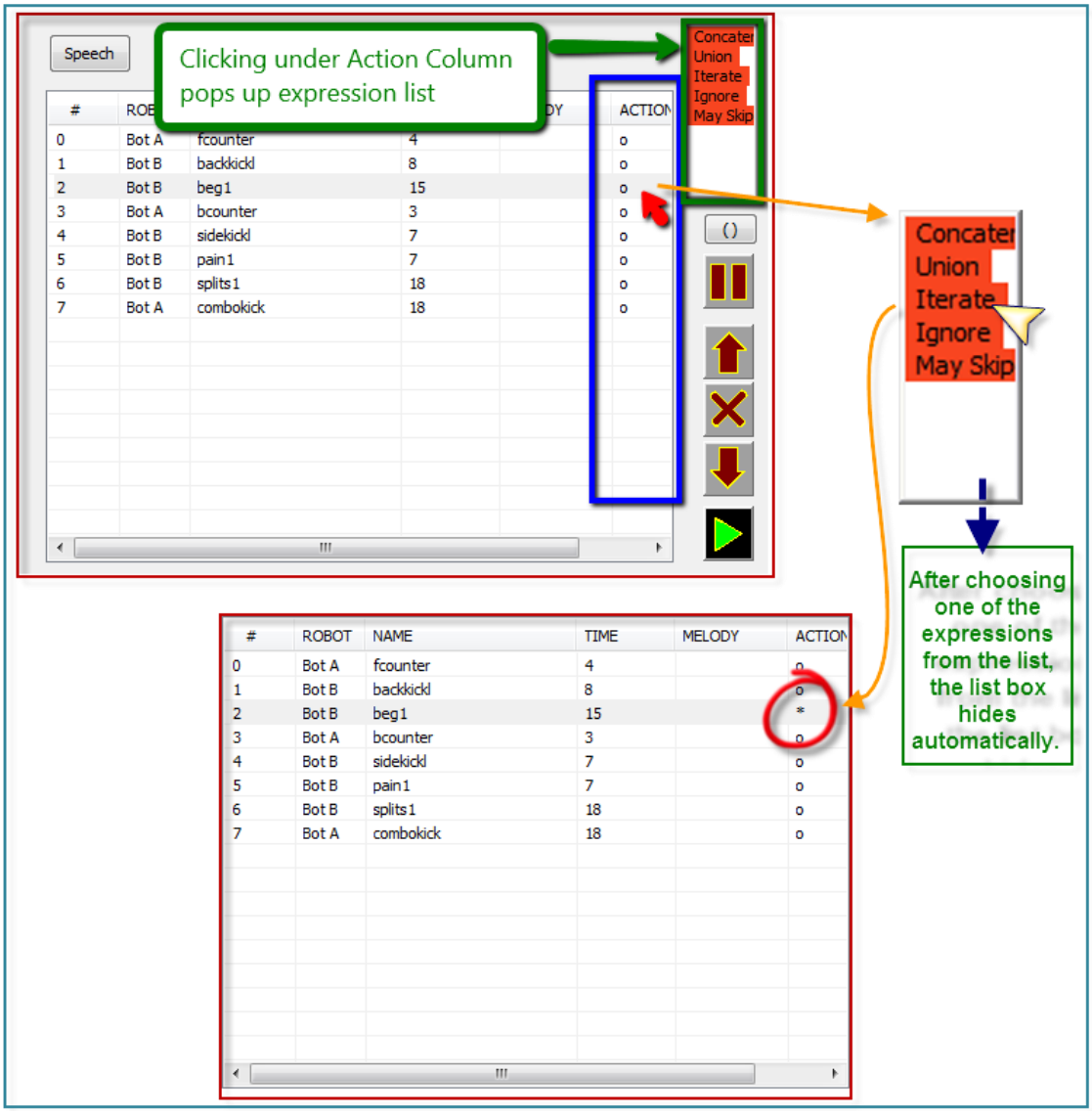


Figure 6:18GUI for Editing Regular Expression

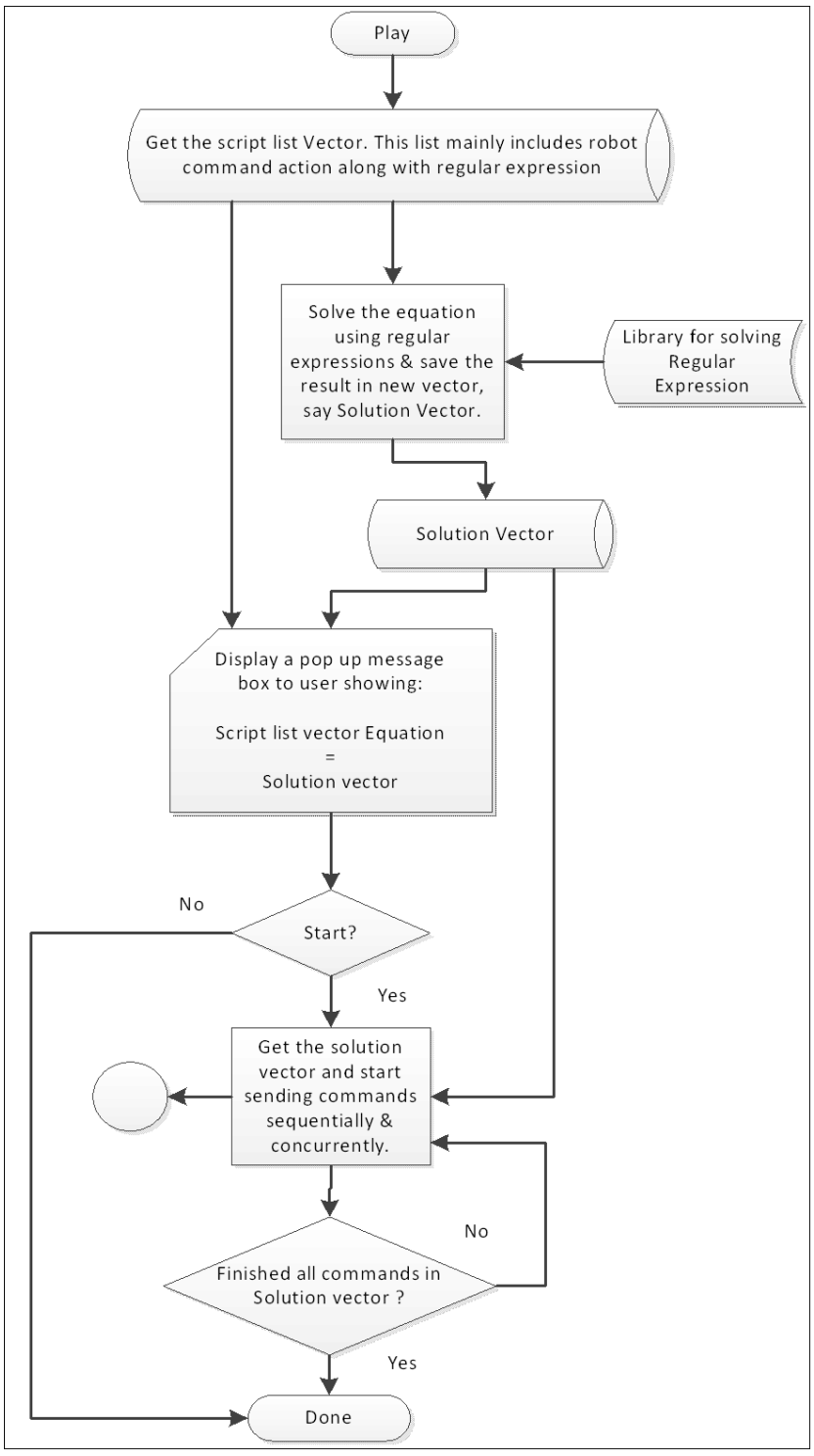


Figure 6:19Play button is to solve regular expression first and then to Execute commands in the solution vector

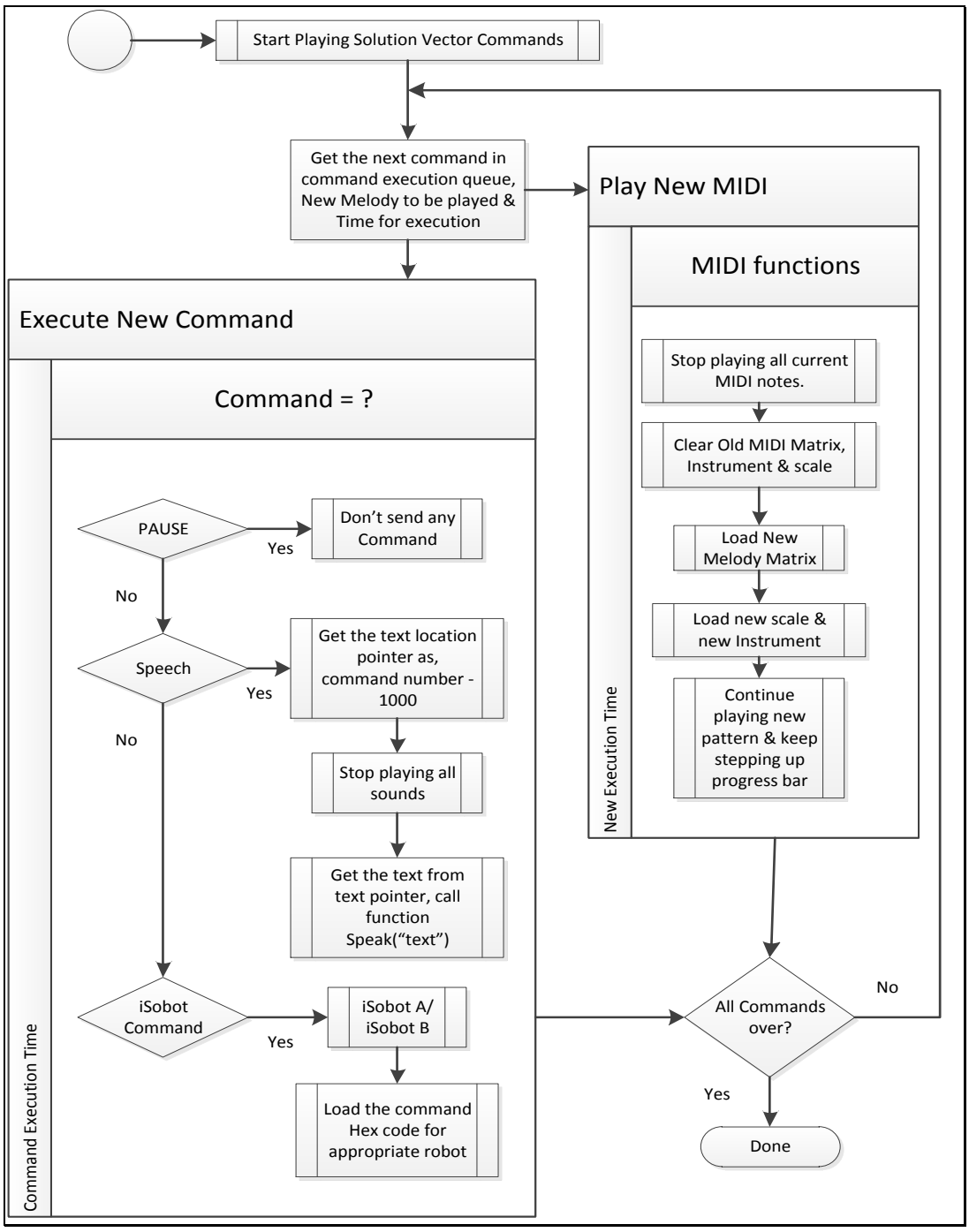


Figure 6:20after solving regular expression; execute the commands in solution vector sequentially as well as concurrently.

Chapter 7 Hardware Setup & Software Manual

The efforts to keep user interface extremely user friendly and self explanatory, makes the job of writing software manual easier. This chapter is more like a manual for using the overall project assembly and full of notes, Instructions.

To set up overall assembly, user needs hardware

1. iSobot A &iSobotB
2. Arduino board controller with USB B connector cable.
3. Chameleon Light controller
4. Power Management Unit (to be attached to Chameleon controller)
5. 5 dimmable lights for stage lightning
6. Audio Speakers
7. Windows based Laptop / PC with software installed on it.

Notes

1. More number of iSobots can be kept, but remember that iSobot has only 2 modes. Hence, all iSobots in same mode will react same as long as they receive the instructions from the arduino controller.
2. iSobot needs 3 AAA batteries, make sure those are charged. iSobot may discharge them quickly like in 2 hours.

3. Arduino board should direct the infrared transmitter on top of iSobot where its IR senson is there. IR sensitivity is in short range so the arduino board should be within a meter range over iSobot head.
4. Chameleon light controller and the power management unit should be connected through serial RS232 cable while in power off mode only. Do not plug / unplug it while power is on otherwise the unit may get damaged. Its setup is in Appendix F.
5. User can choose different kind of lights but should make sure those are dimmable and won't blow up with variable voltage level.

Cost

#	Item	Quantity	Price ($)
1	iSobot	2	160
2	Chamelon controller & Power Management Unit	1	600
3	Lights & connectors	5	~20 – 80
4	Arduino board, components & box	1	40
5	Audio Speakers	1	~20
6	3A batteries (optional)	3 / iSobot	~10

Table 7-1Hardware item cost

Software Manual

The software package includes:

MotionEditor.exe	Application file
commandsA.csv, commandsB.csv	Excel file for iSobot commands database
1.wmv – 140.wmv	A folder has all videos of iSobot
*.bin	Binary files to store software data

Table 7-2List of files

Editor

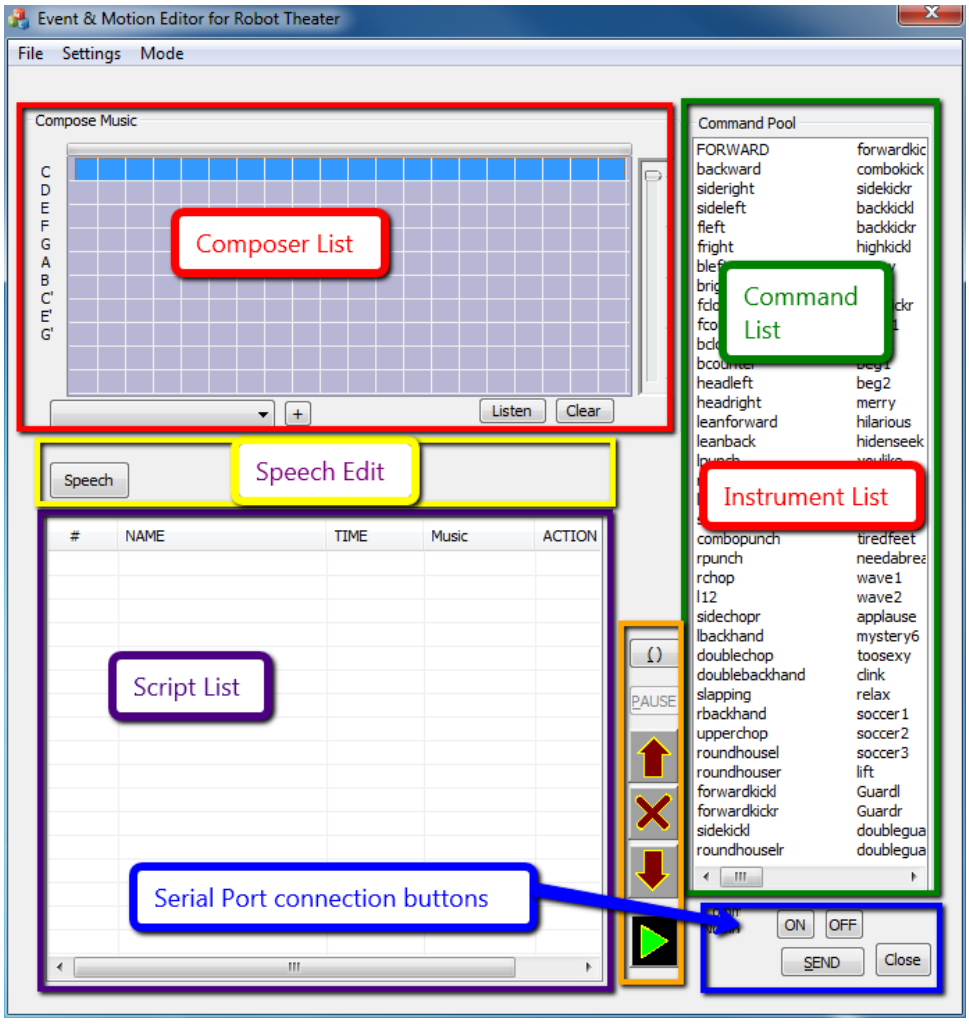


Figure 7:1Motion Editor GUI

Choose right mode

There are 3 modes among which, user can select only 1 at a time. These 3 modes are:

Compose mode:

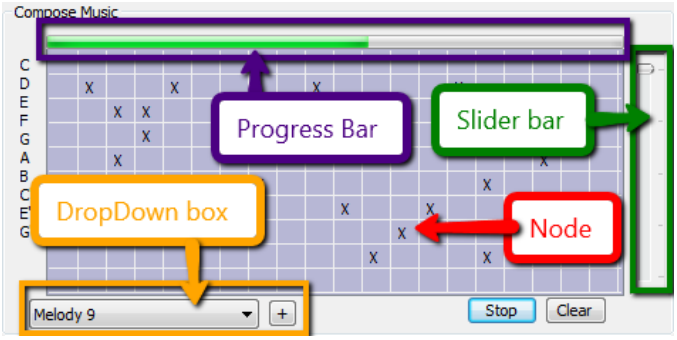


Figure 7:2MIDI Composer

a. User can compose the melody by single clicking Nodes (individual boxes) in Composer List.
b. Create new melody & save it by clicking on '+' button.
c. Load the saved melody in composer list by choosing one from dropdown list.
d. Listen to the melody edited/loaded in Compose list.
e. Command list populates with list of Instruments.
f. Clear the Composer List with clear button to create new one.

Action mode:

In this mode,

a. Command list box is first populates with list of iSobot commands.
b. When user double clicks on a command from Command List, embedded Windows Media Player opens up in another pop up window & plays the video of the selected iSobot command.
c. Make sure that user closes the current pop up window by clicking ‘X’ button else can’t choose and play another command to preview.
d. User can select individual command and click the send button to test command on the iSobot.

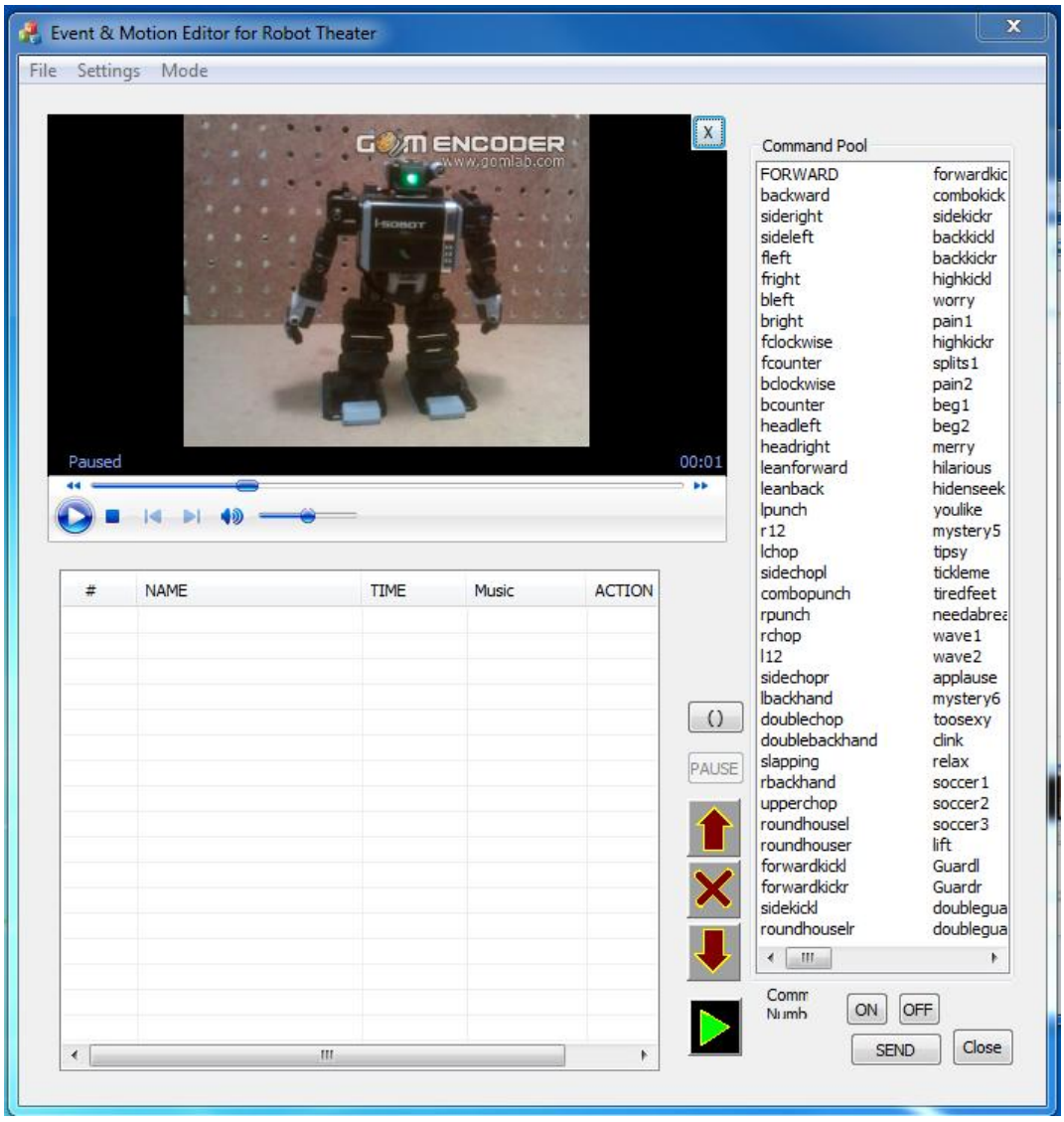


Figure 7:3Embedded Windows Media Player

Script mode:

a. Command list box populates with iSobot commands.

b. User can add commands in the script list by double clicking commands in Command List. Number of commands in this list creates a Script.

c. User can Save the created script by clicking on File > Save. User can save the script in windows file system. It is better that user saves all scripts in Script folder so that later those are easy to be find.

d. User can load the saved commands by clicking on File > Load.

e. Clicking on File > New will clear all commands in Script List.

f. User can choose the order of commands in the Script List by clicking on the buttons.

g. Script can play all commands in the Script list by clicking on Play button.

h. To group the number of commands in the Script, choose multiple commands in the Script List, and then click on button '( )', which group those selected commands.

i. To insert a Pause of 5 seconds in the script, click on a row in Script List and then click on 'Pause' button. The pause instruction gets added before the selected row.

j. To insert speech in the script list, user selects a row and then clicks on 'Speech' button. After editing the text in the speech text box, user can click

'Insert' button to add the speech text after the selected row in Script List.

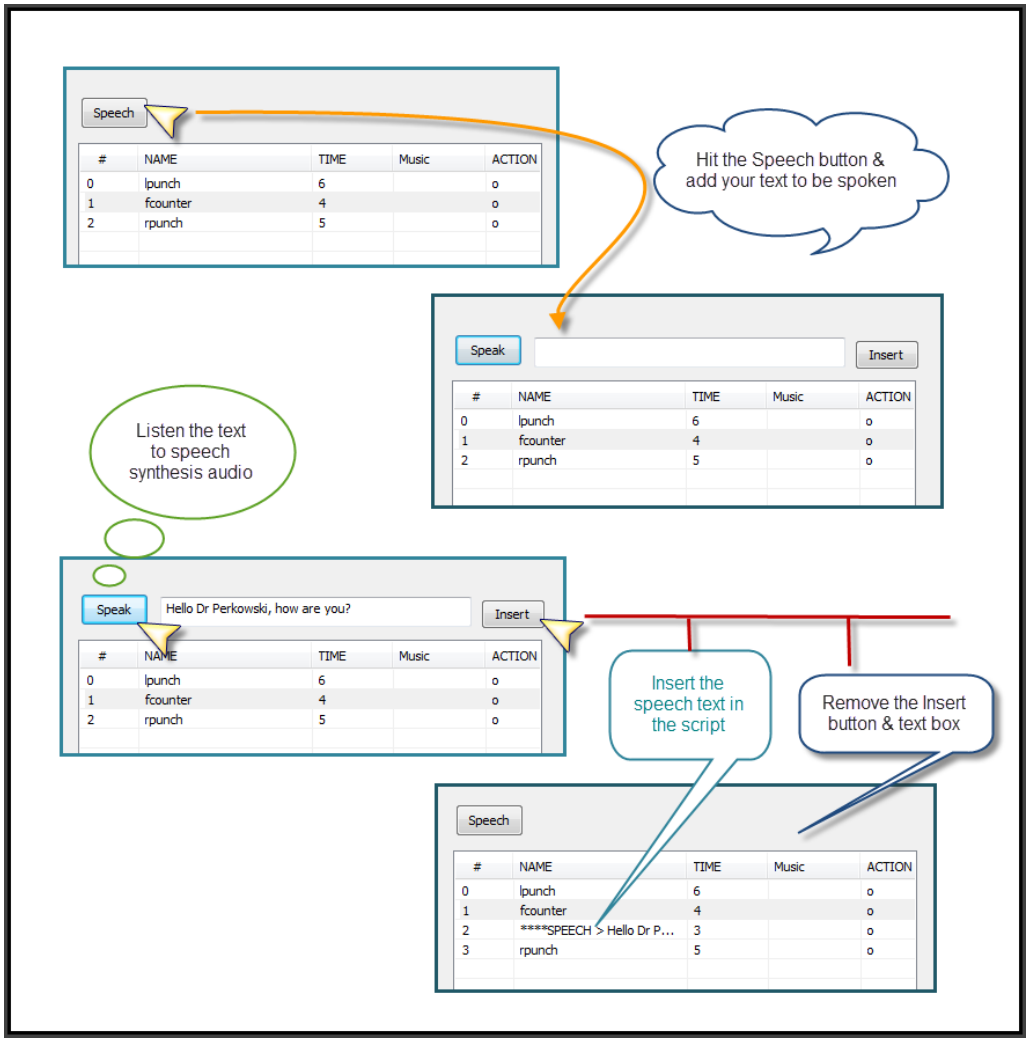


Figure 7:4Add speech text withing script list

k. User can edit the expression by clicking in 'Aciton' column in a row. This opens'Expression List'. Clicking on one of the expression from Expression list adds that Expression in the Action column in the selected row.

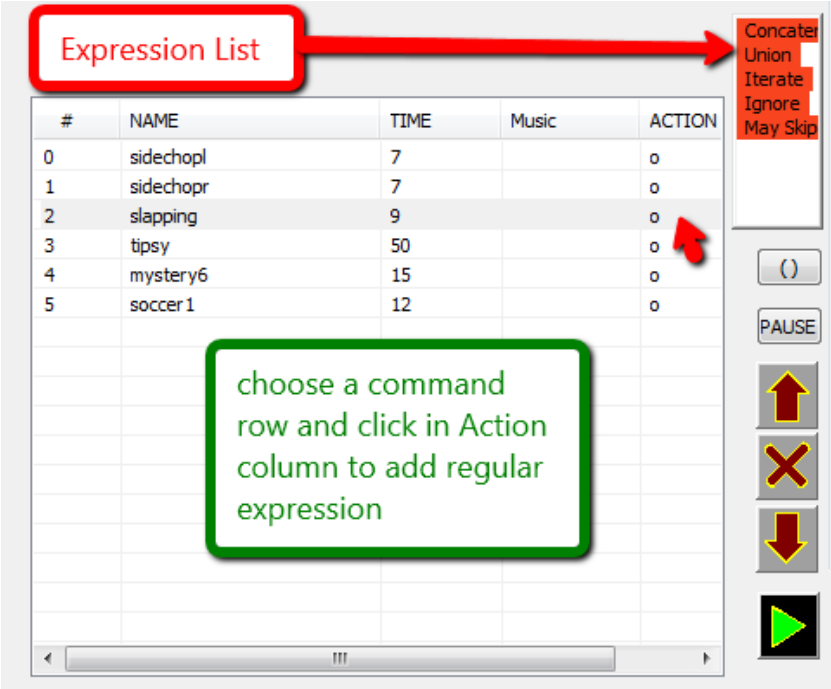


Figure 7:5 Way to add Expression in Script List

l. To add the melodies in Script list, user clicks in 3rd column named ‘Music’ in a row. This opens dropdown box in Composer list and user can select a melody from that list.

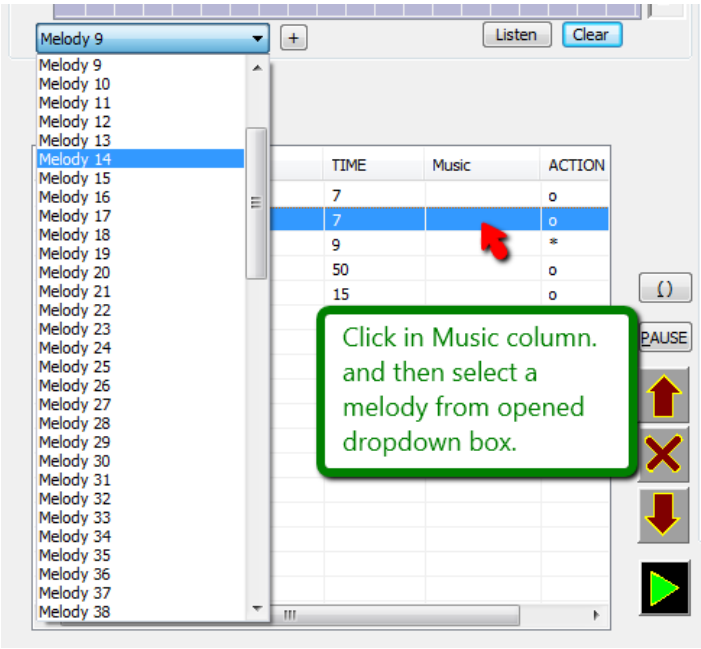


Figure 7:6 Way to add Melodies in Script

Chapter 8 Comparison, Feature Advancement Ideas / Future Work

It is important to study the worth of the product, if it really stands out lucrative.

In order to compare, there must be a similar product, but I never found any such editor available in the market, which is dedicated for robot's theater and can control all light, music, and speech along with robots. But, there are some Motion Editors, which works only for particular robot.

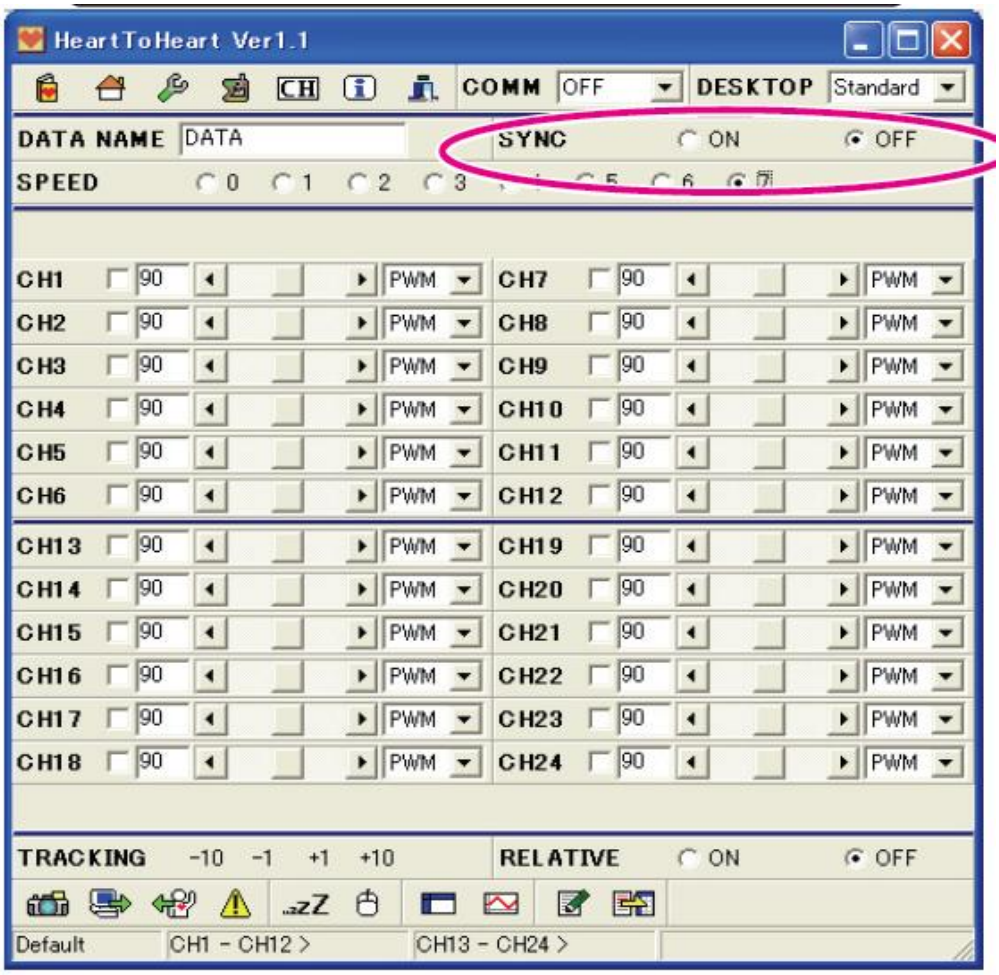


Figure 8:1Heart to Heart Editor for KHR1 Robot

To design an editor which controls every single robot in the world is virtually impossible. Different kinds of robots have their own features, functions, different style of connectivity, varieties of elements, their different DOFs (Degree of Freedom). One wouldn't wonder to know that several new robots born every day. Thousands of researchers are investigating new innovative ideas and trying to bring up a robot which can virtually do all possible work that human kinds do. So, some of the designer's effort were to collaborate library of several robots' action commands and control those through single editor.

In this editor, the approach is more dynamic to make it universal, make it work for many more robots provided the user has command library of those robots. As discussed in chapter 6, user can manage the command library and control the robots other than iSobot too. All one need to know is the commands required to control the new robot, and edit those in simple Excel file. Save that Excel file in '.CSV' format and link that file in editor. The new robot is ready to use to be controlled through this software, if user takes care of hardware connectivity.

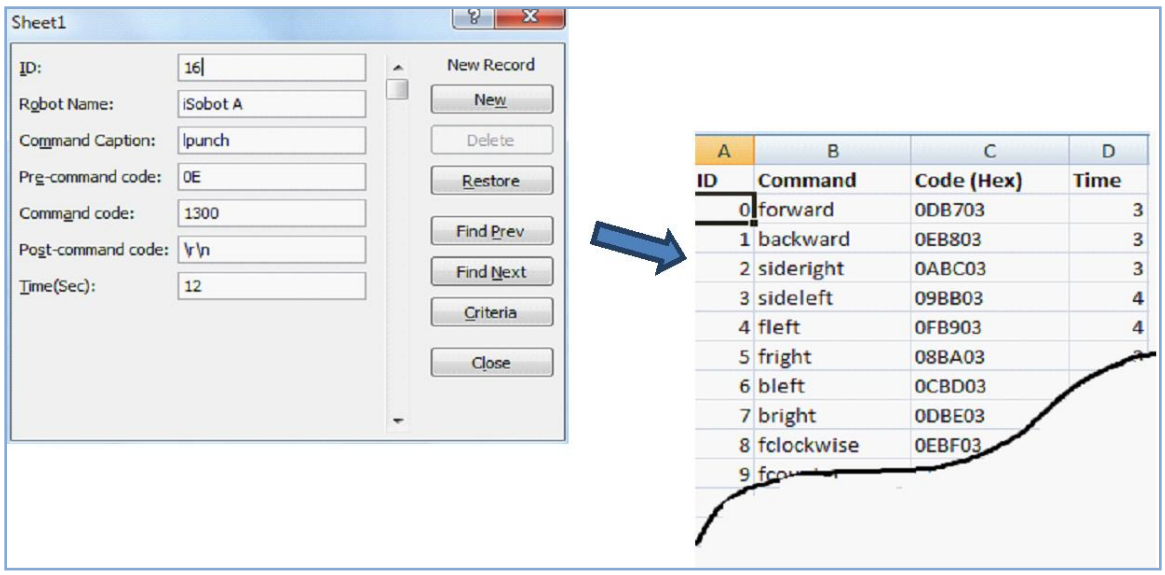


Figure 8:2Flexibility of adding new robot with instruction set using simple Excel datasheet

The iSobot was the cheapest one, readily available, works flawlessly and can be controlled wirelessly, hence was chosen for this project. We also had KHR1 and had build editor for it in Python but most of the time; it was breaking wires, falling down and wearing out some connections. Most of our time was spent in fixing it rather than working on it, controlling it through software. This KHR1 robot works great, but if it comes with better-upgraded version, future students can use it with this editor too.

Creating a speech editor is not a big victory, but adding its interface along with robot play can be innovative. Not all robots can talk or make sound like this cute iSobot does. Hence, sometime, robot actions are not well expressed. A speech announcement can be helpful to tell what robot is doing which makes robot action more understandable and joyful. On other hand, the speech can work as a robot mouth too. For mute robots like

KHR1, or Hahoe robots, those can make just lip movements & some gestures; the speech from the background would sound like the robot speaking.

The speech editor rather can be enhanced within this editor by adding multiple voices for e.g. male voice, female voice.

Being able to preview the robot action within the editor was possible with embedding Media Player. User does not need to plug in all hardware and work with actual robot all the time. Just by watching the video of robot action, user can decide whether he wants to send the same command to robot. This not only saves time but also stops discharging robot batteries. This media player turned out to be helpful to quickly build action script for robot play which hasn't found in other editors. There is a great tools called Grasp Editor [25], a modeling tool that allowing to create new models of robot hands.

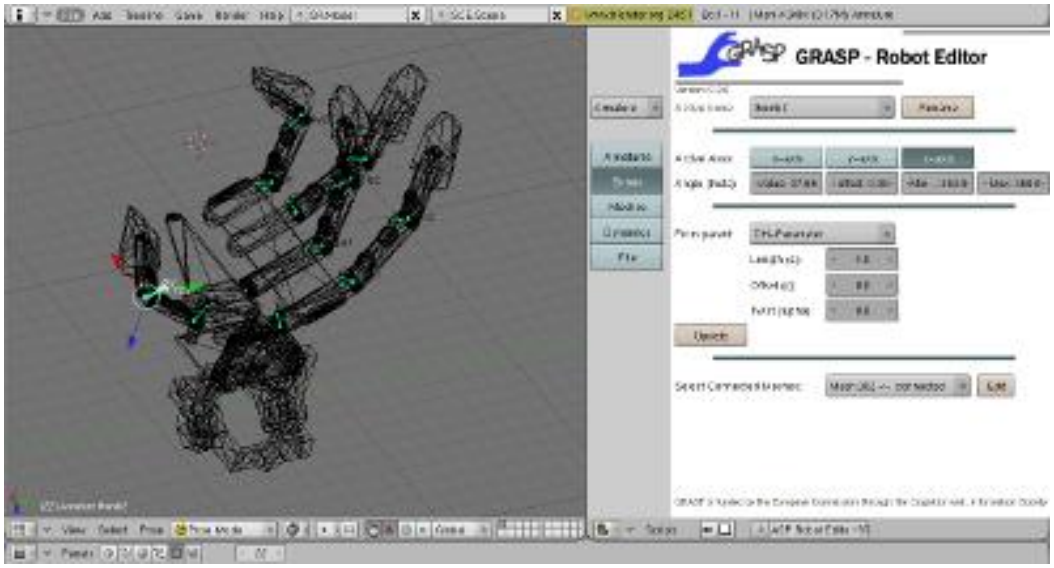


Figure 8:3Graphical Editor for editing micro actions of robot

The tool uses object library, and lets the user move the objects within the graph using mouse click n drag. Such tools are good for creating micro actions of robot. For the robots which already have such actions ready as a library, only the video preview is enough to see the action. For this particular iSobotrobot, user can not edit the micro actions.

Controlling lights in sync with music has several ways. There are several inspiring videos on YouTube with numerous innovative ideas how people synchronized water fountain, or flashing lights in sync with the music. Similarly, there is no limit to user's creativity for making sound using simple MIDI technology. This editor combined both ideas and compressed to design an easy interface for user to create sound quickly and make the stage lights play in sync with it. The editor lets user create the sound melody with 128 different instruments, 5 different scales, save the 100 melodies for reuse.

There are other software's to edit the sound from wave files or recorded music. Wavepad is a feature rich sound editor for Windows. One can record from audio inputs, like a microphone or a record player, import and edit sound files. It is good for making mixes, or digitizing vinyl.

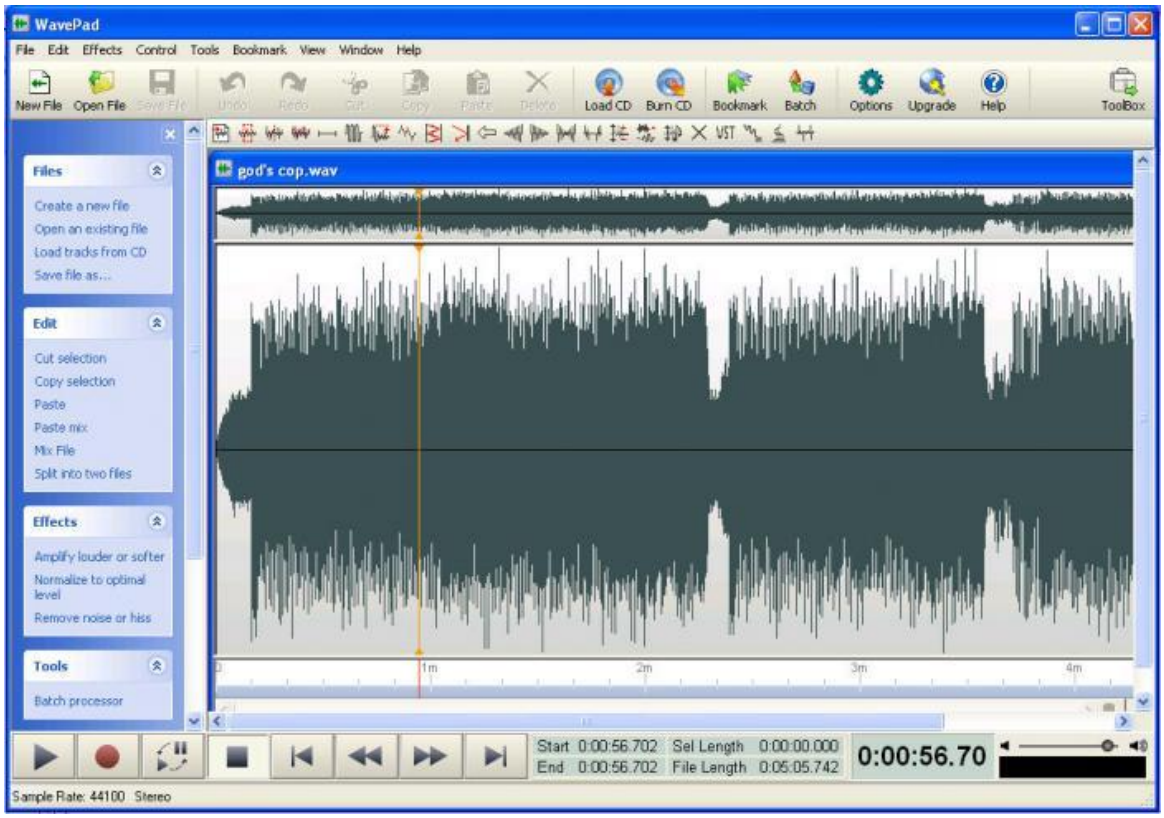


Figure 8:4Sound Editor Software

The Figure 8-4 is pretty self-explanatory what the software can do. Yet, this software needs some wave files already present in the computer that user can copy one or more and mix n edit within. It is not meant to create a melody, or music from the scratch. Our purpose of melody composer with MIDI was to produce sound as well as to control the lights.

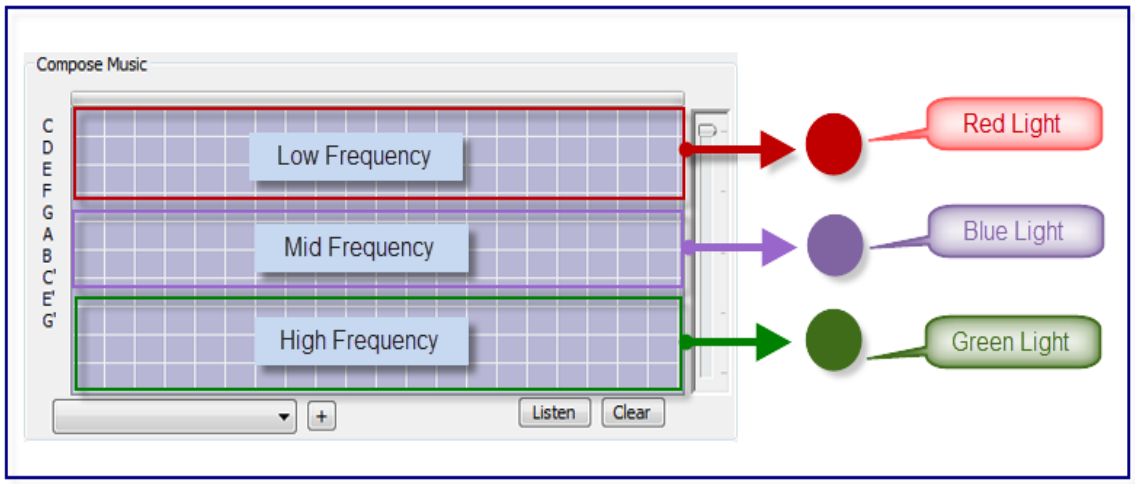


Figure 8:5MIDI Composer has different frequency range, lighting up different lights

Hence, by editing the midi notes within the composer list is indirectly intended to control lights. This is difficult to achieve through Wavepad sound editor. Our sound editor can edit the melody that runs for 4 seconds and it repeats until switched. This may make it sound boring if played for very long. User has always an option to play music files stored in PC. For this, there is not even need of any interface within editor; Windows Media Player is present in almost all PCs.

Feature Advancements, Ideas for Future developers

The editor is a good starting point for developers, and opens the door for much more creativity in the area of robotic theater.

1. The MIDI interface in this editor provides all musical instruments but user is constrained to use only one at a time. It would probably be cool if a melody can

be composed with multiple instruments playing all at a time, and with different scale range. With the current MIDI interface, Only 2 to 3 lights can be controlled with chameleon musical light controller, but with suggested advancement, user will see all lights flashing up.

Also, the melody time is limited to 4 seconds only. With a little work around it, developer can provide more advanced melody composer with variable time, Tempo adjustment. In order to avoid the complexity, the whole MIDI composer could be designed in new dialog window and save the melody within that dialog window. In the Script editor, user would just load those melodies and play. An idea of such interface is shown in Figure 8-4. Remember, this is just an idea, user's creativity has no barrier.

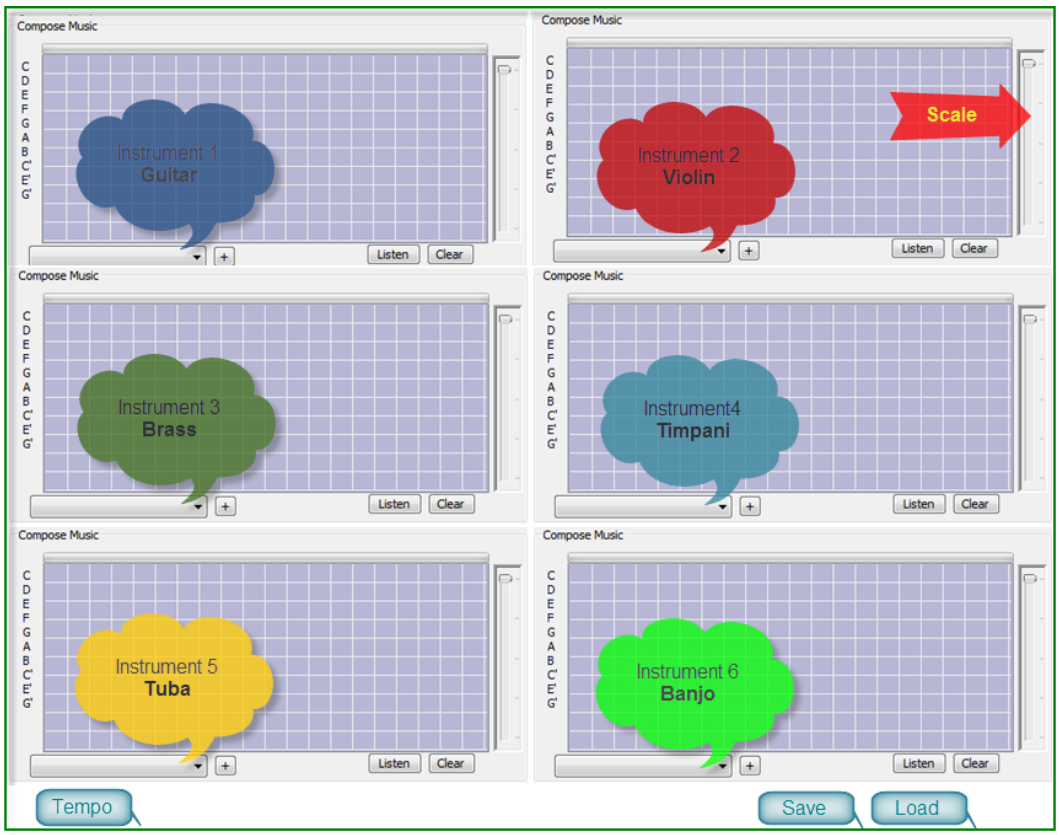


Figure 8:6 Idea for a MIDI interface with different instrument selection option

2. Chameleon Light controller is a ready-made product designed to control the lights in sync with any music. Its inbuilt DSP chip does sound processing and generate appropriate Pulse Width Modulated [PWM] signal to vary the light intensity. It's not designed so as to work ideally with MIDI composer in this editor. The current MIDI interface can only switch up to 2 3 lights but a new separate controller can be designed specially made for it. The controller would be still separate from the main GUI, and would accept direct MIDI messages rather than Audio Input signal. The light controlling would be handled with directly with these messages,

so it would be dedicated only for such MIDI interface and cannot be used for any other general Music like Chameleon does. But, advantage would be that user can gain full access over all connected lights with the controller and set up individual thresholds for turning them ON / OFF, or vary intensity. Chameleon first inputs the audio, decodes it, process & then send out PWM signals.

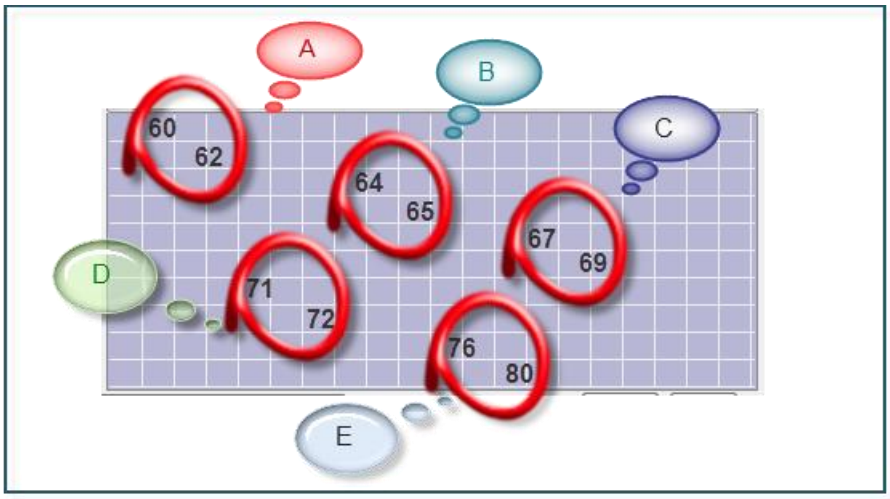


Figure 8:7Use of MIDI notes for controlling lights

Chameleon lighting is frequency range dependant. We can make lightning intensity variation with absolute MIDI note points. As can be seen in the diagram, the MIDI notes can be divided in range like 60 – 63, 64 – 66 likewise. Whenever a note within this range would play, associated light bulb would increase intensity by one count for certain period say 500mSec. If within that time period, another MIDI note within that range hits again, the count will increase by one again. Hence, continuous MIDI note playing within same range would go on increasing

the light intensity, otherwise decreasing. This how, light intensity can be controlled.

3. Can add Drag and Drop functionality in the interface.

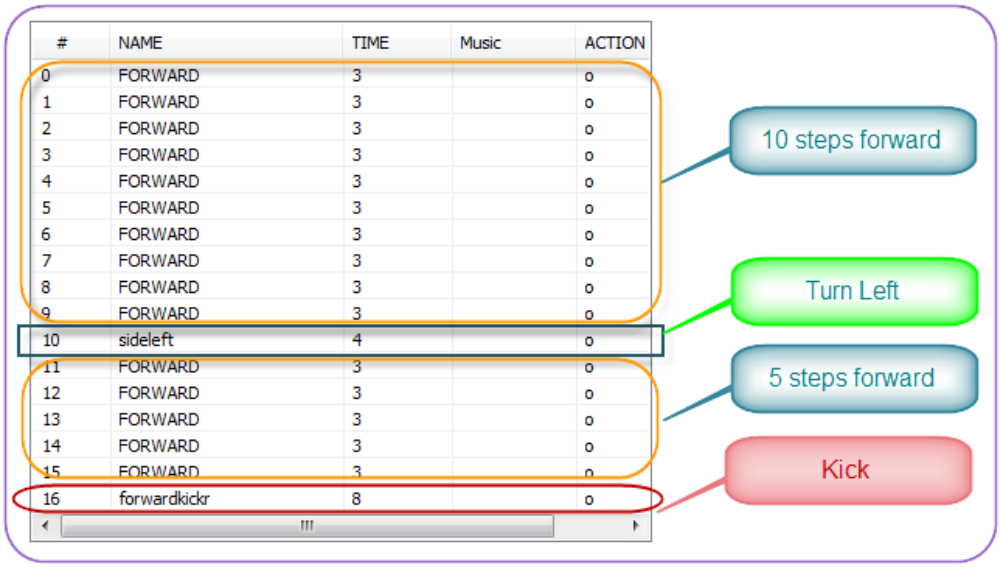
4. DMX lightning systems are also the area to research for musical lighting controller. These systems have DMX connection (a serial interface) and a command protocol. One of these systems might be a better fit than chameleon because they would allow direct control of lights.

5. With this editor, user can only send the commands to the robot but there is no feedback recognition. Future developers can add sensors, and use algorithms to process the data in background to result some signal. The signal would be used as an event. And GUI is nothing but event driven programming. A handler routine would take care of such generated event. This further can be used along with regular expressions to create more environment interactive robot action's play script.

   For e.g.

   Current Script can instruct robot to walk forward for 10 steps. Hopefully it

reaches near to the wall. Then turn left. Then walk forward 5 steps. Kick the ball.



#	NAME	TIME	Music	ACTION
0	FORWARD	3		o
1	FORWARD	3		o
2	FORWARD	3		o
3	FORWARD	3		o
4	FORWARD	3		o
5	FORWARD	3		o
6	FORWARD	3		o
7	FORWARD	3		o
8	FORWARD	3		o
9	FORWARD	3		o
10	sideleft	4		o
11	FORWARD	3		o
12	FORWARD	3		o
13	FORWARD	3		o
14	FORWARD	3		o
15	FORWARD	3		o
16	forwardkickr	8		o

Figure 8:8 Editor with no sensor events, no feedback from environment

With addition of a camera, or some kind of proximity sensor, user can recognize if robot is stepping into the wall, stop it one step prior to it. Then turn left and walk forward until it reaches to the ball.

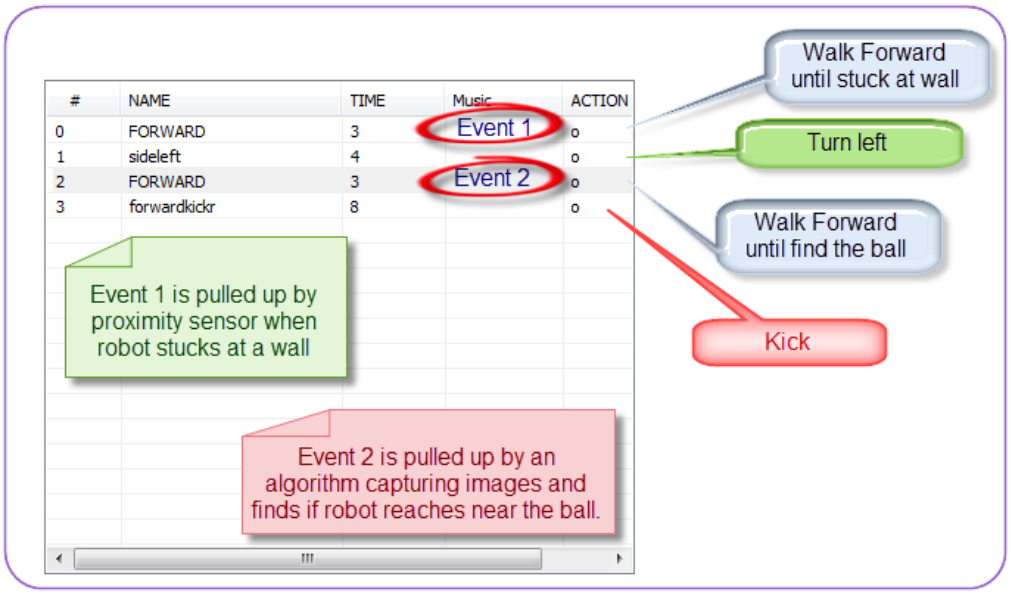


Figure 8:9Editor handling sensor data events

6. The editor can add Bluetooth interface to be able to communicate with mobile device. Recently born Bluetooth technology grown up pretty fast and now available to use to many devices with different profiles. If developer can add this interface in current editor, then he can use mobile device to control the robots indirectly through the editor in PC. Developing applications on Smartphone has become fairly easy to those having even basic knowledge of simple C, JAVA & GUI programming. One can quickly design an app for such interface in mobile device and then the app can be used as a remote controller for editor. Advantage

of this is those users can use mobile gyroscope and turn / move robot with smooth handling.

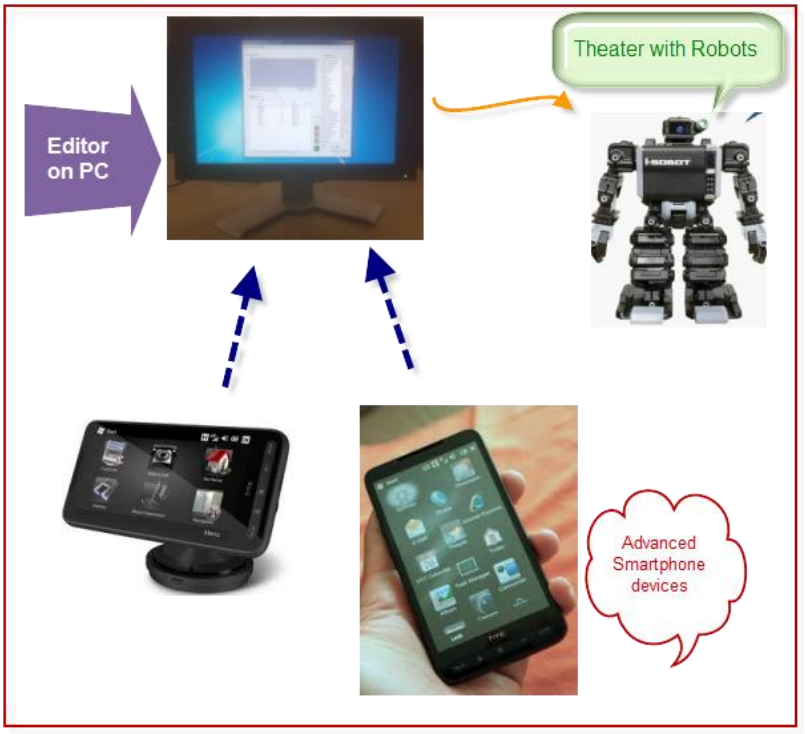


Figure 8:10Editor can be further extended to be controlled through smartphones

7. In this editor, the arduino board is used only to receive hex commands code and send the bit pattern to iSobot after converting it into IR signal. Rest, music is generated within software using MIDI library and the light controller is another microcontroller which acts independently. It would be possible to program an ardunio board which acts for multiple serial port interfaces. This way, lights would be possibly controlled through arduino board only. The central controller

board would receive commands from not only the computer but also the get the feedback from external sensors, or Bluetooth devices like Smart-phones.

Chapter 9 CONCLUSIONS

In this thesis I created an initial theory of how one can create sets of motions of a biped robot for entertainment. Based on extended Brzozowski's method, robot behaviors are converted to finite state machines.

In the practical part of my thesis, I designed an editor to provide a complete graphical user interface to control the entire Robot Theater with robots, lighting, and speech synthesis. This is the first attempt ever to create such an universal language-based editor. With this editor one can not only control the robots but also the lights, background music and speech announcements. In future this approach can be extended to control motors, fog machines and other equipment used in theatres and discotecques. These can be set to be played in real time and in sync. This way, user can create numerous scripts to be played quickly. These scripts can be created, saved and loaded later to use.

Interestingly, the script is not monotonous. Using regular expressions, the script is made dynamic so there is variety of events to be appreciated by the viewers.

REFERENCES

1. http://en.wikipedia.org/wiki/Entertainment_robot
2. http://en.wikipedia.org/wiki/Humanoid_robot
3. http://en.wikipedia.org/wiki/KHR-1
4. http://en.wikipedia.org/wiki/I-SOBOT#Omnibot_17.CE.BC:_i-SOBOT
5. http://en.wikipedia.org/wiki/Theatre_lighting
6. OkanArikanand D. A. Forsyth, Interactive Motion Generation from Examples, http://delivery.acm.org/10.1145/570000/566606/p483-arikan.pdf?key1=566606&key2=1804502031&coll=DL&dl=ACM&ip=131.252.212.103&CFID=16740547&CFTOKEN=44113924
7. http://en.wikipedia.org/wiki/Musical_Instrument_Digital_Interface
8. Jack's **MIDI Music** Home Page. A Great Starting Point For Those Interested In **MIDI Music** Of Many Various Styles With Quality Links, http://www.ajsmidi.com/
9. Algorithm for converting a finite state machine into a regular expression, http://qntm.org/algo
10. Regular Expressions Matching in less that 100 lines of code, http://michid.wordpress.com/2010/12/06/regular-expression-matching-in-100-lines-of-code/
11. JanuszBrzozowski, Derivatives of Regular Expressions, http://delivery.acm.org/10.1145/330000/321249/p481-

brzozowski.pdf?key1=321249&key2=8654502031&coll=DL&dl=ACM&ip=131.252.212.103&CFID=16740547&CFTOKEN=44113924

12. Deterministic finite-state machine ,http://www.scribd.com/doc/19699865/Expression
13. ZviKohavi, Switching and Finite Automata Theory.
14. Laban Notation, http://en.wikipedia.org/wiki/Laban_Movement_Analysis
15. **Twelve Basic Principles of Animation**, http://en.wikipedia.org/wiki/12_basic_principles_of_animation
16. Ollie Johnston and Frank Thomas, *The Illusion of Life: Disney Animation*,1981.
17. Subsumption Architecture, http://ai.eecs.umich.edu/cogarch2/specific/subsumption.html
18. http://en.wikipedia.org/wiki/Hidden_Markov_model
19. http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1237771631
20. http://en.wikipedia.org/wiki/RoBo-One
21. http://www.isobotrobot.com/eng/about/index.html
22. http://www.magnevation.com/
23. http://www.magnevation.com/pdfs/speakjetusermanual.pdf
24. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=01423187&tag=1
25. http://opengrasp.sourceforge.net/robotEditor.html#robot-editor
26. http://www.atmel.com/dyn/resources/prod_documents/doc8025.pdf
27. http://msdn.microsoft.com/en-us/
28. http://msdn.microsoft.com/en-us/library/dd563023(v=VS.85).aspx

29. http://msdn.microsoft.com/en-us/library/hedkd737(VS.80).aspx

30. http://msdn.microsoft.com/en-US/library/59wbz5wc(v=VS.80).aspx

31. J. A. Brzozowski. Derivatives of Regular Expressions. Journal of the ACM (JACM)

    Vol. 11, Issue 4, October 1964

32. M. Lukac, Ph.D. Thesis, PSU 2009.

33. R. Laban. The Mastery of Movement, *Macdonald and Evans*; 4th ed.,revised and enlarged edition, 1980.

34. R. Franklin, D. Carver and B. L. Hutchings. Assisting Network Intrusion Detection with Reconfigurable Hardware.Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, p.111, 2002

35. R. Sidhu and V. K. Prasanna. Fast Regular Expression Matching using FPGAs. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), Apr. 2001, 12 pages.

36. ESRA robot editor. Look for VSA for the robot: http://www.robodyssey.com/index.htm

37. A. Raghuvanshi and M. Perkowski, "Using Fuzzy Quantum Logic to learn facial gestures of a Schrödinger Cat puppet for robot theatre,"*Proceedings of ULSI 2008.*

38. Q. Williams, S. Bogner, M. Kelley, C. Castillo, M. Lukac, D. H. Kim, J. Allen, M. Sunardi, S. Hossain, and M. Perkowski. "An Emotional Mimicking Humanoid

Biped Robot and Its Quantum Control Based on the Constraint Satisfaction Model," *dwave.files.wordpress.com/2008/09/ulsi-2007-robot-1.doc* –

39. M. Sunardi, M.S. Thesis in preparation, PSU, 2010. [6] R. Callois. Les Jeux et les Hommes, Paris: Callimard. 1958.

    [7] R. Laban. The Mastery of Movement, *Macdonald and Evans*; 4th ed., revisedand enlarged edition, 1980.

40. M.A. Perkowski, T. Sasao, J-H Kim, M. Lukac, J. Allen, S. Gebauer: Hahoe KAIST Robot Theatre: Learning Rules of Interactive Robot Behavior as a Multiple-Valued Logic Synthesis Problem. Proc. ISMVL 2005: pp. 236-248
41. M. Perkowski, M. Lukac, S. Gebauer, M. Sunardi, Introduction to Robot Theatre, book in preparation. [9] S. Nakaoka, A. Nakazawa, K. Yokoi, H. Hirukawa and K. Ikeuchi. Generating Whole Body Motions for a Biped Humanoid Robot from Captured Human Dances, *IEEE 2003 International Conference on Robotics and Automation*, 2003.
42. http://msdn.microsoft.com/en-us/library/y04ez4c9.aspx
43. Library for Serial Port Interface: http://www.codeproject.com/KB/system/cserialport.aspx
44. W. B. Kleijn and K. K. Paliwal (ed.), *Speech Coding and Synthesis*,Elsevier, 1995.
45. D. Golubovic, B. Li and H. Hu. A Hybrid Software Platform for Sony AIBO Robots, *RoboCup 2003: Robot Soccer World Cup VII*, pp. 478-486, 2003.

46. T. Ishimura, T. Kato, T. Oda and T. Ohashi. An Open Robot Simulator Environment, *RoboCup 2003: Robot Soccer World Cup VII*, pp. 621-627, 2003.
47. A. Kerepesi, E. Kubinyi, G. K. Jonsson, M. S. Magnusson and A. Kiklosi. Behavioural Comparison of Human-Animal (Dog) and Human-Robot (AIBO) Interactions, B*ehavioural Processes*, vol. 73, no.1, pp. 92-99, 2006.
48. T. Wama, M. Higuchi, H. Sakamoto and R. Nakatsu. Realization of Tai-chi Motion Using a Humanoid Robot, *Entertainment Computing*, Springer LNCS, pp. 14-19, 2004.
49. Aichi EXPO http://www.expo2005.or.jp/en/index.html

APPENDICES

Appendix A iSobot Command Set

Followings are the set of commands which we are using to control iSobot in mode A.

Command	Code (Hex)
Forward	0DB703
Backward	0EB803
sideright	0ABC03
sideleft	09BB03
fleft	0FB903
fright	08BA03
bleft	0CBD03
bright	0DBE03
fclockwise	0EBF03
fcounter	0FC003
bclockwise	08C103
bcounter	09C203
headleft	0DD707
headright	0BD70C
leanforward	0CD706
leanback	0FD701

Command	Code (Hex)
Guardl	093200
Guardr	0A3300
doubleguard1	0B3400
doubleguard2	0C3500
dodgel	0D3600
dodger	0E3700
duck	093800
swayback	0A3900
upblock	0B3A00
splits2	0C3B00
comboblock	0D3C00
zero	0FCA00
homeposition	0BD500
soundoff	0CD303
affirm	083F00
disagree	0C4200

lpunch	0E1300
r12	0A1700
lchop	0D1900
sidechopl	0A1E00
combopunch	091D00
rpunch	0F1400
rchop	0E1A00
l12	0C1800
sidechopr	0B1F00
lbackhand	081500
doublechop	0F1B00
doublebackhand	0E2100
slapping	0D2000
rbackhand	091600
upperchop	081C00
roundhousel	0F2200
roundhouser	082300
forwardkickl	092400
forwardkickr	0A2500
sidekickl	0B2600
roundhouselr	0A2C00

goodmorning	0E4400
greet1	0F4500
greet2	094700
greet3	0B4800
greet4	0C4900
bye1	0D4A00
bye2	0E4B00
bye3	0F4C00
bye4	084D00
bye5	084600
respect	0D4300
thanks1	094E00
thanks2	0A4F00
love1	0D5100
love2	0E5200
love3	0F5300
standupfront	0E3D00
standupback	0F3E00
excited1	0B5700
excited2	0D5800
excited3	0E5900

forwardkicklr	0B2D00
combokick	0C2E00
sidekickr	0C2700
backkickl	0E2800
backkickr	0F2900
highkickl	0D2F00
worry	0A5D00
pain1	0F6100
highkickr	0F3000
splits1	083100
pain2	096300
beg1	0E6000
beg2	0D6E00
merry	087000
hilarious	0F7700
hidenseek	095C00
youlike	0A6B00
mystery5	0B6C00
tipsy	0C6D00
tickleme	0A7800
tiredfeet	0B7900

excited4	097100
party	0A5600
amazed	0B7300
regret1	085B00
regret2	0B5E00
regret3	0C5F00
countonme	0E8300
articulation	0F8400
showoff1	088500
showoff2	098600
showoff3	0A8700
showoff4	0C8800
cominthrough	0D8900
catch	0F5A00
pose1	0BC700
pose2	0DC800
pose3	0EC900
mystery1	0A7200
mystery2	0C7400
mystery3	0D7500
mystery4	086200

needabreak	0C7A00
wave1	0D7B00
wave2	0E7C00
applause	0E7600
mystery6	0E6F00
toosexy	0F7D00
clink	087E00
relax	0B8000
soccer1	0D8200
soccer2	092B00
soccer3	082A00
lift	0C8100

forwardsomersault	0E8A00
headstandexercises	0F8B00
exercises	088C00
airdrum	098D00
airguitar	0A8E00
randomperformance1	0E9100
randomanimal	099400
tropicaldance	0C9700
giantrobot	0E9800
western	0F9900
randomperformance2	099B00

i-SOBOT™ Action Table

Description of icons

P ···Punch button · K ···Kick button · G ···Guard button · GO ···Execute button
1 ···1 button · 2 ···2 button · 3 ···3 button · 4 ···4 button
A ···A button · B ···B button · L ···Left joystick · R ···Right joystick

TOMY®

For combinations of multiple buttons, input in order from the left.

Remote control mode

Joystick Control

No	Action	Input & Movement	LCD
1	Walk Forward	L ⇧	
2	Walk Backward	L ⇩	
3	Walk Fwd Curve Left	L ↖	
4	Walk Fwd Curve Right	L ↗	
5	Sidestep Left	L ⇦	
6	Sidestep Right	L ⇨	
7	Walk Bkwd Curve Left	L ↙	
8	Walk Bkwd Curve Right	L ↘	
9	Rotate Fwd Clockwise	L ⟲ R ⇦	
10	Rotate Fwd Counter-CW	L ⟳ R ⇨	
11	Rotate Bkwd Clockwise	L ⟲ R ⇦	
12	Rotate Bkwd Counter-CW	L ⟳ R ⇨	
13	Move Arms	R button + L R	ARMS
14	Lock/Unlock Arms	L button (R button + L R)	ARMS
15	Turn Head Left	R ⇦	
16	Turn Head Right	R ⇨	
17	Lean Forward	R ⇧	
18	Lean Backward	R ⇩	

Punch Actions

No	Action	Input	LCD
19	Punch (L)	1 P	1.P
20	Punch (R)	2 P	2.P
21	Backhand (L)	3 P	3.P
22	Backhand (R)	4 P	4.P
23	One-Two Punch (R)	1 2 P	1.2 P
24	One-Two Punch (L)	2 1 P	2.1 P
25	Chop (L)	1 1 P	1.1 P
26	Chop (R)	2 2 P	2.2 P
27	Double Chop	3 3 P	3.3 P
28	Upper Chop	4 4 P	4.4 P
29	Combo Punch	1 2 3 4 P	1.2 3.4.P
30	Side Chop (L)	1 4 P	1.4 P
31	Side Chop (R)	2 3 P	2.3 P
32	Slapping	4 3 P	4.3 P
33	Double Backhand	3 4 P	3.4 P

Programming Pause

No	Action	Input	LCD
34	3 Second Pause	4 4 4 A	4.4. 4.A

Kick Actions

No	Action	Input	LCD
35	Roundhouse (L)	1 K	1.K
36	Roundhouse (R)	2 K	2.K
37	Forward Kick (L)	3 K	3.K
38	Forward Kick (R)	4 K	4.K
39	Side Kick (L)	1 1 K	1.1 K
40	Side Kick (R)	2 2 K	2.2 K
41	Back Kick (L)	3 3 K	3.3 K
42	Back Kick (R)	4 4 K	4.4 K
43	Roundhouse (L & R)	1 2 K	1.2 K
44	Fwd Kick (L & R)	2 1 K	2.1 K
45	Combo Kick	1 2 3 4 K	1.2. 3.4.K
46	High Kick (L)	1 3 K	1.3 K
47	High Kick (R)	2 4 K	2.4 K
48	Splits 1	3 4 K	3.4 K

Utility Action

No	Action	Input	LCD
49	Zero Position	4 4 4 B	4.4. 4.B

Action 49 is normally used only for maintenance

Guard Actions

No	Action	Input	LCD
50	Guard (L)	1 G	1.G
51	Guard (R)	2 G	2.G
52	2-Hand Guard 1	3 G	3.G
53	2-Hand Guard 2	4 G	4.G
54	Dodge Left	1 1 G	1.1 G
55	Dodge Right	2 2 G	2.2. G
56	Duck	3 3 G	3.3. G
57	Sway Back	4 4 G	4.4. G
58	Up-sweep Block	1 2 G	1.2. G
59	Splits 2	3 4 G	3.4 G
60	Combo Guard	1 2 3 4 G	1.2. 3.4.G

Stand up from a Prone Position

No	Action	Input	LCD
61	Stand up (if face down)	A	A
62	Stand up (if face up)	B	B

"Do not use Actions 61 & 62 unless i-SOBOT is lying down as noted. If used when standing, i-SOBOT will fall abruptly, risking damage."

Common Phrases & Greetings

No	Action	Input	LCD
63	Affirm	1 A	1.A
64	Disagree	4 A	4.A
65	Good Morning	1 2 A	1.2. A
66	Greet 1	1 3 A	1.3. A
67	Greet 2	2 1 A	2.1. A
68	Greet 3	2 2 A	2.2. A
69	Greet 4	2 3 A	2.3. A
70	Bye 1	3 1 A	3.1. A
71	Bye 2	3 2 A	3.2. A
72	Bye 3	3 3 A	3.3. A
73	Bye 4	3 4 A	3.4. A
74	Bye 5	1 4 A	1.4. A
75	Respect	1 1 A	1.1. A
76	Thanks 1	4 1 A	4.1. A
77	Thanks 2	4 2 A	4.2. A
78	Love 1	1 1 1 A	1.1. 1.A.
79	Love 2	2 2 2 A	2.2. 2.A.
80	Love 3	3 3 3 A	3.3. 3.A.

i-SOBOT™ Action Table

TOMY®

Description of icons

P ···Punch button　K ···Kick button　G ···Guard button　GO ···Execute button
1 ···1 button　2 ···2 button　3 ···3 button　4 ···4 button
A ···A button　B ···B button　L ···Left joystick　R ···Right joystick

For combinations of multiple buttons, input in order from the left.

Remote Control Mode (cont.)

Emotional Actions

No	Action	Input	LCD
81	Excited 1	1 4 B	1.4. B.
82	Excited 2	2 1 B	2.1. B.
83	Excited 3	2 2 B	2.2. B.
84	Excited 4	1 2 2 B	1.2. 2.B.
85	Party	1 3 B	1.3. B.
86	Amazed	1 2 4 B	1.2. 4.B.
87	Regret 1	2 4 B	2.4. B.
88	Regret 2	3 3 B	3.3. B.
89	Regret 3	3 4 B	3.4. B.
90	Worry	3 2 B	3.2. B.
91	Pain 1	4 2 B	4.2. B.
92	Pain 2	4 4 B	4.4. B.
93	Beg 1	4 1 B	4.1. B.
94	Beg 2	1 1 3 B	1.1. 3.B.
95	Merry	1 2 1 B	1.2. 1.B.
96	Hilarious	1 3 4 B	1.3. 4.B.

Showcase Actions

No	Action	Input	LCD
97	Hide N Seek	3 1 B	3.1. B.
98	You Like?	1 2 4 A	1.2 4.A.
99	Mystery 5	1 3 1 A	1.3 1.A.
100	Tipsy	1 3 2 A	1.3. 2.A.
101	Tickle Me i-SOBOT	1 4 1 B	1.4. 1.B.
102	Tired Feet	1 4 2 B	1.4. 2.B.
103	Need a Break	1 4 3 B	1.4. 3.B.
104	Wave 1	1 4 4 B	1.4. 4.B.
105	Wave 2	2 1 1 B	2.1. 1.B.
106	Applause	1 3 3 B	1.3. 3.B.
107	Mystery 6	1 1 4 B	1.1. 4.B.
108	Too Sexy	2 1 2 B	2.1. 2.B.
109	Clink	2 1 3 B	2.1. 3.B.
110	Relax	2 2 1 B	2.2. 1.B.
111	Soccer 1	2 2 3 B	2.2. 3.B.
112	Soccer 2	4 2 K	4.2 K.
113	Soccer 3	3 1 K	3.1 K.

Showcase Actions (cont.)

No	Action	Input	LCD
114	Lift	2 2 2 B	2.2. 2.B.
115	Count on Me	2 2 4 B	2.2. 4.B.
116	Articulation	2 3 1 B	2.3. 1.B.
117	Show Off 1	2 3 2 B	2.3. 2.B.
118	Show Off 2	2 3 3 B	2.3. 3.B.
119	Show Off 3	2 3 4 B	2.3. 4.B.
120	Show Off 4	2 4 1 B	2.4. 1.B.
121	Comin' Through	2 4 2 B	2.4. 2.B.
122	Catch	2 3 B	2.3. B.
123	Pose 1	4 1 1 A	4.1 1.A.
124	Pose 2	4 1 2 A	4.1 2.A.
125	Pose 3	4 1 3 A	4.1 3.A.

Mystery Actions

No	Action	Input	LCD
126	Mystery 1	1 2 3 B	1.2. 3.B.
127	Mystery 2	1 3 1 B	1.3. 1.B.
128	Mystery 3	1 3 2 B	1.3. 2.B.
129	Mystery 4	4 3 B	4.3. B.

"Actions 99, 75 and 126–129 have no sound."

Special Action Mode

No	Action	Input	LCD
130	Forward Somersault	A GO	SA-01 A.
131	Headstand Exercises	B GO	SA-02 B.
132	Excercises	A B GO	SA-03 A.B.
133	Air Drum	A A A GO	SA-04 A.A.A.
134	Air Guitar	B B B GO	SA-05 B.B.B.
135	Random performance	B A B GO	SA-06 B.A.B.
136	Banzai! Japan Cheer 1		SA-07 B.A.B.
137	Japan Cheer 2		SA-08 B.A.B.
138	Randomly imitates an animal	A B B GO	SA-09 A.B.B.
139			SA-10 A.B.B.
140	Dog		SA-11 A.B.B.
141	Cat Eagle Rooster Gorilla		SA-12 A.B.B.
142			SA-13 A.B.B.
143	Tropical Dance	B B A GO	SA-14 B.B.A.
144	Giant Robot	A B A GO	SA-15 A.B.A.
145	Western Movie Scene	A B A B GO	SA-16 A.B.A.B.
146	Random Performance Martial Arts Tai Chi	A A A A GO	SA-17 A.A.A.A.
147			SA-18 A.A.A.A.

Voice Control Mode

No	Voice Command	Action	LCD
148	*Go Forward*	Walks Forward	VC-MODE
149	*Turn Left*	Rotates Left	VC-MODE
150	*Turn Right*	Rotates Right	VC-MODE
151	*Back up*	Steps back	VC-MODE
152	*Action Start*	Special Performance	VC-MODE
153	*How are you?*	Appropriate Response	VC-MODE
154	*What's up?*	Appropriate Response	VC-MODE
155	*i-SOBOT*	Appropriate Response	VC-MODE
156	*Look out!*	Appropriate Response	VC-MODE
157	*Make me laugh*	Appropriate Response	VC-MODE

Appendix B iSobot Robot Features

Package Contents

- Pre-Assembled Robot
- LCD Remote Control Unit
- NiMH Battery Charger
- Hex Wrench
- AAA NiMH Rechargeable Batteries

Product Specifications

- i-SOBOTDimensions H: 6.5" x W: 3.8" x D: 2.6"
- Weight Approx. 12 oz (including batteries)
- Battery Requirements - Robot: 3 AAA NiMH batteries (included) - Remote Control: 4 AAA batteries (not included)
- Battery Life : The i-Sobot robot's batteries will run approximately 60 minutes under normal operation when fully charged. (Actual results will vary by type of use.)
- Transmission :Infared Transmission (2 bands)
- Main Components : Servo Unit 17 degrees of freedom, Built-in Gyro Sensor, Built-in Speaker Unit, Built-in Microphone along with a Voice Recognition Chip
- Servos Height: 3.83", Width: 4", Depth: 0.33", Maximum degree of movement: 220°, Metal gears, Built-in clutches

Operation Modes

- Remote Control Mode : Movements can be controlled directly using joy sticks and command buttons. He can also perform hundreds of amazing pre-programmed actions using short button codes.
- Programming Mode : Program a sequence of up to 80 actions, then easily play the sequence back with the push of a button.
- Special Action Mode : Command i-SOBOT to play air guitar, impersonate a movie character, or perform any of 18 pre-programmed special actions.
- Voice Command Mode :i-SOBOT recognizes 10 voice commands, reacting with dozens of appropriate but unpredictable actions.

Appendix C CHAMELEON CCU SPECIFICATIONS (Musical Light controller)

FREQUENCY RANGE	30 to 30,000Hz
INPUT IMPEDANCE (NOMINAL)	0K ohms
INPUT LEVEL	1 volt peak-to-peak, nominal (350mV rms), diode-limited
CHANNEL OPTIONS	Jumper selectable: stereo, left channel or right channel operation.
DLC	User adjustable, automatic Digital Level Compensation.
LEVEL	User adjustable master LEVEL control.
LEVEL INDICATORS	Green LED for proper audio in. Red LED for level too high.
THRESHOLD	User adjustable base voltage level setting to pre-warm filaments of incandescent or halogen bulbs, thus giving faster response time and longer bulb life. Also enables optimum use of LED and dimmable CFL fixtures.
SAMPLING	digitally-sampled sound with automatic peak detection. Frequency-weighted sampling.
RESPONSE TIME	channel-firing response time is 8.3 milliseconds.
TEST ROUTINE	Pressing TEST to check operation of CCU, PMU and connected

	lights.
AUDIO FILTERING	Four-pole active filters for each of the five channels. (Fliege topology). Adjacent channel rejection: High- and low-pass, 24dB/octave; low-mid and mid-high bandpass, 18dB/octave; midrange, 12dB/octave.
CHANNEL INDICATORS	Separate activity indicators for each channel show actual output to PMU.
INDIVIDUAL LEVEL CONTROLS	Individual level controls for each channel provide +/- 6 dB adjustment.
MICROPROCESSOR	2MHz (8MIPS)
OUTPUT TO POWER MANAGEMENT UNIT (PMU)	Digital (TTL level) pulse width modulation (PWM).
DC INPUT	9-24vdc, 100mA
TEST INDICATOR	Heartbeat (blinking light) indicates systems are operational.
MODE	Narrows channel bandwidths when depressed, increasing selectivity to create a sharper, livelier response.
AUDIO IN/OUT	Standard 1/8" (3.5mm) stereo jacks.
LINK LIGHT	Indicates that a PMU is connected.

Appendix D Text to Speech Synthesis Methods

http://msdn.microsoft.com/en-us/library/ms719576(v=VS.85).aspx

ISpVoice Methods	Description
ISpEventSource inherited methods	All methods of ISpEventSource are accessible from this interface
SetOutput	Sets the current output object. A value of NULL may be used to select the default audio device.
GetOutputStream	Retrieves a pointer to the current output stream.
Pause	Pauses the voice at the nearest alert boundary and closes the output device.
Resume	Sets the output device to the RUN state and resumes rendering.
SetVoice	Sets the identity of the voice used for text synthesis. By default, ISpVoice will use the voice information set in Speech properties in Control Panel.
GetVoice	Retrieves the object token that identifies the voice used in text synthesis.
Speak	Speaks the contents of a text string or file.
SpeakStream	Speaks the contents of a stream.
GetStatus	Retrieves the current rendering and event status associated with this *ISpVoice* instance.
Skip	Causes the voice to skip forward or backward the specified number of items within the text of the

	current speak call.
SetPriority	Sets the priority for the voice. Normal, Alert, Over.
GetPriority	Retrieves the current voice priority level.
SetAlertBoundary	Specifies which event should be used as the insertion point for alerts.
GetAlertBoundary	Retrieves the event that is currently being used as the insertion point for alerts.
SetRate	Sets the text rendering rate adjustment in real time.
GetRate	Retrieves the current text rendering rate adjustment.
SetVolume	Sets the synthesizer output volume level in real time.
GetVolume	Retrieves the current output volume level of the synthesizer.
WaitUntilDone	Blocks the caller until either the voice has completed speaking or the specified time interval has elapsed.
SetSyncSpeakTimeout	Sets the timeout interval in milliseconds after which, synchronous Speak and SpeakStream calls to this instance of the voice will timeout.
GetSyncSpeakTimeout	Retrieves the timeout interval for synchronous speech operations for this ISpVoice instance.
SpeakCompleteEvent	Returns an event handle that will be signaled when the voice has completed speaking all pending requests.
IsUISupported	Determines if the specified type of UI is supported.
DisplayUI	Displays the requested UI.

Appendix E Serial Port Setup

The PC-based remote is controlled by the PC via a virtual serial port. The communication settings for this port are mentioned above.

Serial Protocol

The PC communicates with the remote using a simple ASCII character protocol.

The iSOBOT accepts two kinds of commands. The majority of the commands are 22 bits in length. There are also a small number of commands that provide granular control over the iSOBOT's arm servos and are encoded using 30 bits.

The PC-based remote relays both kinds of command from the PC to the iSOBOT. It expects a string of hexadecimal character that encode the desired command bits. It determines the type of command to be sent based upon the number of characters received:

6 Characters:	22-bit command
8 Characters:	30-bit command

Character strings should be zero padded on the left in order to form a string of the appropriate length. For instance a value of 0x3456 should be sent as 003456 in order to send a 22-bit command and 00003456 in order to send a 30-bit command.

The PC-based controller expects a carriage return (ASCII code 13 – '\r') to terminate the string. Do not null terminate the string – the PC-based remote will not recognize this code.

The PC-based remote does echo the received characters back to the PC. This allows the remote to receive commands for even simple terminal emulators (e.g. TeraTerm or HyperTerm) and provides a simple method for error checking, if desired. (Note that the USB layer of the interface provides its own end-to-end error check so the virtual serial interface is deemed reliable enough not to require error checking.)

Timing

There are several timing related constraints which must be enforced by the user.

First, the PC must allow the PC-based remote to finish transmitting an IR command. This means waiting approximately (2.5ms + 1.5ms*NUMBER_OF_BITS) 35.5ms for 22-bit commands and 47.5ms for 30-bit commands.

Second, the iSOBOT hand held remote sends commands at a rate of only one every 200ms. We take this to imply that the iSOBOT cannot process commands at a much faster rate and recommend that the controlling PC software enforce a gap of 200ms between the start of successive messages to the same iSOBOT.

Note that the PC-based remote can send commands to both channels A and B and that the second, 200ms, constraint does not hold for successive commands to different robots.

System Requirements

The PC-based remote should work with any Windows (Windows 98 or higher), MAC or Linux machine with a powered USB port. A driver for the PC-based remote's on board FTDI USB-over-serial chip is required. Most operating systems should be able to automatically acquire the driver if it isn't already pre-installed, as in the case of Linux kernels 2.6.9 and higher. If the operating system does not automatically obtain the driver, it can be downloaded at: http://www.ftdichip.com/Drivers/VCP.htm

Appendix F Chameleon Quick Start User Guide

Connect your audio source to the Chameleon CCU's AUDIO IN jack via a standard *1/8"* stereo connector. Chameleon is designed for a standard line level input. Sources can be the line out from your stereo system or you can connect *CD/DVD* decks, laptop computers, MP3 devices or any typical earphone output connection.

Connect the CCU (Chameleon Control Unit) to the PMU (power management module) via a standard 9-wire serial cable (included). Serial cables up to 100' long may be used.

Now connect lights to your CCU. You may connect up to 2 amps per channel to a *CCU/10,* and up to 10 amps total for the system. Incandescent lamps often give the most pleasing results, but the system is also compatible with most dimmable LED lights as well. LEDs have less range and they are much quicker to respond, so the results, while pleasing, are quite different than with incandescent lights. Because different settings are generally required, it is advisable not to mix light types.

You may now connect the PMU to a typical household outlet and the CCU can be connected to power via the included power supply.

Turn on the CCU's power switch. The red HEARTBEATITEST light should slowly blink.

A green LINK light should also be glowing in the back, showing that the CCU and PMU

are communication. You should also see a green LINK light and red POWER indicator on the PMU.

You should now set the THRESHOLD control on the read of the CCU. Rotate it until your connected lights just begin to glow, then back it down just a hair. The lights should be just ready to glow, but not quite.

If you are using an earphone output from your audio source, make sure the volume is turned up significantly. Now rotate the LEVEL control on the CCU until the green LEVEL

OK light is on almost constantly and the red LEVEL HI indicator occasionally blinks.

You can adjust the DLC control to your liking. Remember, a little goes a long way. With this control too high, the lights will appear a little "mushy" in their response. You can also experiment with the MODE button on the CCU front panel. Generally the out position is best, but some music with a strong beat is represented better with the MODE button in the "in" position. This is a matter of personal taste.

Finally, you should look to see if anyone channel is too active or if it rarely flashes. Individual channel gain can be adjusted with the five level controls. Remember that you must set these for an average as the next song may have more or less of a particular frequency.

Remember, Chameleon accurately converts your sounds to light. If all frequencies are not present, not all lights may lights. This is natural.

Also keep in mind that you don't always hear what you think you hear. In other words, Chameleon will react to harmonics and overtones that you may not notice. This is why a drummer can kick a bass drum and light the HIGH channel on Chameleon. Some notes may light several channels, depending on the type of instrument.

Enjoy your Chameleon system ... the most sophisticated device to convert sound to music on the market!

Chameleon Operation

The following explains the basic controls, indicators and features of the Chameleon system.

Individual channel controls

The individual level controls adjust their respective band by *+1-* 6dB. The normal position

is top dead center. There is a tiny marking on the control, which points straight up (12 o'clock position). You can adjust at will, but if you're having trouble, always go back to the top dead center position.

Activity indicators

The blue LEOs above each control show the digital output of the Chameleon brain (CCU). If these are blinking with music playing and the level set properly, the CCU is working as designed.

Mode

The MODE button switches between considerable crossover between bands and no crossover. The out (default) position is generally most pleasing, but some types of music work better with no crossover and some people prefer the snappier reaction when there is no crossover. This setting is strictly a matter of personal preference.

Test button

The TEST button runs a routine that checks virtually everything in the CCU and PMU. One should see any connected lights come on in succession, then the entire group fades out. This lets us know each channel is firing properly and that all the dimming circuitry is working. Pressing this button is the first thing to do in the event that one is having a problem.

Audio level indicators

The level OKAY and HIGH LEOs form a basic level detector. Set the level control to where the green is on most of the time and the red occasionally flickers. You now have the appropriate audio level.

AUDIO IN and AUDIO OUT

These standard 3.5mm *(1/8")* jacks are tied together. Chameleon simply samples the sound that is presented at the IN jack. Chameleon is a high impedance device, so it is compatible with line-level outputs as well as headphone level devices, such as an iPod or laptop. You do NOT have to pass sound through Chameleon. The OUTPUT jack is just there as a convenience.

Threshold

The Threshold control sets a minimum level of voltage for the connected bulbs. This keeps the filaments warm for quick reaction and it also effectively increases the sensitivity of the system. It is normally set to where the filaments of the connected lights just begin to glow then backed down just a hair. Some folks prefer to have a minimal amount of light on all the time, so you can increase the threshold level if you prefer.

DLC

The Digital Level Control is a digital method of automatic level control for each channel (individually). Chameleon samples the signal level several times per second and adjusts levels.

This is basically done with digital audio compression. Generally it is recommend a small amount of DLC. If turned off (fully counter clockwise) one loses the automation. If it is turned too high, the response becomes a bit mushy. About a quarter turn of this knob generally works best... Different users might have different opinion about it.

Link light

The link light tells you that there is a connection between the CCU (brain) and PMU. If the link light is out, the system will not function.

DB9 connectors

Left DB9 connector is data in from the CCU. Again, a link light tells you that you have a good connection. Power indicates that you have AC power to the PMU. The five channel indicators actually sample the AC outlets. If they're working, the entire system

is working. The next link light shows a successful connection between this PMU and another daisy-chained PMU. The DB9 on the right side is the digital signal out to the next PMU.

FUSE

The fuse in back is a 12 amp ceramic fast blow. Never use slow blow or larger size fuses.