chapter{Introduction}

\section{Motivation}

From the mechanical music performing automata from middle ages, to the latest Japanese virtualsinger Hatune Miku, there had been many attempts to create automated systems that perform music. However, many of these systems can only generate predefined expressions. State-of-the-art text-to-speech system can already generate fluid and natural speech, but a computer performance system still can't perform very expressively. Therefore, many researchers have devoted their effort to develop systems that can automatically or semi-automatically perform music expressively. There is even a biannual contest for such systems called Music Performance Rendering Contest (RenCon)\cite{RenCon}. The RenCon sets a goal that by 2050, a computer performer can win the Chopin International Piano Contest.

There are many potential applications for a computer expressive performance system, many commercial music typesetting software like Finale\cite{finale} and Sibelius\cite{sibelius} already have expressive playback features built-in. For entertainment industry, such systems can provide personalized music listening experience. For music production industry, this technology can save a lot of cost on hiring musicians and license fees. Such systems also open up a new opportunity in art, such as human-machine co-performance or interactive multimedia installation. In academia, researchers can use this technology to study the performance style of musicians, or restore historical recording archive.

\section{Goal and Contribution}

The ultimate goal of this paper is to be able to play any music in any expressive style specified. But due to technical and time constrains, we narrow down our goal to building a computer expressive performance system that performs monophonic music phrases by off-line supervised learning. The phrasing is left to human, so the system built in this thesis is a semi-automatic one.

The major contribution of this paper is that we apply a structural support vector machine to construct an expressive performance system. No previous systems used the discriminative learning power of the structural support vector machine with hidden Markov model output (SVM-HMM) to improve the computer’s capability for performing an expressive music performance. To build our system, wedeveloped methods and tools to prepare an expressive performance corpus for training and determining necessary parametric values of the SVM-HMM machine. Since there is no unified ways to evaluate the expressive performance, we arranged subjective tests and find that our system performance is not recognized as better than the human performance, however, their difference is without statistical significance for subjects having no music backgrounds.

.

\section{Chapter Organization}

In Chapter \ref{chap:prev}, we give an overview of previous works with varyious goals. These works are grouped by the way they learn performance knowledge, and we will discuss some additional specialities such as special instrument model or special user interaction pattern. In Chapter \ref{chap:proposed}, we first give a brief introduction to the mathematical background of SVM-HMM, and then introduce a top-down explanation to the proposed method. In Chapter \ref{chap:corpus}, we explain how the corpus for training is designed and implemented. In Chapter \ref{chap:exp}, we examine several experiments that demostrate design trade-offs in realizing our system and the subjective test results. Finally, we summarize our work and point out possible improvement in Conclusions and present some software tools used in this research, which may be helpful for other researchers in the computer music field.

\chapter{Previous Works}

\label{chap:prev}

\section{Various Goals and Evaluation}

The general goal of a computer expressive performance system is to generate expressive music, as opposed to the robotic and dull expression of rendered MIDI. But the definition of "expressive" is very vague and ambiguous, so each research will need to define a more precise and measurable goal. The following are the most popular goals a computer expressive performance system wants to achieve:

\begin{enumerate}

\item Perform music notations in a non-robotic way (no specific style).

\item Reproduce a human performance or a certain musician's style.

\item Accompany a human performance.

\item Validate a musicological theory of expressive performance.

\item Directly render computer-composed music works.

\end{enumerate}

Some systems try to perform music notations in a humanic way in a general sense, without a certain style in mind. These systems have been employed in music typesetting softwares, like Finale \cite{finale} and Sibelius \cite{sibelius}, to play the staff expressively. Most systems also implicitly include this capability.

Systems that are designed to reproduce certain humanic performance or style are usually designed and trained using a particular performer's recordings. One commercial example is the Zenph re-performance CD \cite{zenph}. This CD contains music performed by an expressive performance model of Rachimaninov's style, while Rachimaninov had never recorded these pieces in his lifetime.

Accompaniment systems try to render expressive music that act as an accompaniment for a human performance. The challenge is that the system must be able to track the progress of the human performance and adaptively render the accompaniment in real-time. One commercial example is Cadenza \cite{cadenza}, using the technology created by Christopher Raphel. It can track the soloist's performance and play the accompaniment orchestral part accordingly.

Another goal is to validate musicological theories. Musicologist may propose theories on expressive music performance, by building a generative model, they can validate their theories. These systems may focus more on the specific sound effect that the theory tries to explain instead of generating music that is pleasant to human.

Finally, some systems combine computer composition with expressive performance. These systems have a big advantage because the intention of the composer can be shared with the performer. Other systems that perform past compositions can only guess the composer's intention by analyzing the score notation. These systems usually have their own data structure to represent music, which can contain more information than traditional music notation, but the performance system is not backward compatible with past compositions.

Because of the high diversity in the goals they want to achieve, it is very hard to make fair comparison between systems. But we can still evaluate the capability of these systems by the following three key indicators proposed by \cite{THEBOOK}:

\begin{enumerate}

\item Expressive expression capability

\item Polyphonic capability

\item Performance creativity

\end{enumerate}

Expressive expression capability ranges from a very high-level structural expression (e.g., tempo contrast between sections) to note level expression (e.g. onset, loudness, duration) or even sub-note expression (e.g. loudness envelop, timbre). Most systems can generate note-level expression, but higher or lower level expressions are much rare.

Polyphonic capability indicates if the system can perform apolyphonic staff. Polyphonic systems are more challenging than monophonic ones because they require synchronization between voices.

Performance creativity measures the ability of the system to create novel expression. The desired level of creativity varies from goal to goal. A system aiming to recreate human performance may want to produce deterministic expressions based on the learned knowledge , while a system combined with a composition system may want to create highly novel performance.

Each system is along with some different experiments and metrics to verify its goal. Thus, the self-reported results are hardly compared. The only public contest that evaluates expressive performance systems is RenCon (Performance Rendering Contest)\cite{RenCon}. Scores (MIDI) will be given to participants one hour before the competition starts. The participants must generate the expressive version of the MIDIs in the given time; the MIDIs will be played live on a Yamaha Disklavier piano. The audience and a jury cosists of professional musicians will give ratings for each performance. The performances are arranged in random, so the audience and jury won't know which participant is behind each performance.

The RenCon is divided into fully automatic and semi-automatic categories. But the degree of human intervention in the semi-automatic category varies widely between systems. So it's not very fair to compare them.

\section{Researches Classified by Methods Used}

Despite the difference between goals of different expressive performance systems, all expressive performance systems must have some strategy to learn and apply performance knowledge. There are generally two approaches: rule-based and machine-learning-based.

Using rules to generate expressive music is probably the earliest approach. Director Musices \cite{17} is one of the early examples. Pop-E \cite{28} is also a rule-based system which can generate polyphonic music, using its voice synchronization algorithm. Computational Music Emotion Rule System \cite{31} tried to develop rules that express human emotions. Other systems like Hierarchical Parabola System \cite{17,18,19,20}, Composer Pulse System \cite{21,22}, Bach Fugue System \cite{23}, Trumpet Synthesis System \cite{24, 25} and Rubato \cite{26, 27} are also some examples. Most of the rule-based systems focus on expressive attributes like note onset, note duration and loudness, but Hermode Tuning System \cite{29} put special emphasis on intonation. Rule-based systems are generally more computationally efficient because the mathematical model is much simple than those learned by machine learning algorithms. And rules are generally more understandable to human than complex model parameters. But some of the nuance, such as some subconscious deviation, may be hard to describe by rules, so there is an empirical limit on how complex the rule-based system can be. Lack of creativity is also a problem for rule-based approach.

Another approach is to acquire performance knowledge by machine learning. Many machine learning methods have already been applied to this problem. For example, Music Interpretation System \cite{32,33,34} and CaRo \cite{35,36,37} both use linear regression to learn performance knowledge. But it is very unlikely that the expressive performance can be generated fromn a linear system, so Music Interpretation System tries to introduce non-linearity by using logic AND operations on linear regression results. But generally speaking, linear regression is too simple to capture the core of expressive performance.

More complicated machine-learning algorithms have also been applied: ANN Piano \cite{38} and Emotional flute \cite{39} uses artificial neural network. ESP Piano \cite{55} and Music Plus One \cite{52,53,54} uses statistical graphical models such as hidden Markov model (HMM) and Bayesian belief network, but they did no use structural support vector machine to train the HMM. KCCA Piano System \cite{57} uses kernel regression. Drumming System \cite{82} tried different mapping models that generate drum patterns.

Evolutionary computation such as genetic programming is used in Genetic Programming Jazz Sax \cite{88}, Sequential Covering Algorithm Genetic Algorithm \cite{59}, Generative Performance Genetic Algorithm \cite{89} and Multi-Agent System with Imitation \cite{60, 93}. Evolutionary computation takes long training time, and the results are less predictable. But being unpredictable also means that these systems can create interesting performances in an unconventional way.

Another possible approach is to use case-based reasoning. SaxE \cite{40,41,42} uses fuzzy rules based on emotions to generate Jazz saxophone performance. Kagurame \cite{43,44} focuses on style (Baroque, Romantic, Classic etc.) instead of emotion. Ha-Hi-Hun \cite{45} has a more ambitious goal in mind: to accept natural language instructions like \enquote{Perform piece X in the style of Y.} Another series of researches done by Widmer at el., the PLCG \cite{46, 47, 48}, uses data mining technique to find rules for expressive performance. Its successor -- Phrase-decomposition/PLCG \cite{49} added hierarchical phrase structures support to the original PLCG system. And the latest research in the series of DISTALL \cite{50, 51} added hierarchical rules to the original one.

Most of the performance systems discussed above takes musical notation (MusicXML, MIDI, etc.) or inexpressive audio as input. They have to figures out the expressive intention of the composer by analyzing the score. But another type of computer expressive performance has a big advantage over the previous ones, by combining computer composition and expressive performance, the performance module can receive the composition intention directly from the composition module. Ossia \cite{61} and pMIMACS \cite{pmimacs} are two examples of this category. This approach provides great possibility for creativity, but can only play their own composition, which limits its range of application.

\section{Additional Specialties}

Most expressive performance systems implicitly or explicitly generate piano performance, because it's relatively easy to collect training samples for piano, and a piano sound is relatively easy to synthesize. Yet, some systems generate music in other instruments, such as saxophone \cite{40, 41, 42}, trumpet \cite{24, 25}, flute \cite{39} and drums \cite{56}. These systems require extra effort in creating instrument models in training from, generation and synthesis of expressive music. Y.-H Kuo et al. \cite{profsu} also proposed a way to re-synthsize individual notes into a performance with smooth timbre variation, but they focused more on sub-note level timbre synthesis.

If not specified, most systems handle traditional western tonal music. However, most saxophone-based work \cite{40, 41, 42} generates Jazz music, because saxophone is an iconic instrument in Jazz performance. And the Drumming System \cite{56} generates Brazilian drumming music.%The Bach Fugue System \cite{23}, literally, focuses on fugue works composed by bach.

Performing polyphonic music is much more challenging than monophonic music, because it requires synchronization between voices. Pop-E \cite{28} uses a synchronization mechanism to achieve polyphonic performance. Bach Fugue System \cite{23} is created using the polyphonic rules in music theory about fugue, so it's inherently able to play polyphonic fugue. KCCA Piano System \cite{57} can generate homophonic music -- an upper melody with an accompaniment -- which is common in piano music. Music Plus One \cite{52,53,54} is a little bit different because it's an accompaniment system, and it adapts a non-expressive orchestral accompaniment track to the user's performance. %Other systems usually generates monophonic tracks only.

\section{A Brief Introduction to SVM-HMM}

\label{sec:svm-hmm}

In this thesis, we use structural support vector machine to learn performance knowledge from expressive performance samples. Unlike traditional SVM algorithm, which can only produce univariate prediction, structural SVM can produce structural predictions like trees, sequences and hidden Markov models. Structural SVM with hidden Markov model output (SVM-HMM) has been successfully applied to the part-of-speech tagging problem\cite{svm2009}. The part-of-speech tagging problem has some similarity with expressive performance problem. In part-of-speech tagging, one tries to identify the role a word plays in the sentence, while in expressive performance, one tries to determine how a note should be played, usually based on its role in the musical phrase. For example, an authentic cadence at the end of a phrase is usually played louder and stronger than an embellishment note in the middle of a phrase. Thus, we believe that SVM-HMM is also a good candidate for expressive performance. The following introduction and formulas are summaries of \cite{svm2009, svm2005, svm2003}.

Traditional SVM prediction problem can be described as finding a function

$$h: \mathcal{X \rightarrow Y}$$ with the lowest prediction error. $\mathcal{X}$ is the input features space, and $\mathcal{Y}$ is the prediction space. In traditional SVM, elements in $\mathcal{Y}$ are labels (by classification) or real values (regression). But structural SVM extends the framework to generate structural output, such as a tree, a sequence, or a hidden Markov model.

To extend SVM to support structured output, the problem is modified as finding a discriminant function $f: \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{R}$, in which the input/output pairs are mapped to a real number score. To predict an output $y$ for an input $x$, one tries to maximize $f$ over-all $y \in \mathcal{Y}$.

$$h\_{\mathbf{w}}(x) = \argmax\_{y\in\mathcal{Y}} f\_{\mathbf{w}}(x,y)$$

Let $f\_{\mathbf{w}}$ be a linear function of the form:

$$ f\_{\mathbf{w}} = \mathbf{w}^{T}\Psi(x,y)$$,

where $\mathbf{w}$ is the parameter vector, and $\Psi(x,y)$ is the kernel function relating input $x$ to output $y$. $\Psi$ can be defined to accommodate various kind of structures.

For each structure we predict a loss function that measures the accuracy of a prediction. A loss function $\Delta:\mathcal{Y}\times\mathcal{Y}\rightarrow R$ need to satisfy the following properties:

$$\Delta(y, y') \geq for\ y \neq y'$$

$$\Delta(y, y) = 0 $$

The loss function is assumed to be bounded. Let's assume that the input-output pair $(x,y)$ is drawn from a join distribution P(x,y), the prediction problem is to minimize the total loss:

$$R\_p^\Delta = \int\_{\mathcal{X} \times \mathcal{Y}} \Delta (y, f(x))dP(x,y)$$

Since we can't directly find the distribution $P$, we need to replace this total loss with an empirical loss, which can be calculated from the observed training set of $(x\_i, y\_i)$ pairs.

$$R\_s^\Delta(f) = \frac{1}{n}\sum^n\_{i=1}\Delta(y\_i, f(x\_i))$$

Now we are ready to extend SVM to structural output, starting with a linear separable case, and we will then extend it to a soft-margin formulation.

A linear separable case can be expressed by a set of linear constraints

$$\forall i \in \{1,\cdots,n\}, \forall \hat{y\_i}\in\mathcal{Y}: \mathbf{w}^T [\Psi(x\_i, y\_i) - \Psi(x\_i, \hat{y\_i})]\leq 0$$

However, in the SVM context, we want the solution to have the largest margin possible. So the above linear constraints become the optimization problem:

$$

\begin{aligned}

& \max\_{\gamma, \mathbf{w}:\|\mathbf{w}\| = 1} \gamma \\

& s.t \; \forall i \in \{1,\cdots,n\}, \forall \hat{y\_i} \in\mathcal{Y}: \mathbf{w}^T [\Psi(x\_i, y\_i) - \Psi(x\_i, \hat{y\_i})] \leq \gamma\\

\end{aligned}

$$

, which is equivalent to the convex quadratic programming problem:

$$

\begin{aligned}

& \min\_{\mathbf{w}, \xi\_i \geq 0} \frac{1}{2}\mathbf{w}^T\mathbf{w} \\\

&s.t.\; \forall i \in \{1,\cdots,n\},\hat{y\_i} \in \mathcal{Y}: \mathbf{w}^T[\Psi(x\_i,y\_i) - \Psi(x\_i,\hat{y\_i})] \geq 1\\

\end{aligned}

$$

To extend the linear-separable case to a non-separable case, slack variables $\varepsilon\_i$ are introduced to penalize prediction errors, which results in a soft-margin formalization:

$$

\begin{aligned}

& \min\_{\mathbf{w}, \xi\_i \geq 0} \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{C}{n}\sum^n\_{i=1}\xi\_i\\

&s.t.\; \forall i \in \{1,\cdots,n\},\hat{y\_i} \in \mathcal{Y}: \mathbf{w}^T[\Psi(x\_i,y\_i) - \Psi(x\_i,\hat{y\_i})] \geq 1 - \xi\_i \\

\end{aligned}

$$

$C$ is the weighting parameter controlling the trade-off between low training error and large margin. The optimal $C$ varies between different problems, so experiments should be conducted to find the optimal $C$ for our problem.

Intuitively, a constraint violation with larger loss should be penalize more than the one with smaller loss. So I. Tsochantaridis et al. \cite{svm2005} proposed two possible ways to take the loss function into account. The first way is to re-scale the slack variable by the inverse of the loss, so a high loss leads to smaller re-scaled slack variables:

$$

\begin{aligned}

& \min\_{\mathbf{w}, \xi\_i \geq 0} \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{C}{n} \sum^n\_{i=1}\xi\_i\\

&s.t.\; \forall i \in \{1,\cdots,n\},\hat{y\_i} \in \mathcal{Y}: \mathbf{w}^T[\Psi(x\_i,y\_i) - \Psi(x\_i,\hat{y\_i})] \geq 1 - \frac{\xi\_i}{\Delta(y\_i, \hat{y\_i})} \\

\end{aligned}

$$

The second way is to re-scale the margin, which yields

$$

\begin{aligned}

& \min\_{\mathbf{w}, \xi\_i \geq 0} \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{C}{n} \sum^n\_{i=1}\xi\_i\\

&s.t.\; \forall i \in \{1,\cdots,n\},\hat{y\_i} \in \mathcal{Y}: \mathbf{w}^T[\Psi(x\_i,y\_i) - \Psi(x\_i,\hat{y\_i})] \geq \Delta(y\_i, \hat{y\_i}) - \xi\_i\\

\end{aligned}

$$

But the above quadratic programming problem has a very large number ($O(n|\mathcal{Y}|)$) of constraints , which will take considerable time to solve. I. Tsochantaridis et al. \cite{svm2005} proposed a greedy algorithm to speed up the process by selecting only part of the constraints that contribute the most to finding the solution. Initially, the solver starts with an empty working set containing no constraints. Then the solver iteratively scans the training set to find the most violated constraints under the current solution. If a constraint is violated more times than a desired threshold, the constraint is added to the working set of constraints. Then the solver re-calculates the solution under the new working set. The algorithm will terminate once no more constraint can be added under the desired precision.

In a later work by Joachims et al.\cite{svm2009}, they created a new formulation and an algorithm to further speed up the algorithm. Instead of using one slack variable for each training sample, which results in a total of $n$ slack variables, they use a single slack variable for all $n$ training samples. The following formula is the 1-slack version of slack-rescaling structural SVM:

$$

\begin{aligned}

& \min\_{\mathbf{w}, \xi\_i \geq 0} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C \xi\\

&s.t.\; \forall i \in \{1,\cdots,n\},\hat{y\_i} \in \mathcal{Y}: \mathbf{w}^T[\Psi(x\_i,y\_i) - \Psi(x\_i,\hat{y\_i})] \geq \frac{1}{n}\sum^n\_{i=1}1 - \frac{\xi}{\Delta(y\_i, \hat{y\_i})} \\

\end{aligned}

$$

And margin-rescaling structural SVM:

$$

\begin{aligned}

& \min\_{\mathbf{w}, \xi\_i \geq 0} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C \xi\\

& s.t.\; \forall i \in \{1,\cdots,n\},\hat{y\_i} \in \mathcal{Y}: \mathbf{w}^T[\Psi(x\_i,y\_i) - \Psi(x\_i,\hat{y\_i})] \geq \frac{1}{n}\sum^n\_{i=1}\Delta(y\_i, \hat{y\_i}) - \xi \\

\end{aligned}

$$

Detailed proof on how the new formulation is equally general as the old one is given in the paper \cite{svm2009}.

With the framework described above, the only problem left is how to define the general loss function for hidden Markov model (HMM)? In \cite{svm2003}, Y. Altun et al. proposed two types of features for an equal-length observation/label sequence pair $(x,y)$. The first is the interaction of an observation with a label, the other is the interaction between neighboring labels.

To illustrate the method, we use an example from music: for some observed features $\Phi\_r(x^s)$ of a note $x$ located in $s$th position of the phrase, and assume that $\left[ \left[ y^t = \tau \right] \right]$ denotes the $t$th note is played at a velocity of $\tau$, the interaction of the two predicate can be written as

$$\phi^{st}\_{r\sigma}(\mathbf{x}, \mathbf{y}) = \left[\left[y^t = \tau \right] \right]\Psi\_r(x^s),\; 1\leq\gamma\leq d,\; \tau \in \Sigma $$

And for interaction between labels, the feature can be written as

$$\hat{\phi}^{st}\_{r\sigma}(\mathbf{x}, \mathbf{y}) = \left[\left[y^s = \sigma \wedge y^t = \tau \right] \right],\; \sigma, \tau \in \Sigma $$

By selecting an order of dependence for the HMM model, we can further restrict $s$'s and $t$'s. For example, for a first-order HMM, $s = t$ for the first feature, and $s = t-1$ for the second feature. The two features on the same time $t$ is then stacked into a vector $\Psi(x,y;t)$. The feature map for the whole sequence is simply the sum of all the feature vectors

$$\Phi(\mathbf{x}, \mathbf{y}) = \sum^T\_{t=1}\Phi(\mathbf{x}, \mathbf{y};t)$$

Finally, the distance between two feature maps depends on the number of common label segments and the inner product between the input features sequence with common labels.

$$\langle\Phi(\mathbf{x}, \mathbf{y}), \Phi(\mathbf{\hat{x}}, \mathbf{\hat{y}})\rangle = \sum\_{s,t}\left[\left[y^{s-1} = \hat{y}^{t-1}\wedge y^s = \hat{y}^t\right] \right] + \sum\_{s,t}\left[\left[y^{s} = \hat{y}^{t}\right] \right]k(x^s, \hat{x}^t)$$

Finally a Viterbi-like decoding algorithm is used to speed up the computation of $F$ for HMM..

\chapter{Proposed Method}

\label{chap:proposed}

\section{Overview}

%input output only melodic constrains

%flow chart

\begin{figure\*}[tp]

\begin{center}

\includegraphics[width=\textwidth]{fig/high\_lev\_arch}

\end{center}

\caption{High-level system architecture}

\label{fig:flow}

\end{figure\*}

The high-level architecture of the proposed system is shown in Fig. \ref{fig:flow}. The system has two phases, the upper half of the figure is the learning phase, and the lower half is the performing phase. In the learning phase, score and expressive human recording pairs, split into phrases by human, are used as training examples for structural support vector machine with hidden Markov model output (SVM-HMM) algorithm to learn a performance model with knowledge. In the performing phase, a score will be given to the system for expressive performance. The SVM-HMM generation module will use the performance knowledge learned in the previous phase to produce expressive performance. The SVM-HMM output then go through a MIDI generator and MIDI synthesizer to produce audible performance.

All the scores and recordings are monophonic and contain only one musical phrase. The phrasing is done by the author, thus the system is semi-automatic. The learning algorithm applied in the learning phase, namely SVM-HMM, can only be executed off-line, while the performing phase is much faster; expressive music can be generated almost instantaneously.

There are many ways the user can control the performance style of the final output: First, the user can choose the training corpus. Theoratically, a model of a particular style can be learned from a set of samples with that particular style. Second, the user can control the structural expression in phrasing the score..

In the following sections, we give an overview of the theoratical background behind SVM-HMM, then walk through the detail steps in the learning and the performing phases, Some implementation details are also described. The features used will be presented in the end of this chapter.

\input{theoraticalbg}

\section{Learning Performance Knowledge}

\label{sec:learn}

\begin{figure\*}[tp]

\begin{center}

\includegraphics[width=\textwidth]{fig/learn\_arch}

\end{center}

\caption{Learning phase flow chart}

\label{fig:learnflow}

\end{figure\*}

In this section, we will introduce the components used in the learning phase.

The main goal in the learning phase is to extract performance knowledge from training samples. Fig. \ref{fig:learnflow} shows the internal structure of the learning phase.

Training samples are pairs of matched score and recorded expressive performance (their format and preparation process is discussed in Chapter \ref{chap:corpus}). The raw data from the samples are too complex to process, so we need to extract important features from them. Two types of features will be extracted from the samples: the musicological cues from the scores (score features) and the measurable expression from the expressive performances (performance features). We want the system to learn how score features are \enquote{translated} into performance features. This process can be analogize to a human performer reading the explicit and implicit cues from the score, and perform the music with certain expressive expression. The definition of the features used will be presented in Section \ref{sec:features}.

\subsection{Training Sample Loader}

The training samples are loaded by the sample loader. Since a training sample consists of a score (musicXML format) and an expressive recording (MIDI format), the sample loader finds the two files, and load them into an intermediate representation (\texttt{music21.Stream} object provided by the \texttt{music21} library \cite{music21} from MIT). The music21 library will convert the musicXML and MIDI format into a Python Object hierarchy that is easy to access and manipulate by Python code.

One caveat here is that the music21 library will quantize the time in MIDI, which will destroy the subtle onset and duration expressions. And the music21 library don't handle the \enquote{ticks per quarter note} information in the MIDI header \cite{midispec}, which is essential for the MIDI parser to interpret the correct time scale. So we must explicitly disable quantization and specify the \enquote{ticks per quarter note} value during MIDI loading.

\subsection{Features Extraction}

In order to keep the system architecture simple, feature extractors are designed to be independent of other feature extractors, so features can be included or removed without affecting the rest of the system. Furthermore, this enables parallel feature extraction. But sometimes a feature inevitably depends on other features, for example, the \enquote{relative duration with the previous note} is calculated based on the \enquote{duration} feature. Since we want to avoid the complex dependence management, the \enquote{relative duration with the previous note} feature extractor has to invoke the \enquote{duration} extractor, instead of waiting for the \enquote{duration} extractor to finish first. Therefore, the \enquote{duration} feature extracted will be computed twice. To avoid redundant computation of the feature extractors, we implemented a caching mechanism. Once the \enquote{duration} feature had been computed, no matter it is calculated during \enquote{duration} extraction or during the \enquote{relative duraion with the previous note} extraction process, its value will be cached during this execution session. So no matter how many feature extractors uses the \enquote{duration} feature, they can get the value directly from the cache. This can speed up the execution without needing to handling dependences.

The extracted features are aggregated and stored into a JavaScript Object Notation (JSON) file for the SVM-HMM module to load. By saving the features in a human-readable intermediate file, we can debug potential problems easily.%the learning module can then load this file as the input. The learning algorithm can then do any pre-processing on the features, such as aggregation or quantization. The output of this module is the algorithm=specific model description. For example, a linear regression algorithm will output the regression parameters. The algorithm is required to produce a model file containing the model description, but the system doesn't care about the internal format of the model description file, it will simply feed this model file to the generation module in the generation stage. So the developer of the learning module has to implement methods to write and read the model file themselves.

%In the early stage of this research, linear regression is used. The results are shown in \cite{Lyu2012}. In this thesis, Structural Support Vector Machine \cite{svm2009} is used instead. The detail of Structural SVM will be in the next Chapter.

\subsection{SVM-HMM Learning}

After all features are extracted, the next step is to learn performance knowledge from the features. In the early stage of this research, we have successfully applied linear regression \cite{Lyu2012}. However, assuming this problem to be linear is clearly an oversimplification, so we switch to structural support vector machine with hidden Markov model output (SVM-HMM)\cite{svm2009, svm2005, svm2003} as our supervised learning algorithm.

The SVM-HMM learning module loads the feature file from the previous stage, and aggregate the features to fit the required input format of the SVM-HMM learner program. However, most features from the previous stage are real values, but SVM-HMM only takes discrete performance features\footnote{SVM-HMM is initially designed for tasks like part-of-speech tagging, in which real value or binary features are used to predict discrete part-of-speech tags.}, so quantization is required. There are many possible way to quantize the features; each will result in different outputs. Here we introduce a quantizer design as an example: For each performance feature, the mean and standard deviation from all training samples are calculated first. The range between mean minus or plus four standard deviations is divided into 128 uniform intervals. Values greater than the mean value plus four standard deviations are quantized into the 128th bin, and values smaller than the mean value minus four standard deviations are quantized into the 1st bin. The number of intervals decides how fine-grain the quantization is, if the number is too small, subtle expressions will be lost due to high quantization error. However, if the number is too large, there will be too few samples for each interval, which is bad from a statistical learning perspective. Also the training process will take a lot of CPU and memory resources without significant gain in prediction accuracy. The range of four standard deviation is chosen by trial and error, a narrower range will make most of the extreme values be quantized into the largest of smallest bin, so the performance will have a lot of saturated values. But a very large range will make the interval between each quantization bin too large, rising the quantization error. %Feature values smaller than the mean value minus three standard deviations are quantized to the lowest bin, while values greater than the mean value plus three standard deviations are quantized to the highest bin. The range of three standard deviation and 1024 intervals are chosen by experience, which can be modified to fit different corpus. The information of mean, standard deviation and number of intervals are stored in a file for the performing phase to dequantize the output.

The theoretical background of SVM-HMM has already been mentioned in Section \ref{sec:svm-hmm}. We leverage Thorsten Joachims's implementation called $SVM^{hmm}$ \cite{Joachims2008}. $SVM^{hmm}$ is an implementation of structural SVMs for sequence tagging \cite{svm2003} using the training algorithm described in \cite{svm2005} and \cite{svm2009}. The $SVM^{hmm}$ package contains a SVM-HMM training program called \texttt{svm\\_hmm\\_learn} and a prediction program called \texttt{svm\\_hmm\\_classify}. For architectural simplicity, we train one model for each performance feature; each model uses all the score features to predict a single performance feature. The \texttt{svm\\_hmm\\_learn} reads the features from a file in the following format:

Each line represents features for a note in time order, and the format is as

\begin{lstlisting}[style=nonumbers]

PERF qid:EXNUM FEAT1:FEAT1\_VAL FEAT2:FEAT2\_VAL ... #comment

\end{lstlisting}

\texttt{PERF} is a quantized performance feature. The \texttt{EXNUM} after \texttt{qid:} identifies the phrases, all notes in a phrase will have the same \texttt{qid:EXNUM} identifier. Following the identifier are quantized score features, denoted as \texttt{feature name : feature value}, and separated by spaces. The text following a \texttt{\#} symbol is treated as a comment. %An example of the training file is shown in Fig. \ref{fig:expinput}

There are some key parameters need to be adjusted for the training program. The first is the $C$ parameter in SVM, which controls the trade-off between lowering training error and maximizing margin. Larger C result in lower training error, but the margin may be smaller. The second is the $\varepsilon$ parameter that controls the required precision for termination. The smaller the $\varepsilon$, the higher precision, but it may require more time and computing resources. Finally, for the HMM part of the model, the order of dependences of transition states and emission states needs to be specified. In our case, both are set to defaults: The transition dependence is set to one, which stands for first-order Markov property, and the emission dependence is set to zero. Since we train one model for each performance feature, each model can have their own set of parameters. The parameter selection experiments will be presented in Chapter \ref{chap:exp}.

Finally, the training program will output three model files (because we use three performance features) which contains SVM-HMM model parameters, such as the support vectors and other metadata. Since it takes considerable time (roughly a dozen minutes to a few hours) to train a model, depending on the amount of training samples and the power of the computer, the system can only support off-line learning. But the learning process only needs to be run once. The performance knowledge model can be reused over and over again in the performing phase.

\section{Performing Expressively}

\begin{figure\*}[tp]

\begin{center}

\includegraphics[width=\textwidth]{fig/perf\_arch}

\end{center}

\caption{Performing phase flow chart}

\label{fig:perfflow}

\end{figure\*}

The performing phase uses the performance knowledge model learned in the previous phase to generate expressive performances. The input is a score file to be performed, which should not be used as training sample to prevent overfitting. Score features will be extracted from it using the same routine as in the learning phase. The SVM-HMM generation module will use the learned model and the score features to predict the performance features. These features will then be de-quantized back to real values using the method described previously. A MIDI generation module will apply those performance features onto the score to produce a expressive MIDI file. The MIDI file itself is already an expressive performance, and to listen the sound, a software synthesizer can be used to render the MIDI file into WAV or MP3 format.

\subsection{SVM-HMM Generation}

The feature extraction and aggregation process in the performing phase is similar to the learning phase, but the \texttt{PERF} fields in the SVM-HMM input file are left blank for the algorithm to predict. The \texttt{svm\\_hmm\\_classify} program will take these inputs with the learned model file and predict the quantized labels of the performance features. These performance features are de-quantized back to the middle point of each bin.

\subsection{MIDI Generation and Synthesis}

%Since the output of the SVM-HMM learner is the quantized label for the performance feature, dequantization is required to turn those values back to real-valued performance features. The dequantizer will load the quantization parameters from the learning stage to understand the range and intervals used in the quantizer. Each quantization label is dequantized into the mean value of the interval it belongs.

The predicted performance features are then applied onto the input score, i.e. the onset timings will be shifted, the duration extended or shortened, and the loudness shifted according to the predicted performance features. The resulting expressive performance will be transfomed into MIDI files using \texttt{music21} library \cite{music21}.%For example, if a performance feature represents that the note's duration should last for 1.2 times of its nominal value, the duration in the score is multiplied by 1.2. After all the performance features are applied, the expressive version of the score is stored in MIDI format using the \texttt{music21} library.

In order to actually hear the expressive performance, the MIDI file can be rendered by a software MIDI synthesizer. %Since the output is a standard MIDI file, the user can choose any compatible software or hardware synthesizer.

For example, \texttt{timidity++} software synthesizer for Linux can render the MIDI into a WAV (Waveform Audio Format) file, which can be compressed into MP3 (MPEG-2 Audio Layer III) by \texttt{lame} audio encoder. Alternatively, one can use hardware synthesizers, for example, RenCon \cite{RenCon} contest uses Yamaha Disklavier digital piano to render contestants' submissions.

Because sub-note level expression is not the primary goal of this research, we choose standard MIDI grand piano sound to render the music. The system can be extended to use more advanced physical model or instrument-specific audio synthesizer. Some sub-note level features, such as special techniques for playing violins, can be added to the list of features and be learned by the SVM-HMM model.

\section{Features}\label{sec:features}

%The system is trying to mimic the process of human performance: the musican reads the explicit and implicit cues from the score and transforms them into musical expressions. So the features can be categorized into two categories: score features and performance features. Score features are information contained in the score. Performance features corresponds to the musical expression. The basic time unit for both features is a note.

Now we will present the features used in the system, and discuss the difficulties encountered.

\subsection{Score Features}

Score features are musicological cues presented in the score. The purpose of score features is to simulate the high level information a performer may perceive when he/she reads the score. Each note will have all features presented below.

Score features includes:

\begin{description}

\item [Relative position in the phrase:]

The relative position of a note in the phrase, its value ranges from 0\% to 100\%. %Since there are often salient musical expressions during the opening or closing of a phrase, this feature is used to capture the start or end of the phrase.

This feature is intended to capture the special expression in the start or the end of a phrase, or time-variant expression like arch-type loudness variation.

\item [Pitch:]

The pitch of a note denoted by MIDI pitch number (resolution is down to semitone). %For a phrase of $n$ notes with pitch $P\_1, P\_2, \dots, P\_n$, $$RP = \frac{P\_i -min(P\_1, P\_2, \dots, P\_n) }{max(P\_1, P\_2, \dots, P\_n)-min(P\_1, P\_2, \dots, P\_n) }$$ Where $P\_i$ is the pitch of note at position $t$.

\item [Interval from the previous note:] The interval between the current note and its previous note (in semitone). This represents the direction of the melodic line. See Fig. \ref{fig:interval} for an example. $$\Delta P^- = P\_{i} - P\_{i-1} $$

\item [Interval to the next note:] The interval between the current note and its previous note (in semitone). See Fig. \ref{fig:interval} for an example. $$\Delta P^+ = P\_{i+1} - P\_i$$

\begin{figure}[tp]

\begin{center}

\includegraphics[width=0.4\textwidth]{fig/interval\_arrow}

\end{center}

\caption{Interval from/to neighbor notes}

\label{fig:interval}

\end{figure}

\item [Note duration:] The duration of a note (quarter notes).

Grace notes have no duration in musicXML specification \cite{musicxml}. The reason for this is that grace notes are considered as very short ornaments that does not occupy real beat position. But zero duration is hard to handle in mathematic formulation. So we assigned a duration of a sixty-fourth note, because it's far shorter than all the notes in our corpus.

\item [Relative Duration with the previous note:] The duration of a note divided by the duration of its previous note. See Fig. \ref{fig:duration} for example.

For a phrase of $n$ notes with duration $D\_1, D\_2, \dots, D\_n$, $$RD^- = \frac{D\_i}{D\_{i-1}} $$ This feature is intended to locate local changes in tempo, such as a series of rapid consecutive notes followed by a long note, which will cause a discontinuity in this feature.

\item [Relative duration with the next note:] The duration of a note divided by duration of its next note. See Fig. \ref{fig:duration} for example.

$$RD^+ = \frac{D\_i}{D\_{i+1}} $$

\begin{figure}[tp]

\begin{center}

\includegraphics[width=0.4\textwidth]{fig/duration}

\end{center}

\caption{Relative duration with the previous/next note}

\label{fig:duration}

\end{figure}

\item [Metric position:] The position (beat) of a note in a measure. For example, under a time signature of $^4\_4$, if a measure consists of five notes, they will have metric position of 1, 2, 2.5, 3 and 4, respectively.

Metric position usually implies beat strength. In most tonal music, there exists a hierarchy of beat strengths. For example, for a time signature of $^4\_4$, the first note is usually the strongest, the third note is the second strongest, and the second and fourth notes are the least strong ones. %\cite?

\begin{figure}[tp]

\begin{center}

\includegraphics[width=0.4\textwidth]{fig/metrical}

\end{center}

\caption{Metric position}

\label{fig:metrical}

\end{figure}

\end{description}

%TODO: link to narmour group

\subsection{Performance Features}

Performance features are the expressive expressions we would like to learn from a performance. Performance features are extracted by calculating how the expression deviates from the nominal notation in the score.

Performance features includes:

\begin{description}

\item [Onset time deviation:]

A human performer usually adds conscious or unconscious rubato of their performance. The onset time deviation is the difference of onset timing between the performance and the score. Namely,

$$ \Delta O = O\_i^{perf} - O\_i^{score} $$ Where $O\_i^{perf}$ is the onset time of note $i$ in the performance, $O\_i^{score}$ is the onset time of note $i$ in the score.

However, the above formula assumes that the performance is played at exactly the same tempo assigned by the score. In reality, performers don’t always keep up with the speed of the score because of limited piano skills, or that the performer may speed up or slow down certain sections to express his interpretation on the music. Therefore, the performance should be linearly scaled to avoid systematic bias, We will present a solution to this issue in Section \ref{sec:normalize}.

\item [Loudness:] The loudness of a note. Measured by MIDI velocity level 0 to 127.

\item [Relative duration:]

The performed duration of a note divided by the nominal duration of the score.

$$ RD = \frac{ D\_i^{perf}}{D\_i^{score}}$$

\end{description}

% \section{Melodic Similarity and Sample Selection}

% Once a score is given to the system for playing, all samples in the database will be ranked by the melodic similarity with the given score. Here we use the melodic distance function provided by the MIDI Toolbox \cite{Eerola2004}, which is defined as follows:

% \begin{enumerate}

% \item Melodic contour is calculated by connecting each note's pitch, forming a piece-wise linear contour.

% \item Subtract the contour by its mean to preserve only the relative part.

% \item If the two phrases have different length, re-sample both phrases with fixed intervals so both of the phrase will have contour vector of the same length.

% \item The L1 norm (a.k.a Taxicab or Manhattan distance) of these two contour vectors is a similarity measure.

% \end{enumerate}

% The melodic contour is chosen because it yields the best results in finding similar melodies, as shown in \cite{Hoffmann-engl2005}.

%TODO: not included features: e.g. notation

\subsection{Normalizing Onset Deviation}

\label{sec:normalize}

\begin{figure}[tp]

\begin{center}

%TODO:Fig.:Normalization Schemes

\includegraphics[width=0.8\textwidth]{fig/prob\_onset\_diff}

\end{center}

\caption{Systematic bias in onset deviation }

\label{fig:normalizationprob}

\end{figure}

The previous section has pointed out that the onset deviation feature may face some difficulties, when the performer did not play at the exact tempo indicated by the score. As illustrated in Fig. \ref{fig:normalizationprob}, if the performance is played slower than expected, the deviation will grow larger and larger over time same, and vice versa. The systematic bias caused by the difference in total duration mixes up with the local deviation, For a long phrase, the onset deviation of the last notes can be as larger as a dozen of quarter notes. These kind of extremely large values will be learned by the model and cause erroneous predictions. A note may be delayed for a few quarter notes, and the notes will be played in the wrong order.

In other words, the onset deviation actually contains two types of deviations: A global/systematic deviation caused by the difference between performed and nominal tempos, and a local deviation caused by the note-level expression. Since the intention of the onset deviation feature is to capture the note-level expression, the performance must be linearly scaled to cancel out the global deviation.

Initially, we tried two possible ways of normalization:

\begin{enumerate}

\item Align the onset of the first notes, and align the onset of the last notes.

\item Align the onset of the first notes, and align the end (MIDI note-off event) of the last notes.

%\item Don't align the onset of the first notes, but align the onset of the last notes

%\item Don't align the onset of the first notes, but align the end of the last notes

\end{enumerate}

%The incentive for not aligning the first note is that the performer may intend to use an early start or delayed start as an expression, if the first note is aligned by it's onset, the first note in every phrase will have a onset timing bias feature of value zero. In other words, the early/delayed start expression is lost. %But each normalization method are equally reasonable theoretically, so we need to use empirical data to verify them. The experiment is explained in section \ref{TODO:experiment}. The experiment results are shown in [TODO: result]

However, neither of the method can robustly eliminate extreme values. Therefore, we propose an automated approach to find the best scaling ratio such that the normalized onset deviations in the performances fits best with those in the score. The measure of fitness is defined as the Euclidean distance between the normalized performance onset sequences and the score onset sequences, represented as vectors. %Note that the two vectors must have the same size, because the recordings are required to match note-to-note with the score.

Brent's Method \cite{brent1973} is used to find this optimal ratio. To speed up the optimization and prevent unreasonable local minima value, a search range of $[initial\ guess \times 0.5 , initial\ guess \times 2]$ is imposed on the optimizer. The $initial\ guess$ is used as a rough estimate of the ratio, calculated by aligning the first and last onsets. Than we assume that the actual ratio is not smaller than half of $initial\ guess$ and not larger than twice of $initial\ guess$. The two numbers 0.5 and 2 are chosen by trial and error, and most of the empirical data supports this decision. We will demonstrate the effectiveness of this solution in Section \ref{sec:onsetnormexp}.

%So we have defined an automatic method to dynamically adjust the normalization ratio to eliminate systematical error in the onset deviation feature. Comparing this method with the naïve approach not using any normalization (see Fig. \ref{fig:afternorm}), the adoptedmethod can produce very low onset deviations, roughly centered on zero, while the results by no normalization will show a clear trend of increasing deviations.

\chapter{Corpus Preparation}

\label{chap:corpus}

An expressive performance corpus is a set of performance samples. Since this research is based on a supervised learning algorithm, a high-quality corpus is essential to our success. Each sample consists of a score and its corresponding human recording. Some metadata such as results by phrasing, structure analysis, or harmonic analysis may be included, too. In this chapter, we will review some the existing corpora, specifications and formats of our corpus, and how we actually construct it.

\section{Existing Corpora}

Unlike other research fields like speech processing or natural language processing, there exists virtually no public accessible corpus for computer expressive performance research. CrestMusePEDB \cite{crestmuse} (PEDB stands for "Performance Expression Database") is a corpus created by Japan Science and Technology Agency's CREST program. However, until the time that this thesis is almost completed, we can't establish any contact with the database administrators to gain access to it. The corpus is claimed to have a GUI tool for annotate the expressive performance parameters from audio recordings. Their repertoire covers many piano works from well-known classical composers like Bach, Mozart, and Chopin, and the recordedmusic were performed by world-famous pianists. From their website \cite{crestmuse} they declared that the database contains the following data: PEDB-SCR - score text information, PEDB-DEV - performance deviation data and PEDB-IDX - audio performance credit. But the quality of the data is unknown.

Another example is the Magaloff Project \cite{magaloff}, which is created by some universities in Austria. They invited Russian pianist Nikita Magaloff to record all solo works for piano by Frederic Chopin on a Bösendorfer SE computer-controlled grand piano. This corpus became the material for many subsequent researches \cite{Goebl2009, Grachten2011, Flossmann2009, Grachten2012, Flossmann2013, Flossman2011, Flossmann2010a}. Flossmann, the prioncipal investigator of this project, also won the 2008 RenCon contest with a system based on this corpus called YQX \cite{yqx}. However, the corpus is not open to the public.

Since both corpora are not available, we need to implement our own. We will start by defining the specification.

\section{Corpus Specification}

The corpus we need must fulfill the following criteria:

\begin{enumerate}

\item All the samples are monophonic, containing only a single melody without chords.

\item No human error, such as insertion, deletion, or wrong pitch exists in the recordingso that the score and recording are matched note-to-note.

\item Phrasing is annotated by human.

\item The scores, recordings and phrasing data are in machine-readable format.

%\item The tempo label in MIDI recordings are the tempo by which the musician played.

\end{enumerate}

Some potentially useful information are not included because they are less relevant to our goal. Examples are:

\begin{enumerate}

\item Advanced structural analysis, such as GTTM (Generative Theory of Tonal Music)\cite{GTTM}

\item Harmonic analysis

\item Piano paddle usage

\item Piano fingering

\item Other instrument specific techniques, such as violin pizzicato, tapping, or bow techniques.

%\item Other instrumentspecific instructions, such as piano fingering, violin bow techniques etc.

\end{enumerate}

We choose Clementi's Sonatina Op.36 for our corpus, because it is a must-learn repertoire for piano students, so it's easy to find performers with a wide range of skill level to record the corpus. These sonatinas are classical works, so the learned model can easily be extended to other classical era works like Mozart and Haydn. There are six sonatinas included in Op.36, the first five have three movements each, and the last one has two movements. The titles and time signatures of all the pieces are listed in Table \ref{tab:cleminfo}

\begin{table}

\centering

\caption{Clementi's Sonatinas Op.36 }

\label{tab:cleminfo}

\begin{tabular}{lll}

\hline

\textbf{Title} & \textbf{Movement} & \textbf{Time Signature}\\

\hline

No.1 Sonatina in C major& I. Allegro &4/4\\

& II. Andante &3/4\\

& III. Vivace &3/8\\

No.2 Sonatina in G major& I. Allegretto &2/4\\

& II. Allegretto &3/4\\

& III. Allegro &3/8\\

No.3 Sonatina in C major& I. Spiritoso &4/4\\

& II. Un poco adagio &2/2\\

& III. Allegro &2/4\\

No.4 Sonatina in F major& I. Con spirito &3/4\\

& II. Andante con espressione &2/4\\

& III. Rondó: Allegro vivace &2/4\\

No.5 Sonatina in G major& I. Presto &2/2\\

& II. Allegretto moderato &3/8\\

& III. Rondó: Allegro molto &2/4\\

No.6 Sonatina in D major& I. Allegro con spirito &4/4\\

& II. Allegretto &6/8\\

\hline

\end{tabular}

\end{table}

%we choose scores to choose from, such as MusicXM \cite{Good2001}, LilyPon \cite{LilyPond}, Finale, Sibelius, ABC, MuseData, and Humdrum. The book \cite{Selfridge-Field1997} has a comprehensive review on this issue. %For research purpose, proprietary format like Finale and Sibelius is abandoned because of their limited support from open source tools.

MusicXML is used to represent Clementi's work in digital format.

MusicXML is a digital score notation using XML (eXtensible Markup Language), it can express most traditional music notations and metadata. Most music notation software and software tool supports musicXML format. %An example snippet of a musicXML score is shown in Fig. \ref{fig:expxml}%LilyPond is a \LaTeX-like language for music typesetting. %ABC, MuseData and Humdrum are based on ASCII codes and each defines their unique representation for music score.

Although MIDI is also a possible candidate for representing scores, it is designed to hold instrument control signal rather than notation, so some music symbols may not be available in MIDI. Furthermore, MIDI represents music as a series of note-on and note-off events, which requires additional efforts to transform them into traditional notations.

But for representing performance, MIDI is the most suitable format. Using a key-pressure-sensitive digital piano, pianist can record their performance in a natural way. The recordings have high precision in time, pitch and loudness (key pressure);and polyphonic tracks can easily be recorded separately. Although WAV (Waveform Audio Format) audio recording has a higher fidelity than MIDI, but they are harder to parse by computers. Without robust onset detection, pitch detection, and source separation technology, the information is extremely difficult to extract. Manually annotate each WAV recording takes much effort, and the accuracy across different annotators may not be consistent.

There is an impractical way to keep both the score and the recording in one single MIDI file. Instead of recording the actual note-on and note-off timing, we keep the nominal note-on and note-off in the score. Then, MIDI tempo-change events are inserted before each note to shift the performed timing of the recorded notes. Thus, the nominal time of each note represents the score, and the rendered time represents the performance. ButMIDI is limited as a score format, and requires complex calculations to recover the performance; this method is not used in the research.

Finally, we store the phrasing, which is the only metadata we used, in a plaintext file.Each line in the phrasing file stands for the starting point of each phrase. The starting point is defined as the onset timing (in quarter notes) counted from the beginning of the piece\footnote{For a phrase that start at a point which is a circulating decimal, for example $2\frac{1}{3}=2.333\cdots$, the starting point can be alternatively defined as any finite decimal between the end of the last phrase and the start of the current phrase. For example, if the last phrase stops at beat 1, the second phrase start at $2\frac{1}{3}=2.333\cdots$ beat, the start point of the second phrase can be written as 2.3 or 2.0, etc.}. The phrasing is decided according to the following principles:

\begin{enumerate}

\item Phrase may be separated by salient pauses.

\item Phrase may end with a cadence.

\item Phrase may be separated by dramatic changes in tempo, key or loudness.

\item Repeated structures in tempo or pitch may be a repeated phrase.

\end{enumerate}

Since phrasing controls the structural interpretation of a piece, we would like to leave this freedom for expression to the user. However, if there exists any good automatic phrasing algorithm, it can be easily integrated into the current system to make it fully-automatic.

\section{Implementation}

\subsection{Score Preparation}

The digital scores are downloaded from KernScore website \cite{KernScores}. The scores are transformed into MusicXML from the original Hundrum file format (.krn) using the music21 toolkit \cite{music21}. Because this research focuses on monophonic melody only, the accompaniments are removed and only theth highest-pitched note in chords are kept, which is usually a part of the most salient melody. The reduced scores are double-checked against a printed version publish by Durand \& Cie., Paris \cite{Clementi1915} to eliminate all errors. %We use MuseScore notation editor to view and edit MusicXML; some metadata errors are corrected by editing the MusicXML with text editors.

\subsection{MIDI Recording}

We have implemented two methods for recording: First, using a Yamaha digital piano to record MIDI. Second, by tapping on a touch-sensitive device to express tempo, duration and loudness. Due to accuracy consideration, only the recordings from Yamaha digital piano are used in the experiments.

In those experiments Yamaha P80 88-key graded hammer effect\footnote{Graded Hammer Effect feature provides realistic key pressure response similar to a traditional acoustic piano.} digital piano is played for recording. Through a MIDI-to-USB converter, the keyboard was connected to Rosegarden Digital Audio Workstation (DAW) software on a Linux computer. The Rosegarden DAW also generated the metronome sound to help the performer maintain a steady speed. Metronome is mandatory because if the performer plays freely, the tempo information written in the MIDI file will be invalid, which makes subsequent parsing and linear scaling very difficult. So the performers were asked to follow the speed of the metronome, but they can adjust the metronome speed as they like, and apply any level of rubato as long as the overall tempo is steady.

The second and not-chosen method, utilizes a touch-enabled input device like smartphone touchscreen or laptop touchpad. We have implemented a prototype using a Synaptics Touchpad on a Lenovo ThinkPad X200i laptop. When the user taps the touchpad once, one note from the score will be played, and the duration and loudness will be controlled by the duration and pressure of the tapping action. So the user can \enquote{play} the touchpad like a musical instrument. This idea has already been used in musical games and toys. This method is more user-friendly to general public because it requires minimal instrument skill and applies widely available hardware. But most touchpad estimates pressure by finger contact area, so the accuracy in pressure is not very satisfactory, though it is indeed a low-cost alternative to MIDI digital piano.

\subsection{MIDI Cleaning and Phrase Splitting}

After MIDIs are recorded, we use Python scripts to check if each recording is matched note-to-note with its corresponding score. If not, the mistakes are manually corrected. %For example, if a note was played to a wrong pitch, we would correct the pitch but keep its onset, duration and intensity.

If some small segments are totally messed up, they will be reconstruct using repeated or similar segments from the same piece. The matched score and MIDI pairs are then split into phrases according to the corresponding phrasing file. The split phrases are checked once again for note-to-note match before they are put into experiments.

\section{Results}

Six graduate students (not majored in music) were invited to record the samples. The number of mistakes they made are listed in Table \ref{tab:mistakes}.\footnote{The performers are allowed to re-record as many time as they want, so the actual number of mistakes may be higher.} These mistakes are identified using the unix \texttt{diff}\cite{diff} tool. Five of them (A to E) finished Clementi's entire Op.36, while performer F only recorded part of the work. The total number of recordings and the corresponding phrase/note counts are shown in Table \ref{tab:corpuscount}.

\begin{sidewaystable}[bp]

\centering

\caption{Number of mistakes in the corpus. Blank cell means the performer didn't record that part of the piece}

\label{tab:mistakes}

\begin{tabular}{c|rrrrrrrrrrrrrrrrr|r}

\hline

Performer&1-1&1-2&1-3&2-1&2-2&2-3&3-1&3-2&3-3&4-1&4-2&4-3&5-1&5-2&5-3&6-1&6-2&Subtotal\\

\hline

A&0&5&2&4&3&0&4&2&2&4&5&9&9&2&3&4&1&59\\

B&2&1&1&2&2&1&6&0&3&2&3&6&12&3&3&10&7&64\\

C&1&1&0&1&0&1&2&0&0&3&2&3&10&1&35&6&1&67\\

D&0&1&1&2&3&1&4&1&1&10&6&3&10&2&7&13&2&67\\

E&2&3&4&4&0&3&4&0&0&21&6&22&23&3&9&18&13&135\\

F&1&3&2&11&6&8&7&2&6&&15&&&&20&&&81\\

\hline

Subtotal&6&14&10&24&14&14&27&5&12&40&37&43&64&11&77&51&24&473\\

\end{tabular}

\end{sidewaystable}

\begin{table}[bp]

\centering

\caption{Total recorded phrases and note counts}

\label{tab:corpuscount}

\begin{tabular}{l|rrr}

\hline

\bf Title&\bf Recordings&\bf Total Phrases&\bf Total Notes\\

&\bf Count&&\\

\hline

No.1 Mov. I&6&72&1332\\

No.1 Mov. II&6&60&882\\

No.1 Mov. III&6&102&1566\\

No.2 Mov. I&6&108&1920\\

No.2 Mov. II&6&36&750\\

No.2 Mov. III&6&168&2484\\

No.3 Mov. I&6&156&3156\\

No.3 Mov. II&6&42&444\\

No.3 Mov. III&6&120&2628\\

No.4 Mov. I&5&80&2325\\

No.4 Mov. II&6&78&1332\\

No.4 Mov. III&5&85&1920\\

No.5 Mov. I&5&85&3360\\

No.5 Mov. II&5&70&1580\\

No.5 Mov. III&6&144&3384\\

No.6 Mov. I&5&145&4180\\

No.6 Mov. II&6&78&2754\\

\hline

Total&97&1629&35997\\

\hline

\end{tabular}

\end{table}

The number of phrases (according to our phrasing annotation) and notes are shown in Table \ref{tab:clemcount}. Fig. \ref{fig:notes} shows the length distribution of each movement. From the figure we can observe that most movements have around a few hundred notes, except the long No.6 and some short second movements. Fig. \ref{fig:phrases} shows the length distribution in numbers of phrases, most movements are around 20 phrases. The length distribution of the phrases in all six pieces is shown in Fig. \ref{fig:phrlength}, most phrases are shorter than 30 notes. Some very long phrases are usually virtuoso segments with very fast note sequences, so it's hard to split them further.

\begin{table}[bp]

\centering

\caption{Phrase and note counts for Clementi's Sonatina Op.36}

\label{tab:clemcount}

\begin{tabular}{l|rr}

\hline

\textbf{Title}&\textbf{Phrases Count}&\textbf{Notes Count}\\

\hline

No.1 Mov. I&12&222\\

No.1 Mov. II&10&147\\

No.1 Mov. III&16&261\\

No.2 Mov. I&18&320\\

No.2 Mov. II&6&125\\

No.2 Mov. III&28&414\\

No.3 Mov. I&25&526\\

No.3 Mov. II&6&74\\

No.3 Mov. III&19&438\\

No.4 Mov. I&25&465\\

No.4 Mov. II&12&222\\

No.4 Mov. III&16&384\\

No.5 Mov. I&17&672\\

No.5 Mov. II&13&316\\

No.5 Mov. III&24&564\\

No.6 Mov. I&28&836\\

No.6 Mov. II&11&459\\

\hline

\textbf{Total} &286&6445\\

\hline

\end{tabular}

\end{table}

\begin{figure\*}[tp]

\begin{center}

\includegraphics[width=\textwidth]{fig/notes}

\end{center}

\caption{Movement length (notes) distribution}

\label{fig:notes}

\end{figure\*}

\begin{figure\*}[tp]

\begin{center}

\includegraphics[width=\textwidth]{fig/phrases}

\end{center}

\caption{Movement length (phrases) distribution}

\label{fig:phrases}

\end{figure\*}

\begin{figure\*}[tp]

\begin{center}

\includegraphics[width=\textwidth]{fig/phrlength}

\end{center}

\caption{Phrase length (notes) distribution}

\label{fig:phrlength}

\end{figure\*}

%TODO:recording example pic

\chapter{Experiments}

\label{chap:exp}

In this chapter, we will show some experiment results to prove the effectiveness of our method. Section \ref{sec:onsetnormexp} deals with the onset deviation problem highlighted in Section \ref{sec:normalize}. Section \ref{sec:paramselect} discusses how various parameters in our system are chosen. Section \ref{sec:turing} describes a subjective test to check if audiences can or cannot identify the difference between the system-generated and human performances.

\section{Onset Deviation Normalization}

\label{sec:onsetnormexp}

As mentioned in Section \ref{sec:normalize}, a bad normalization usually results in unreasonable high onset deviation. To overcome this challenge, we proposed an automated way to select the best way of normalization. In this section, we will evaluate the effectiveness of this method.

We extract the onset deviation feature from performer E's recording\footnote{The effect of this method is less obvious for performer with better piano skill, because they have better control over tempo stability.}, using the two types of fixed normalization methods and the adopted automatic normalization. The onset deviations extracted by each method are shown in Fig. \ref{fig:norm1}, Fig. \ref{fig:norm3} and Fig. \ref{fig:normauto}. Each dotted line from left to right represents a phrase in the corpus. Each dot represents the onset deviation value of a note. The notes are spreaded uniformly +on the horizontal axis, which only shows the order of appearance, instead of the real time scale. First, we can see in Fig. \ref{fig:norm3} that by aligning the note-off events of the last notes results in very large deviations in some phrases, because extending the last note in certain phrases to emphasize the ending is a commonexpression way. This kind of extension will cause the last notes onset in the performance to be far apart from the score. Fig. \ref{fig:norm1} and Fig. \ref{fig:normauto} seem to work better. Although they look similar, but the onset deviation values in Fig. \ref{fig:norm1} is dramatically more than those in Fig. \ref{fig:normauto}, which shows that our automatic normalization method can reduce the onset deviations in general. Another benefit of the automated normalization method over the method aligning last notes onset is that the last notes are not forced to align, which allows more space for free expressionof the last note. This effect can be seen in Fig. \ref{fig:norm1}, in which the right-most end of a line, i.e. the last note, always goes back to zero, while in Fig. \ref{fig:normauto}, the end of a line can end in different values

\begin{figure}[tp]

\begin{center}

%TODO:Fig.:Normalization Schemes

\includegraphics[width=0.8\textwidth]{fig/lian\_onset\_2}

\end{center}

\caption{Onset deviations by aligninglast note onset}

\label{fig:norm1}

\end{figure}

\begin{figure}[tp]

\begin{center}

%TODO:Fig.:Normalization Schemes

\includegraphics[width=0.8\textwidth]{fig/lian\_onset\_4}

\end{center}

\caption{Onset deviations by aligning last notes note-off}

\label{fig:norm3}

\end{figure}

\begin{figure}[tp]

\begin{center}

%TODO:Fig.:Normalization Schemes

\includegraphics[width=0.8\textwidth]{fig/lian\_onset\_1}

\end{center}

\caption{Onset deviations using automated normalization method}

\label{fig:normauto}

\end{figure}

\section{Determination of Parameter Values}

\label{sec:paramselect}

\subsection{SVM-HMM-related Parameters}

There are many parameters which need adjustments in SVM-HMM. Two most important parameters, the termination accuracy $\varepsilon$ and the misclassification penalty factor C in SVM, are systematically tested in this experiment to find the optimal value. Since SVM-HMM is an iterative algorithm, the $\varepsilon$ parameter defines the required accuracy for the algorithm to terminate. A smaller $\varepsilon$ will result in higher accuracy, but may take more iterations. The C parameter determines how much weight should be assigned to penalized non-separable samples. A larger C will sacrifice larger margin for lower misclassification error, but it will make the execution time longer.%Therefore, we will leave the rest of the parameters to their default value, and try to find the best C parameter.l

A's performance recordings are split into two sets: the training set including pieces No.2 to No.6, and the testing set including piece No.1. We train a model with the training set, and use the learned model to generate the testing set. The generated expressive performance is compared to the corresponding human recordings to calculate the accuracy of the prediction.

Ideally, the generated performance will be very similar in expression to the recording. In order to choose the best $\varepsilon$, we calculate the median of similarities between the generated and recorded performances for each $\varepsilon$ choice. Note that each performance feature has its own model, so we will be looking at one performance feature and its $\varepsilon$ parameter at a time.

First, the generated performance feature sequence and the recorded one are normalized to a range from 0 to 1, because the generated performance may have the same up-and-downs as the score, but the value ranges may be different, so we use normalization to ease our these difference. The Euclidean distance between the two normalized sequences is calculated and divided by the length (number of notes) of the phrase, since the phrase can have arbitrary length. Similar procedure is applied to find the best C.

First we fixed C at $0.1$ and tried different $\varepsilon$'s: $100$, $10$, $1$, $0.75$, $0.5$ and $0.1$. Then, we fix $\varepsilon$ at the optimal value determined in the previous step and test different C's: $10^{-3}$, $10^{-2}$, $10^{-1}$, $0.5$, $1$, and $5$. For each $\varepsilon$ and C combination, we calculate the distance between the generated pieces and recorded examples for all phrases in the testing set for each performer. Then we take the median of all these distances for each $\varepsilon$ or C. The optimal $\varepsilon$ or C is the one that minimize the median of the distances.

The median distance of the generated performance from the recording for various $\varepsilon$'s are shown in Fig. \ref{fig:eps\_accu}. The execution time for various $\varepsilon$'s are shown in Fig. \ref{fig:eps\_time}. For $\varepsilon$ values 100 and 10, the termination criteria is too generous so SVM-HMM terminates almost immediately without actually learned anything. Therefore, the outputs are a fixed value for any input. We abandon the data points for $\varepsilon = 100$ or $10$. We can see that the distance drops slowly when $\varepsilon$ becomes smaller. We choose $\varepsilon = 0.1$ for the best accuracy-time tradeoff. %But for the interval after $\varepsilon$ being smaller than 0.5, the accuracy doesn't drop anymore. So we will choose $\varepsilon = 0.5 $ for the rest of the experiments to avoid unnecessary computations.

\begin{figure\*}[tp]

\begin{center}

%TODO:Fig.:Example JSON code

\includegraphics[width=\textwidth]{fig/eps\_accu.eps}

\end{center}

\caption{Median distance between generated performances and recordings for different $\varepsilon$'s}

\label{fig:eps\_accu}

\end{figure\*}

\begin{figure\*}[tp]

\begin{center}

%TODO:Fig.:Example JSON code

\includegraphics[width=\textwidth]{fig/eps\_time.eps}

\end{center}

\caption{Execution time for different $\varepsilon$'s}

\label{fig:eps\_time}

\end{figure\*}

As for different C parameter, the accuracy and execution time are shown in Fig. \ref{fig:c\_accu} and Fig. \ref{fig:c\_time}, respectively. We can't find a clear trend in Fig. \ref{fig:c\_accu}, but we can find that for C over $10$ and under $0.01$, the model failed to produce meaningful model (i.e. the output is a fixed value), so the data point is omitted in the figure. Therefore, choosing a C in the middle will produce more robust model. In Fig. \ref{fig:c\_time} the execution time grows as C goes larger. After considereing the robustness (always producing meaningful model) and time tradeoff, we choose C = 0.1 as our optimal C.

\begin{figure\*}[tp]

\begin{center}

%TODO:Fig.:Example JSON code

\includegraphics[width=\textwidth]{fig/C\_accu}

\end{center}

\caption{Median distance between generated performances and recordings for different C's}

\label{fig:c\_accu}

\end{figure\*}

\begin{figure\*}[tp]

\begin{center}

%TODO:Fig.:Example JSON code

\includegraphics[width=\textwidth]{fig/C\_time}

\end{center}

\caption{Execution time for different C's}

\label{fig:c\_time}

\end{figure\*}

\subsection{Quantization of Parameter values}

Besides $\varepsilon$ and C, the number of quantization levels for SVM-HMM input also has some impact on the execution time. If the performance features are quantized into more fine-grained levels, the quantization errors can be reduced, but the execution time and memory usage will grow dramatically. Also, larger number of intervals doesn't imply a more accurate or robust model. Because SVM-HMM is originally used in part-of-speech tagging, if we divide the performance features into more intervals, there will be fewer samples in each interval. From a statistical learning point of view, it is desirable to have fewer bins with more samples in each bin, rather than a large number of bins with very sparse samples in each bin. To illustrate this point, consider a three-note segment played once in the fMIDI velocity: (60, 70, 80), and the same segment is played again in (60.1, 69.9, 80.1). If we have a quantization interval width of, say, 0.05, then 60 and 60.1 may be quantized into different bins, and 70 and 69.9 may also be quantized into different bins, so the two phrases will be considered as two different cases. However, if the quantization interval width is 1, both phrases may be quantized into the same label sequence, which is more desirable because the SVM-HMM algorithm can capture the similarity in the two samples.

Initially, we quantized the values into 1025 uniform bins, wishing to minimize the quantization error. But it takes very long time (hours, even days) to learn a model, and the output only falls on a very sparse set of values. So we reduce this number to 128. This level of quantization is fine enough to capture the performance nuance. Taking a rough estimate, onset deviation feature rarely exceeds $\pm 1$, so the quantization interval width is around $\frac{1-(-1)}{128} = 0.015625$. Most duration ratios falls between zero and three, so the interval width is $\frac{3-0}{128} = 0.0234375$. MIDI velocity is roughly around 30 to 90, so the interval is about $\frac{90-30}{128} = 0.46875$. This level of granularity is good enough for our performance system, and can dramatically reduce the execution time without sacrificing the expressiveness of the models.

We repeated the $\varepsilon$ selection experiment for quantization level of 1025 and 128. The execution time (in CPU second) is shown in Fig. \ref{fig:quant\_comp}. The time required for 1025 is longer than 128 by orders of magnitudes, but the expressiveness does not improve much.%The expressiveness of the output is even improved (evaluated by subjective listening).

\begin{figure}[tp]

\begin{center}

%TODO:Fig.:Normalization Schemes

\includegraphics[width=\textwidth]{fig/quant\_comp}

\end{center}

\caption{Execution time for differnt number of quantization levels}

\label{fig:quant\_comp}

\end{figure}

\section{Human-like Performance}

\label{sec:turing}

The goal of our system is to create expressive, non-robotic music as an opposite to deadpan MIDI. Therefore, we need to perform a subjective test to verify if people can tell our generated performances apart from real human performances.

In this survey, 1518 computer generated expressive phrases and their corresponding human recordings were selected as samples. Each test subject was given 10 randomly selected computer generated phrase and 10 human recordings, these 20 phrases are presented in random order. He/She was asked to rate each phrase according to the following criteria, which were proposed by the RenCon contess \cite{RenCon}:

\begin{enumerate}

\item Technical control: How does the performance sound like one generated by a technical system, and thus performed with accurate and secure notes, rhythms, tempo and articulation.

\item Humanness: Does the performance sound like one played by a human.

\item Musicality: How musical is the performance in terms of tone and color, phrasing, flow, mood and emotions

\item Expressive variation: How much expressive variation (versus deadpan) is in the performance.

\end{enumerate}

In RenCon, each judge was asked to give separate ratings for each criterion. We believe that this is too demanding for less-experienced participants, so we asked each test subject to give an overall rating from one to five. One means very bad, and five stands for very good. The test subjects are also asked to report their musical proficiency in a three level scale:

\begin{enumerate}

\item No experience in music

\item Amateur performer

\item Professional musician, musicologist or student majored in music

\end{enumerate}

To generate the expressive performance phrase correctly, we follow a six-fold cross-validation pattern:For each performer in the corpus, we use all his/her recorded phrases of Clementi's Op.36 No.2 to No.6 to train a model. Then the model is used to generate all phrases from Clementi's Op.36 No.1. The generated phrases and the performer's recordings of piece No.1 are allincluded as samples. The process is repeated, but each time the piece excluded for training will be changed to No.2, No.3 and so on. So all six pieces will have a computer-generated version (trained by each player's corpus) and a recorded version.

We have also tried using all performers' recordings to train a single model. However, the expressive variation from that model is much smaller than a model trained by a single performer's recordings. This is because expression from different performers may cancel each other out. This phenomena can be found in the distribution histograms for each performance features (Fig. \ref{fig:distonset}, Fig. \ref{fig:distdur} and Fig. \ref{fig:distvel}). The features generated from the full corpus are slightly more concentrated, which results in less dramatic expression.

\begin{figure}[tp]

\begin{center}

%TODO:Fig.:Normalization Schemes

\includegraphics[width=0.8\textwidth]{fig/all\_01\_onset}

\end{center}

\caption{Distribution of onset deviation values from full corpus versus single performer's corpus}

\label{fig:distonset}

\end{figure}

\begin{figure}[tp]

\begin{center}

%TODO:Fig.:Normalization Schemes

\includegraphics[width=0.8\textwidth]{fig/all\_01\_duration}

\end{center}

\caption{Distribution of duration ratio values from full corpus versus single performer's Corpus}

\label{fig:distdur}

\end{figure}

\begin{figure}[tp]

\begin{center}

%TODO:Fig.:Normalization Schemes

\includegraphics[width=0.8\textwidth]{fig/all\_01\_velocity}

\end{center}

\caption{Distribution of MIDI velocity values from full corpus versus single performer's corpus}

\label{fig:distvel}

\end{figure}

We received 119 valid samples for the survey. Fifty of them are from people with no music background, 59 are from amateur musicians, and the rest 10 are from professional musicians. The average ratings given to computer generated performances and human recordings are listed in Table \ref{tab:avg\_rating}. It is clear that for professional and amateur musician, the average rating given to human performances is higher than that given to computer performances. However, for participants who have no experience in music, the ratings are much closer. A Student’s *t*-test on the two ratings given by participants with no experience yields a p-value of 0.0312, therefore we can't reject the null hypothesis that the two ratings are different under a significance level of 99\%. Therefore we can say that for participants with no music experience, the computer generated music and human recordings are indistinguishable.

\begin{table}

\centering

\caption{Average ratings for generated performance and human recording}

\label{tab:avg\_rating}

\begin{tabular}{r|rr}

\hline

&Computer &Human \\

\hline

No experience&3.243&3.391\\

Amateur&2.798&3.289\\

Professional&2.430&3.010\\

\hline

Total&2.952&3.306\\

\hline

\end{tabular}

\end{table}

In order to get more insight from the ratings, we can further divide the performers in the corpus into two categories by their piano skill level. By the number of mistakes made (Table \ref{tab:mistakes}), performer A and B are considered more skillful than performer C, D, E and F. The average ratings given to the performances generated from the model trained by samples of the two categories are listed in Table \ref{tab:good-bad\_rating}. The distances between computer and human performances are smaller for a less-skillful group (C to F) than the skillful group (A and B). A possible reason is that our system makes some mistakes that are similar to the mistakes made by less-skillful performers. For example, unsteady tempo, sudden change in loudness, and hesitation are common problems in both of the less-skillful performances and the computer generated performance. For experienced listenners, who have better technical control and have better sense of musical structure, the problems described above happen rarely. This makes the generated works sound much worse comparing to the better performance.%This may be regarded as that our system still can't capture every nuance of human performances.

\begin{table}

\centering

\caption{Average ratings for generated performance and human recording under different part of the corpus}

\label{tab:good-bad\_rating}

\begin{tabular}{r|rr|rr}

\hline

&A,B&&C-F&\\

&Computer &Human &Computer &Human \\

\hline

No experience&3.067&3.302&3.363&3.451\\

Amateur&2.680&3.347&2.863&3.286\\

Professional&2.048&3.162&2.708&2.921\\

\hline

Total&2.776&3.313&3.066&3.323\\

\hline

\end{tabular}

\end{table}

If we look into each individual participant, we can check if a participant gives higher (average) rating to computer or human performances, or equal ratings for both. The number of participants who fall into each categories are shown in Table \ref{tab:avg\_count}. Twenty-six of the non-experienced participants give higher or equal rating to computer than to human, slightly higher than twenty-four people who gives higher rating to human. For amateur and professional musicians, the number of people who prefers human is much higher. In Table \ref{tab:good-bad\_count} the generated performances are split into two categories just like Table \ref{tab:good-bad\_rating}. The results are similar to Table \ref{tab:avg\_count}: the difference between computer and human is larger for skillful performers (A and B) than less-skillful performers (C to F).

Therefore we can conclude that our system has the same expressive power for participants with no music background. But for amateur and professional musician, the system requires further improvements to be comparable to human musician.

\begin{table}

\centering

\caption{Number of participants who gives higher rating to generated performances, human recordings or equal rating}

\label{tab:avg\_count}

\begin{tabular}{r|rrr|r}

\hline

&Computer&Equal&Human&Total\\

\hline

No experience&19&7&24&50\\

Amateur&7&3&49&59\\

Professional&1&1&8&10\\

\hline

Total&27&11&81&119\\

\hline

\end{tabular}

\end{table}

\begin{table}

\centering

\caption{Number of participants who gives higher rating to generated performances, human recordings or equal rating under different part of the corpus}

\label{tab:good-bad\_count}

\begin{tabular}{r|rrr|rrr|r}

\hline

&&A,B&&&C-F&&Total\\

&Computer&Equal&Human&Computer&Equal&Human&\\

\hline

No experience&5&4&6&14&3&18&50\\

Amateur&2&1&18&5&2&31&59\\

Professional&0&1&3&1&0&5&10\\

\hline

Total Result&7&6&27&20&5&54&119\\

\hline

\end{tabular}

\end{table}

\chapter{Conclusions}

We have created a system that can perform monophonic score expressively. The expressive performance knowledge is learned from human performance recordings using structural support vector machine with hidden Markov model output (SVM-HMM). We have also created a corpus consisting of scores and MIDI recordings. From our subjective test, we showed that although the amateur and professional musician can still differentiate generated performances from human recordings. Test subjects without music background rated equal (with statistical significance) to the generated performances and human recordings. This can be explained as that our system has the same expressive power as human has for general listeners who has no music background..

There are still much room for improvement. Structural expressions such as phrasing, contrast between sections, or even contrast between movements can be added, which requires automatic structural analysis. Other information like text notations, harmonic analysis and other musicological analysis can be added to the learning process. Supporting homophonic or polyphonic music is also important for the system to be useful. Sub-note expressions like physical model synthesizer or envelope shaping can also be applied to generate performances for specific musical instruments. It's also crucial to test the system on more samples of different genre or music style. We also believe that combining rule-based model and machine-learning model may be a possible direction for computer expressive music performance research. With rules serving as a high level guideline for structural expression, the machine-learning model can focus more on note or sub-note level expressions, anduserser can gain more control by tweaking the rules to interpret a piece of music in her/his style..