Robotic Musicianship
CS 7633, Fall 2016 - Final Project

Matthew Barulic

December 6, 2016

1 Background

Playing music in an ensemble presents many interesting human-robot interaction challenges. One particularly interesting challenge is communication between a human and robot performer. The sound of the music should not be interrupted by speech between the human and the robot and non-verbal communication is limited as a person's hands and face are usually involved in playing the instrument. More subtle forms of non-verbal communication must be used: slight head motion, swaying of the body, etc.

In "Synchronization in Human-Robot Musicianship" [1], Georgia Tech's Shimon robot demonstrates the ability to play music accompanying a human musician. Specifically, they describe a system that allows the robot to play a response phrase to the human musician's call phrase, matching tempo and communicating with the human through physical gestures. They demonstrate a variety of relationships between levels of robot embodiment and the human player's ability to adapt to errors in the robot's timing.

2 Method

For this project, I attempted to implement and extend the work presented in [1]. This involved the design and construction of my own robotic musician, creation of software to mimic the call and response behavior demonstrated in [1], and a small experiment to demonstrate the robot's abilities.

2.1 Hardware Implementation

Much of the time spent on this project focused on constructing a robotic musician, called BassBot. I chose bass guitar as the instrument the robot would play for a few reasons:

- I have access to a bass guitar to use for the project.
- I wanted the robot to play a different instrument than the human player, ruling out playing piano.
- Of the stringed instruments I own, the bass guitar has the fewest strings, reducing the number of actuators needed to play it.
- Because most songs don't require the bass to play two strings at once, I can use a simple design, such as the roller used in BassBot, to adjust the note each string plays.

The full robot was designed in Autodesk Inventor CAD software before being built. All actuators and sensors are from the VEX Robotics line of products primarily because I already had a large stock of these parts on hand. The main body of the robot is cut from common board lumber. Other materials used include structural components from VEX Robotics, several 3D printed custom parts, and a laser-cut, acrylic mount for the servos.

BassBot works by moving the roller carriage along the neck to clamp all strings at a specified fret, then the bank of servos at the base of the neck pluck individual strings using custom-cut guitar picks. The carriage is actuated via a timing belt pulled by the motor at the top of the neck. Because the carriage clamps to the neck of

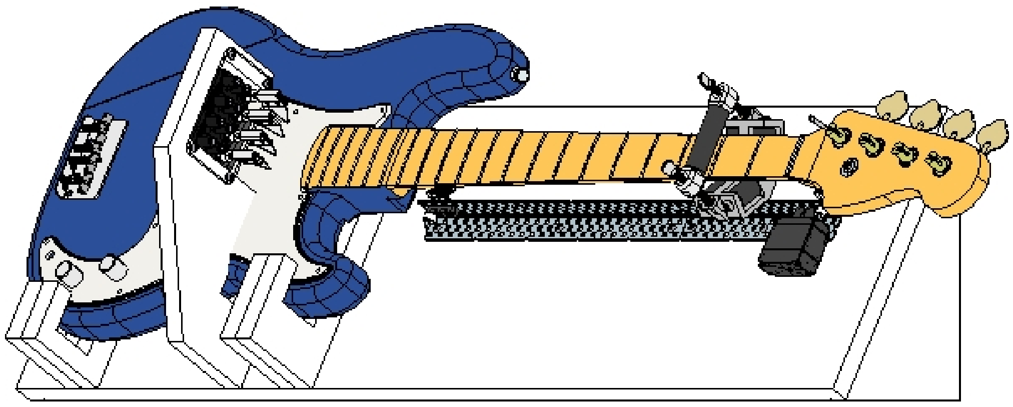
Figure 2.1: CAD design of BassBot completed in Autodesk Inventor

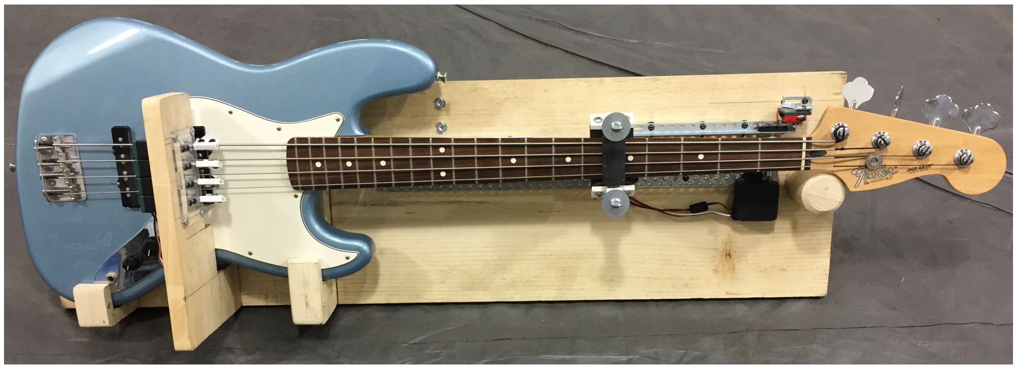
Figure 2.2: Fully constructed BassBot

the instrument, no guiding rails are necessary to keep the carriage aligned. The robot keeps track of the carriage position with an integrated encoder module on the motor. A limit switch is installed at the top of the neck and is used to zero the encoder with the carriage at the first fret. This zeroing process must be done every time the robot is turned on.

Custom picks were cut from Delrin acetal resin, a material commonly used in professional guitar picks. These picks included features to attach them to the servo output shafts with 3D-printed clamps.

The robot is controlled by an Arduino UNO mounted on the rear of the main body. A custom PCB I had previously designed is used to interface between the Arduino and VEX products. Details on that board are available at [2].

2.2 Software Implementation

The BassBot software is designed and implemented on top of the Robot Operating System (ROS) framework [4]. This allows the design to be highly modular and offers a large bank of community-maintained components. Figure 2.3 shows the high-level structure of the ROS nodes in the BassBot package.

The MIDI Listener node connects to a MIDI input device, in this case a keyboard, using the RtMidi library [3]. Whenever a note is played on the device, the node publishes a message of type $bassbot/MidiNote$. This message includes the MIDI note number of the note played, the velocity with which the note was played, and a timestamp of when the note was played.

The Call Finder node subscribes to the $/midi$ topic from MIDI Listener and looks for the predefined call pattern in the stream of MIDI messages. The pattern is defined in a text file which includes MIDI note numbers and note durations in beats. Pattern matching ignores these durations and simply looks for the right notes played in the right order. Once the pattern is found, the note durations are used to calculate the tempo at which the

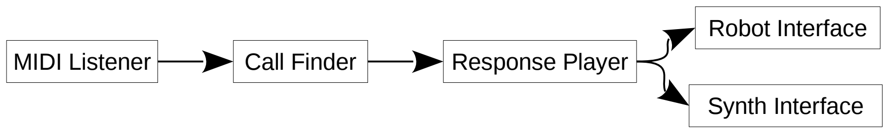

Figure 2.3: High-level structure of nodes and connections in BassBot software

phrase was played using the formula

$$T = \frac{\sum_{i=1}^{n-1} d_i^C}{t_n - t_1} \tag{2.1}$$

where n is the number of notes in the pattern, d_i^C is the duration of the ith note in the call pattern, and t_i is the timestamp of the ith note. Once the call pattern is found, the Call Finder node publishes a $bassbot/CallFoundNotification$ message, which includes the tempo of the call phrase and a timestamp for the first downbeat in the call phrase.

The Response Player node is responsible for generating the playback commands for the response phrase. It receives the $bassbot/CallFoundNotification$ messages from Call Finder and publishes $bassbot/Note$ messages on the $/play$ topic. When a call notification is received, the downbeat timestamp for the response phrase is calculated as

$$t_R = t_C + \sum_{i=1}^{n} d_i^C \tag{2.2}$$

where t_C is the downbeat timestamp of the call phrase. The timestamp for each note in the response phrase is calculated using the formula

$$t_i^R = t_R + \frac{\sum_{j=1}^{i} d_j^R}{T} \tag{2.3}$$

Each $bassbot/Note$ message includes the MIDI note number and the timestamp at which the note should be played. The Response Player node also includes the mechanisms for simulating errors in the robot's timing for the purposes of the experiment discussed below.

The Robot Interface node connects to the robot through a serial connection to the Arduino. Every message received on the $/play$ topic is queued and sorted by timestamp. Each note is separated into a fret move and string pluck. Commands to the Arduino are sent with enough time to account for serial communication delays and time needed to execute the action (ie. roll the carriage to the correct fret). This is a simple, yet effective way to ensure that notes played by the robot are in time with the tempo the computer is tracking.

The Synth Interface node receives messages on the $/play$ topic and uses the $sound\_play$ ROS package to generate synthesized bass guitar sounds. These sounds were sampled from the same bass guitar the robot is playing in order to match the audio as best as possible.

2.3 Demonstrative Experiment

While this project is not intended to be a full user study, it is important to demonstrate the robot's ability to replicate the behavior discussed in [1]. To this end, I replicated a similar experimental setup for a single test user. The test included six different conditions across two variables. The first variable is the level of embodiment for which there are two conditions:

- **Synth** The robot is turned off and playback is synthesized over speakers
- **Robot** The robot is turned on and physically playing the instrument.

The original proposal included a third level involving an emotive head attached to the robot, but technical difficulties prevented this addition from working correctly. The second variable is the type of timing error including three conditions:

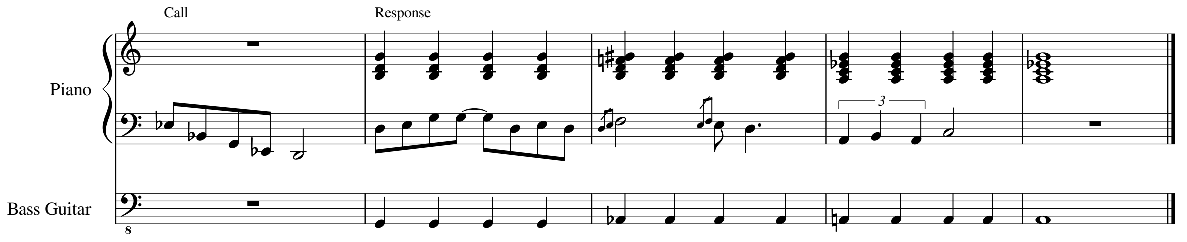

Figure 2.4: Music phrases used for the single-user experiment

- **Straight** The robot matches the human player's tempo.
- **Slow** The robot plays a constant 20 BPM slower than the human player's tempo.
- **Ramp** The robot starts at the human player's tempo but slows down 5 BPM ever 4 beats.

The call and response phrases actually chosen for the experiment were selected from George Gerswhin's *Rhapsody in Blue* mostly because the test subject was already knew the piece and a simple bass line could be arranged for it. Figure 2.4 shows the specific phrases used.

2.4 Results

Unfortunately, a failure in the data collection system after the experiment resulted in a loss of data from the human playback. This prevents me from giving numeric results from the experiment. Anecdotally, physical embodiment did seem to improve synchronization between performers. The human player reported watching the movement of the carriage to anticipate the robot's timing, which was not possible in the synthesized condition. The smaller tempo changes in the ramp timing condition seemed to improve synchronization as well, though lack of playback data prevents any numeric measures on that condition.

3 Discussion

Designing and implementing a robotic musician presents many challenges. Many simplifying decisions were made in the hardware design such as the decision to set all strings to the same fret rather than actuating each individually. This, combined with the relative slowness of the VEX Robotics motor hardware limits the kinds of phrases the robot can play.

Coordinating timing between hardware and software components was a difficult but critical task. Breaking the note playing up into fret moves and string plucks mirrors the original research's gesture-based approach. This was an important part of making sure the robot could keep up with the timing of the human player. Had my implementation been a bit more generalized, adding the emotive head may have been easier by simply including it in the same action queue as the note playing commands.

References

[1] G. Hoffman and Weinberg G. "Synchronization in Human-Robot Musicianship". In: *The 19th International Symposium on Robot and Human Interactive Communication (RO-MAN 10)* (2010).

[2] Matthew Barulic. *VEX Shield for Arduino*. 2014. URL: `http://barulicm.github.io/projects/arduino_vex_shield.html`.

[3] Gary P. Scavone. *RtMidi*. 2016. URL: `https://github.com/thestk/rtmidi`.

[4] Open Source Robotics Foundation. *Robot Operating System (ROS)*. URL: `http://www.ros.org/`.