392255 ISY Project: Dexmo hand exoskeleton teaches how to play piano

Tobias Coppenrath
tcoppenrath@techfak.uni-bielefeld.de

Jessica Seidel
jseidel@techfak.uni-bielefeld.de

Janneke Simmering
jsimmering@techfak.uni-bielefeld.de

Summer Term 2020

1 Abstract

"Learning a new skill, such as playing sports or a musical instrument is a great challenge. To be successful, the learner needs an extensive supervision of an expert and a high training motivation. In this project we will employ *Dexmo* hand exoskeleton [1], [2] to guide the hand, to motivate the learner and by this to accelerate the training process." – ISY project description

The goal was to create a prototype program that gives haptic feedback via Dexmo to the user for a random task, i.e. a music piece, which allows them to practice playing the piano. The user's improvement, i.e. his learning success, can then be determined by calculating the error between the music piece to be played and the actually played notes.

2 Introduction

Anyone who has already started learning how to play the piano will certainly have come across a number of learning programs designed to facilitate this process. Most programs (for example *flowkey* [3] or *OnlinePianist* [4]) proceed in such a way that a music piece is given, which the user plays and then an error is computed and displayed. The incoming data, i.e. the notes played by the user, are transmitted to the program via the keyboard's MIDI port. This is what we wanted to take up in this project and extend it with haptic feedback via Dexmo in such a way that the learning task is simplified and optimized, if possible.

The feedback to Dexmo and the choice of the next piece of music should no longer be randomly chosen at a later stage of the program's development. Instead, a new task corresponding to the user's abilities should be generated. This should support and improve the learning success of the user according to the scaffolding principle [5]. This principle describes an improved learning process through support and guidance, whereby this "scaffolding", i.e. the support, is removed step by step to enable the user to solve the tasks independently.

3 Related work

The idea of supporting motor skill learning through haptic feedback or haptic guidance is not completely new. Several other projects and studies have also dealt with this topic.

That haptic feedback is an advantage in learning motor skills was shown in a study which had a LEGO biplane model constructed with and without force feedback [6]. Subjects who had force feedback completed the biplane in the virtual world twice in 30 minutes on average. Subjects who trained without haptics only achieved that once the same period [6].

Another project was a slalom ski simulation where the next movement was announced by feedback [7]. It was found that 100% feedback is more effective for more complicated motor tasks, while 50% feedback delivers better results in case of simpler movements. Therefore it might make sense for us to give feedback for all notes in the beginning and only generate feedback on difficult parts at a later stage.

A further approach to the topic of learning motor skills with guidance was scaffolding in neurosurgery [8]. The participants were trained with three practices: guidance, demonstration and modeling. The result of this evaluation showed that feedback on the process is more important than feedback on the end result, which encourages own initiative when learning. For our project, this could mean to maybe give no feedback on mistake but rather before a tone as a hint or while playing, and not only after the task is finished.

Some studies tested motor learning with both haptic and visual feedback. Most of them show that visual feedback alone is very useful [9], even more useful than haptic guidance alone, but best is a combination of both. However, a distinction must be made here between which motor skills are learned. Visual training was more effective w.r.t. position and trajectory measures, haptic training was more effective w.r.t. timing [10].

Haptic training has also been compared to artificial error amplification with regard to each one's impact on learning [11]. The task was a time-based reaction test requiring a certain motion of the test subject. The conclusion was that both methods promoted learning, but there was a difference between the user's skill levels: Error amplification had a greater benefit for more skilled subjects, whereas haptic guidance seemed to have a greater learning impact for less skilled ones.

A further experiment also deals with the error analysis of music pieces with haptic support [12]. Here, different dimensions like number of notes, length in seconds, loudness etc. are considered, which requires several different evaluation metrics and especially onset times normalized to begin at time 0. The timing precision measure and the velocity measure are also discussed in detail.

Another interesting paper in the context of this project deals with adaptive firmness of guidance [13]. There, subjects performed slightly better when the feedback was fitted to their needs instead of being fixed. The conclusion for that experiment was that haptic guidance can lead to a short-term learning improvement as well as to decreasing the amount of errors in training periods.

4 Method

The project consists of multiple functional components which are described in this section. A detailed representation of the program structure is shown in the diagram in figure 1.

The user interacts with the graphical user interface (GUI). Its main functions are to parameterize and play the tasks and guidance, display the sheet music and visualize the error. In the diagram, these functions are indicated by some of the arrows pointing towards the actor. The displayed sheet music is generated separately by module 5 in the diagram, the process is explained in section 4.2. The mapping between the generated notes and the respective fingers is part of the MIDI generation module (number 5) and is described in section 4.3. The next parts, see section 4.4 and section 4.5, describe the functionality of Dexmo and its client, module 8 in the diagram, and the communication between the system components. Afterwards, section 4.6 characterizes the error computation which corresponds to module 6 in the diagram and section 4.7 explains which files are generated and how they are stored.

4.1 Graphical User Interface (GUI)

First of all, note that the GUI has no direct representation in figure 1 but takes place at the relations between the following boxes: $1 \rightarrow 4$, $6 \rightarrow 1$ and $7 \rightarrow 1$.

When the program is started, the GUI shows up immediately. The first window contains the MIDI port selection for the I/O devices, i.e. Dexmo, the

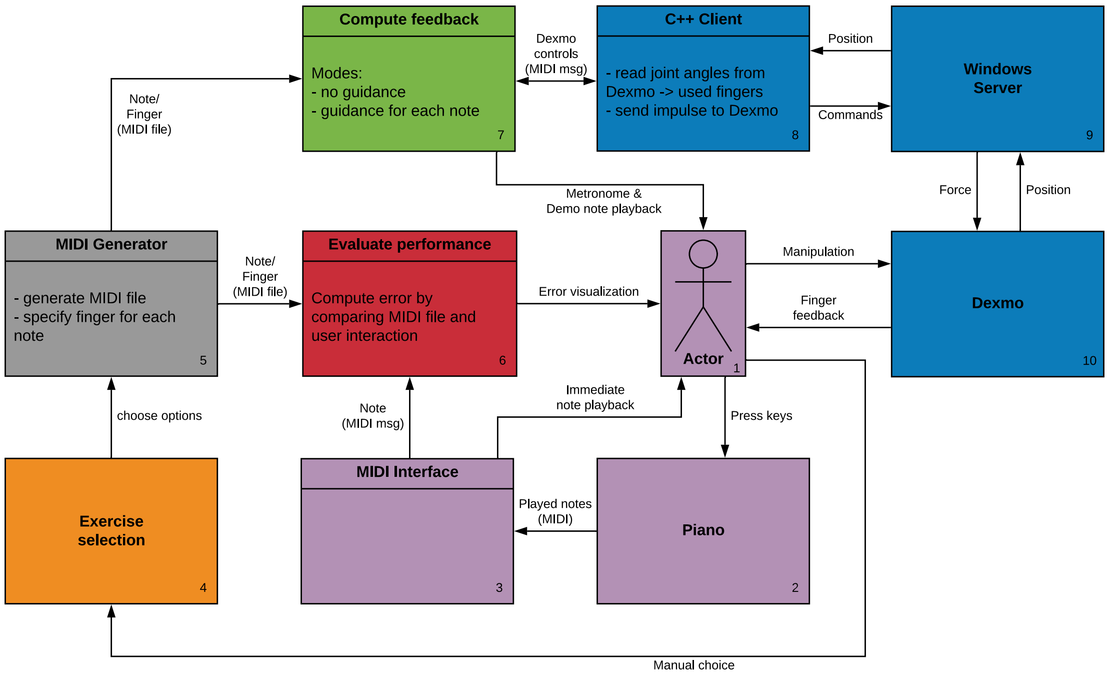


Figure 1: Overview of the program's main components

keyboard and the audio output (e.g. a synthesizer like *Qsynth*), as shown in figure 2. The ports of some devices, for example Dexmo, can be detected automatically and will then be preselected. Regardless of that, the user is always able to set each port manually. When the selection is done, the respective button invokes a switch to the main window.

The main window (see figure 3) shows the current task's sheet music, a graphical error representation and several buttons/interaction elements. The shown score is converted from MIDI using *LilyPond* [14]. At startup, a task is instantly generated with the default settings and also displayed. A new one can be generated by clicking the respective button. To change the current settings, the button *Specify next Task* opens the options window which will be explained below. The adjusted settings will be applied to the next generated task. It is also possible to use a custom MIDI file. Clicking on that button will show a file dialog in which the desired file can be selected. Note that it will be displayed a bit differently, since it has to be processed internally to provide having a metronome and automatic fingering; both are explained in section 4.2. Using MIDI files containing too few notes as well as less hand tracks than the program is currently set to can both not be handled properly yet; an error message will be shown in that case. The slider at the bottom can be used to adjust the bpm with which the custom MIDI file will be played. Note that this will not change the tempo number in the displayed sheet music as it would need to be modified

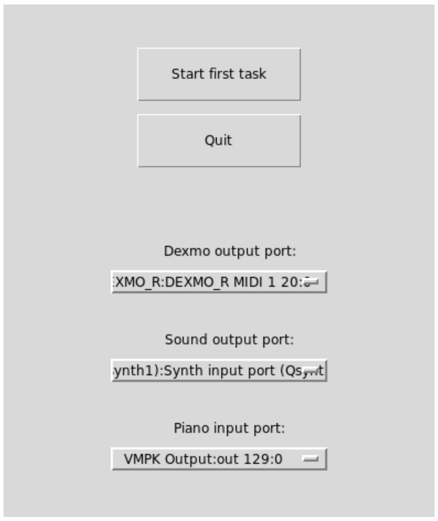


Figure 2: MIDI port selection window

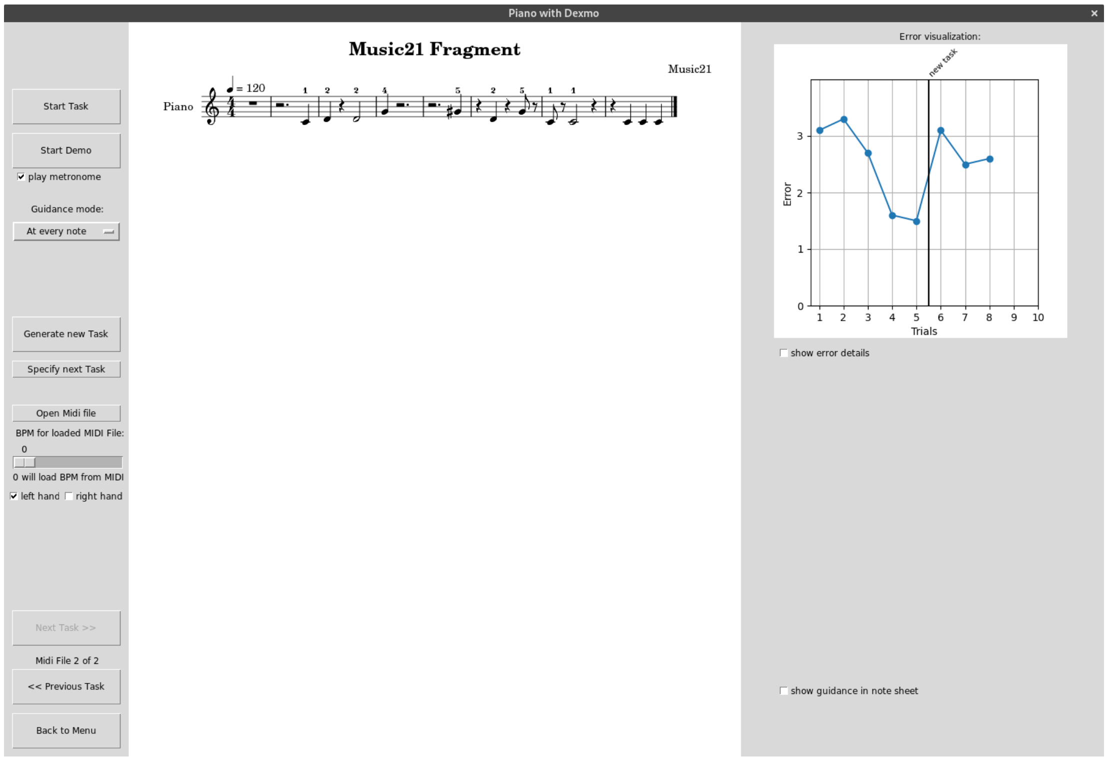
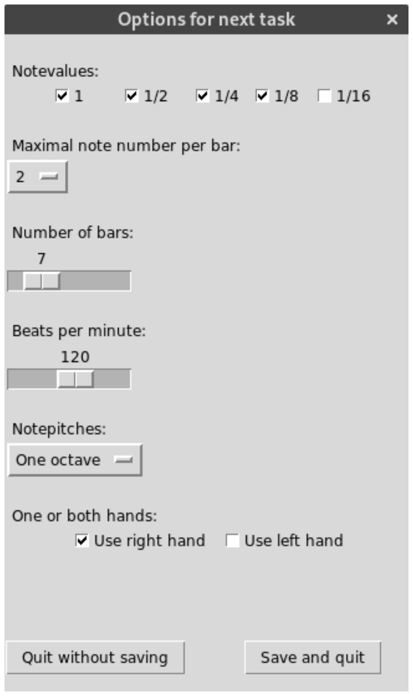


Figure 3: Main window

then. The Dexmo guidance mode can be set with the drop-down menu, as long as a Dexmo device is connected. To just listen to a task, e.g. before practicing the first time, the *Start Demo* button will play back the whole piece once. To practice it, the top button starts the task without playing back the notes. While the task is running, the notes pressed on the keyboard are taken into account. The errors are computed and stored afterwards. See section 4.6 for details about the error computation and visualization.

To toggle the metronome, the respective checkbox can be used which works for both the demo and the practicing mode.



Options for next task

Notevalues:
✓ 1 ✓ 1/2 ✓ 1/4 ✓ 1/8 ☐ 1/16

Maximal note number per bar:
2

Number of bars:
7

Beats per minute:
120

Notepitches:
One octave

One or both hands:
✓ Use right hand ☐ Use left hand

Figure 4: Options window

In the options window, shown by figure 4, the task generation can be adjusted in various ways. The note values that are enabled at the top will appear in the sheet music. Below that, the user can set the maximum number of notes that can appear per bar. The next options are the total number of bars in the piece and the tempo in beats per minute (BPM). The possible note pitches are to be selected in the drop-down menu, which can also depend on the choice of active hands below. The checkboxes just enable or disable the respective hand. In accordance, there will only be tracks for enabled hands. Further explanation about the generation options can be found in the following section.

4.2 MIDI Generator

The notes forming the user's practice task are generated randomly using a standard uniform distribution. The output score depends on multiple generation options with different effects: The note values (duration), the amounts of notes per bar and the note pitches (these contain values that *can* occur) as well as the number of bars in total are features that directly influence the note generation. The other options have more general roles like tempo (beats per minute) and usage of the left, the right or both hands in the given score.

Most of these options mainly influence the task's difficulty, and a certain option setting used for multiple generations leads to comparable practice tasks. This comparability enables increasing or decreasing the complexity in relation to the user's progress, which could also be done automatically by having the program adjust the settings depending on the user's performance.

In the initial program in the scope of this project, the simple time signature $\frac{4}{4}$ is used, as it is one of the most common ones used in sheet music.

Depending on the selected task configuration (left hand only, right hand only or both hands), the generated MIDI files contain up to 5 tracks. For each hand, one track contains the piano notes and another one the Dexmo guidance messages (also MIDI notes). The last track is for the metronome, which matches the given tempo and time signature. Currently, there is an empty bar at the beginning for only the metronome. So the notes do not start immediately and the user can get ready and tune into the beat.

The random note generation works in two principal steps: Initially, the times at which notes will be added to the score are randomly selected. Currently the possible times in each bar are set to quarters, hence a bar can hold up to 4 notes depending on their duration: For example, one whole note, two half notes or four quarter notes would fit in, but even though 8 eighth notes would add up to a full bar, there are only 4 quarter "slots". In the second step, a note is generated for each chosen time. At first, potentially infeasible durations from the predefined range are removed, for example if the note would overlap with the next one. If no duration is possible, the note is skipped. The last steps are the random selection of value and pitch, both within their respective ranges.

At the very end, three extra notes are added per hand (duplicating the last generated note) as a workaround for the automatic fingering algorithm, which is explained in the next section.

Information about storing the generated MIDI files is provided in section 4.7.

4.3 Finding Fingerings

The open-source library *PianoPlayer* from marcomusy under the MIT License is used for generating finger numbers for both generated and imported MIDI files as well as a MusicXML file used for visualizing the sheet music with finger numbers [15].

The algorithm is a search algorithm with the goal of minimizing the effort of the hands by minimizing the finger's speed needed to play the task and avoiding unnecessary movements. The search space includes all feasible finger combinations. An example for a unlikely and therefore as unfeasible excluded fingering combination is the 3rd finger crossing the 4th.

As mentioned in the previous section (section 4.2), there were some problems with generating the finger numbers in the form that no fingering was found for the last three notes of each hand. This is probably due to the algorithm not having enough information for the last notes to generate the finger information. As a workaround, the last note for each hand was added three times at the end of the task so that all actual notes have fingering information. This however only works for generated tasks and not for imported MIDI files.

Another problem is that for too few notes, the algorithm doesn't deliver any fingering information because there is too little information for the search. In this case, the task is regenerated with a reduced set of notes from C to G where those 5 notes are linearly mapped to the 5 fingers. Again this only works for generated tasks and not for imported files. If this C–G-guidance is used, the fingering information can not be shown in the GUI because the necessary XML file is generated through the library and is not available for this "hard-coded" guidance.

4.4 Dexmo Client Interface

One part of the project was to enrich the existing Dexmo Client in a way that on the one hand allows to control Dexmo based on music, meaning that Dexmo should provide haptic feedback for when a note should be played, and on the other hand sends its position to the main application for further processing. There were mainly two options for implementing the communication between the Dexmo Client and the rest of the program: ROS and MIDI. ROS had the disadvantages that this interface would have been the only part of the system using ROS, so the overhead of using ROS would be required for relatively little gain and people outside of the robotics community are usually not familiar with ROS. So the communication with the Dexmo Client is implemented through MIDI messages, which is platform-agnostic and consistent with the communication in the rest of the project as well as other systems in music production. For sending out Dexmo positions and receiving commands, different MIDI message types were used which will be explained in the following sections.

4.4.1 MIDI message commands for Dexmo

The commands sent to Dexmo need to contain all the aspects that are necessary to provide the required haptic feedback. First, the type of feedback that is required needs to be communicated. In the initial project phase, two types of feedback were considered and later implemented in the Dexmo Client: impulse and guidance. In impulse mode, the finger is moved sightly either in the direction of the keyboard when a note should be played or away from the keyboard when one should stop playing a note. The guidance mode guides the finger onto the key and holds it there until it is stopped or guides the finger away from the keyboard and holds it there until stopped. Next, the finger and hand, for which the haptic feedback should be provided, needs to be contained in the command. Lastly, since the haptic feedback for a finger should start before or at the time of a note event from the currently played MIDI file and stop at some point depending on the mode, the command needs to contain information about the direction of the feedback. After considering these aspects, note-on and note-off events where chosen for the Dexmo commands because the controlling of Dexmo is so closely related to when notes are played. It also make sense conceptually because note events are the only events which produce a physical effect (playing a sound) on a device and the Dexmo commands should have the effect of producing haptic feedback.

The aspects mentioned above are packaged in the MIDI note events as follows: One note event correlates to one note being played by one finger. It therefore makes sense that one event is used per finger. To distinguish between fingers, the key/note number of the message is used. Each finger is assigned to an octave according to that finger's number in piano sheet music (1 for thumb to 5 for pinkie). Now the different notes in the octave can encode different feedback modes. However, we have two hands that need to be considered. There are theoretically 10 Octaves from Octave 0 to Octave 9, but the ninth octave is not complete. Using these octaves to encode both hands would in theory be possible, but using the associated finger numbers as octaves for each finger would not work anymore and the number of possible modes would be reduced since the last octave is not complete. There are however two Dexmo devices, one for each hand, and they controlled through two separate commands. Hence it makes sense to consider them as two separate devices and therefore use different MIDI channels to differentiate between them. By using the channel and the note to encode hand, finger and mode, there is the message's velocity slot left to pass further information about the mode, if needed.

In the project's initial phase, two modes where considered: impulse and guidance. The guidance mode should basically move the learner's finger onto the note, hold it there while the note is being played and remove it afterwards. In impulse mode, the finger which should play the note is only moved part of the way, telling the learner to use this finger to play the note. The impulse mode stops automatically once the finger complies with the impulse, therefore mapping this mode to note-on and note-off events is straightforward: The note-on event gives an impulse to press a note and a note-off event gives an impulse to stop playing a note. The guidance mode however is a little more complex since the guidance is held for a certain time, dependent on the note length for playing a note and the finger's speed for releasing a note, and should also be controlled through a MIDI message. So guidance for playing a note as well as guidance for stopping a note has a start and a finish, each requiring a note-on and note-off event. Therefore the guidance mode is split into two modes, guidance inwards and guidance outwards. This leads to three modes with the following key/note mapping: (i) **Note A** for actuate stands for **impulse mode**, note-on being impulse inwards and note-off impulse outwards. The velocity parameter in this message is used to set the threshold (approximated in degrees) that is used to register the compliance of finger and impulse. (ii) **Note F** for flex stands for **guidance mode inwards**. Note-on meaning the guidance should start and note-off stopping the guidance. Here the velocity parameter controls the speed of the guidance. (iii) **Note E** for extend stands for **guidance mode outwards** and is configured the same as the guidance inwards mode in anything but direction. The MIDI message structure is summarized in table 1.

Status	Data Bytes	Description
1000nnnn/ 1001nnnn	0kkkkkkk 0vvvvvvv	1000: Note Off event. Message sent when a note is released/ended. 1001: Note On event. Message sent when a note is pressed/started. nnnn = 0-15 MIDI Channel Number 1-16 Channel determines which hand is used. kkkkkkk = the key (note) number: octave determines finger, note in octave determines mode. vvvvvvv = the velocity, here a parameter depending on the mode.

Table 1: MIDI message structure for note events sent from the main application to the Dexmo Client. Table modified from Summary of MIDI messages by the MMA where the basic structure of the major MIDI message types can be found [16].

4.4.2 Dexmo position as a MIDI message

In order to be able to use information from Dexmo in the main application, for example to validate that the correct finger was used to play a note, the information about Dexmo's position needs to be communicated. Dexmo has 11 measured angles: Each of the five fingers is measured in bend and split direction, where the bend angle measures how far the finger is bent towards the palm of the hand and the split angle perpendicular to the bend angle in the direction of the other fingers of the hand. The 11th angle is the thumb rotation angle. There were two message types considered for this: control change messages and system-exclusive messages. System-exclusive messages are custom messages intended for device specific data. It would be easy to fit the intended data into this message type, however this would be completely device-specific and not in the sense of the generic approach which was one of the reasons for choosing MIDI messages in the first place. Hence, control change events where chosen. These require sending one message per angle, where the value parameter encodes the angle (approximately) in degrees, i.e. Integers from 0 to 127. If higher precision is necessary, it would be possible to send a second message for each angle containing for example decimal point encoding. Since sending the full Dexmo position whenever there are little changes would be a lot of information that would need to be processed by the main application, messages are only sent when an angle changes by more than a predetermined threshold and only for the angle that changed.

When using control change messages, there are some control change numbers which are usually used for general purpose control changes and therefore should be avoided for the Dexmo communication. Free consecutive control change number ranges are 20–30 and 52–62. For the same reasons as in the previous section (section 4.4.1), different channels are used for the two hands. The following mapping is used between angles and control change numbers:

- Thumb: rotate = 20; split = 21; bend= 22
- Index: split = 23; bend = 24
- Middle: split = 25; bend = 26
- Ring: split = 27; bend = 28
- Pinky: split = 29; bend = 30

A summary of the MIDI message structure can be found in table 2.

Status	Data Bytes	Description
1011nnnn	0ccccccc 0vvvvvvv	Control Change event. Message sent when a controller value changes. nnnn = 0-15 MIDI Channel Number 1-16 determines which hand the data is from. ccccccc = control change numbers determine which angle this message contains. vvvvvvv = the value parameter encodes the angle (approximately) in degrees.

Table 2: MIDI message structure for control change events sent from the Dexmo Client. Table modified from Summary of MIDI messages by the MMA [16].

4.4.3 Implementation of the Dexmo Client

In each loop iteration of the Dexmo Client main loop, Dexmo's position (i.e. the values of the 11 measured angles of Dexmo) is updated if Dexmo sent updated angle values to the client. The haptic feedback currently provided by Dexmo is then checked based on the updated Dexmo position. If a new guidance or impulse command was received, previous commands for the associated finger are stopped, the current angle of the associated finger is saved and the force point for the associated finger is computed either as the maximum of the range for impulses or based on the set guidance speed for guidances. For ongoing impulses, the system checks whether that impulse should stop because the associated finger moved by at least the amount that was set for the threshold in the initial command. If that is the case, the corresponding force is released, otherwise the force remains unchanged. The force points for ongoing guidances are updated based on the guidance speed that was set in the initial command, except if the guidance target was reached, in this case the force will remain constant on the guidance target. A guidance is stopped and the corresponding force released if a MIDI message to stop the guidance was received. Next, the current Dexmo position is checked against previously sent values. Sufficient changes in position are sent as a MIDI message, whose structure is explained in section 4.4.2. The MIDI messages sent to Dexmo are received in a callback outside of the main loop. During processing of these messages, variables are set which are considered when the Dexmo commands are computed in the main loop.

4.5 Communication between the system components

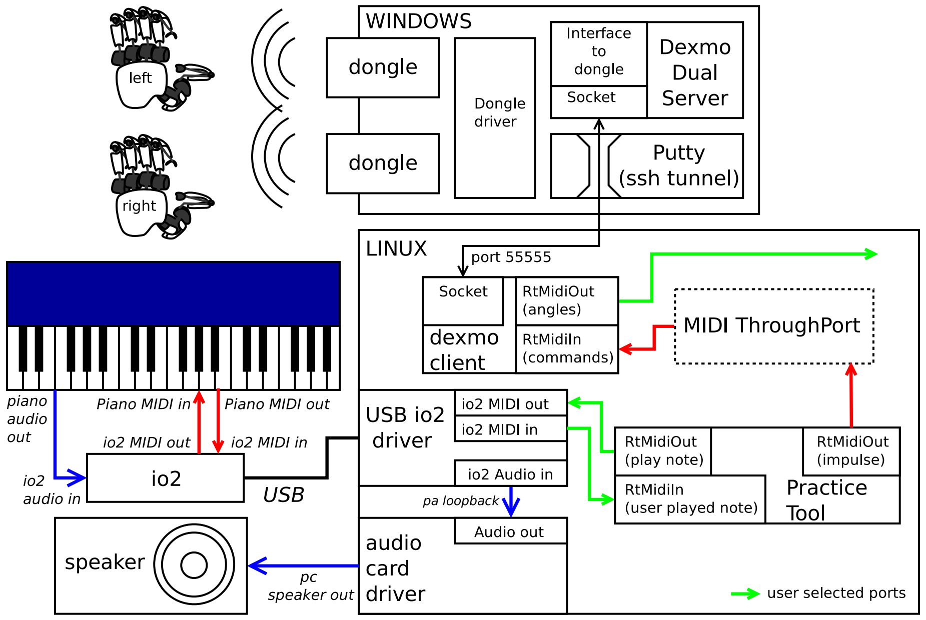


Figure 5: Diagram visualizing the system's internal and external communication through MIDI ports for the full lab setup including Dexmo and a MIDI-to-USB interface to connect the Piano. Image slightly modified from a diagram supplied by Guillaume Walck.

As mentioned before in section section 4.4, not only the system's communication with the outside but also the internal communication is realized through MIDI messages, which is visualized in figure 5. The practice tool receives user input from the connected keyboard the user is practicing with on the RtMIDIIn (*user played note*) port. The system sends MIDI messages to two MIDI ports: The RtMIDIOut (play note) port of the practice tool sends messages which can be interpreted by e.g. a synthesizer to play the current practise task, while the RtMIDIOut (impulse) port of the practice tool sends messages through the MIDI Through Port to the Dexmo Client which controls Dexmo to provide appropriate feedback. The port on which the keyboard is connected as well as the one to which the MIDI notes of the practice task are sent for synthesizing have to be set in the initial screen of the practice tool, shown in figure 2. If the tool is used in combination with the Dexmo Client, the main application's Dexmo output port, which sends the impulses to the Dexmo Client, has to be set to the MIDI Through Port in the initial screen of the practice tool, also shown in figure 2. If the tool is used with the alternative LEGO-Dexmo, the port selection is not restricted at this point. For future extension, the Dexmo Client sends out its position to a user selected port which is so far not considered by the practice tool.

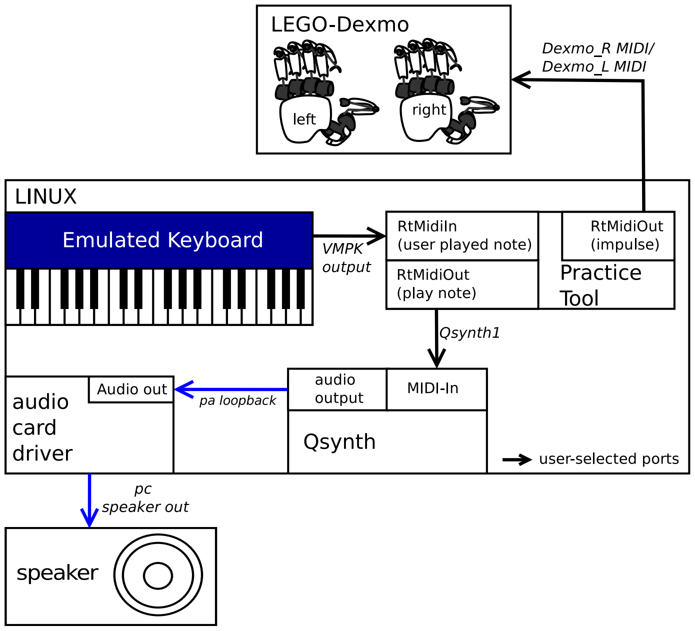


Figure 6: Diagram visualizing the system's internal and external communication through MIDI ports for a minimal setup with LEGO-Dexmo (optional). Image slightly modified from a diagram supplied by Guillaume Walck.

A (minimal) example configuration, as shown in figure 6, using LEGO-Dexmo (right hand) with a MIDI keyboard could be as follows: At first, the Dexmo output port has to be chosen, i.e. "Dexmo_R MIDI". The sound output port is operated by the synthesizer *Qsynth* (running on the computer) and is named "Qsynth1". This is used for the notes being played by the program itself. The notes played on the keyboard are received on the piano input port, whose name depends on the keyboard or interface that is connected. We use an *Alesis Q25* MIDI keyboard which sets up its own port named "Q25". For testing, a virtual MIDI keyboard can also be used instead, for example *VMPK*. To play back the keyboard notes while playing, it is necessary to hook up the keyboard port into the synthesizer, e.g. with *aconnect*.

When haptic guidance via Dexmo is enabled, the corresponding MIDI messages are logged to */tmp/DexmoPiano/* for debugging purposes.

4.6 Error computation

To evaluate the user's performance while and after playing, an error measure is necessary. For the sake of testing and simplicity, a rather primitive one is initially used and visualized: Within a task, the overall time (in milliseconds) of any key being pressed is added up. The sum is then compared to the task's target sum, i.e. the overall time when some note should be played. The resulting error value is simply their absolute difference in milliseconds.

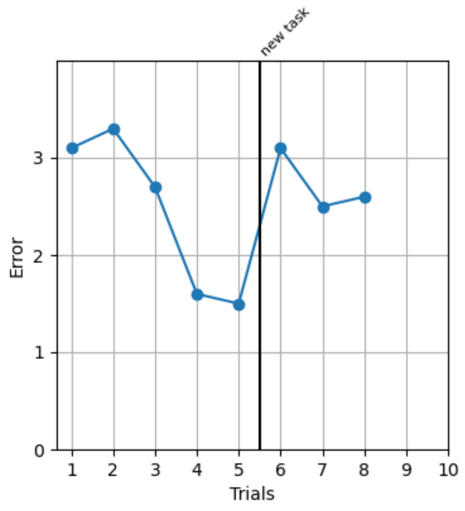


Figure 7: Exemplary error visualization. The first 5 trials show the performance progress of some task. Afterwards, a new task was generated (using the same options) and practiced for three times.

Currently, the user's error is displayed as a graph, showing the error value for each trial of the same task. When a new task is generated with the same options, the graph continues after a thick bar to highlight the change. Figure 7 shows a visualization example for a few trials on two tasks with the same generation options. When different options are used for a new task, the error graph will be cleared.

4.7 Practise data storage

For each generated task, multiple files are created and needed apart from the task's MIDI file itself. These files are contained in a ”temporary” subdirectory within the project folder, which is cleared regularly since most of those files are not needed after the task is finished. MIDI files that were used for practicing are copied to another (non-temporary) subdirectory next to the temporary one and can therefore be reused. In such cases, an XML file is created which contains task-specific information, mainly generation settings, the list of notes to be played and a timestamp (also used as a unique identifier).

Every time the user practices a certain task, information about that trial is written to the respective XML file. This includes a timestamp, the notes that were actually played, the guidance mode and the computed error value(s).

5 Results

The goal of the project, a working prototype of the piano learning program with haptic feedback from Dexmo, was achieved. It is currently possible to generate pieces of music for both or one of the two hands, which the user can practice. One can choose how many bars, how many notes at most in each bar, which pitches and which note lengths the music piece should contain. There is also a possibility to load saved MIDI files.

To improve the task, one can practice it and the evaluated error is displayed for each trial. However, it is also possible to have the piece played as a demo with haptic feedback to get to know it. The metronome, which gives the user the rhythm and thus the timing, runs during both runs, but can also be switched off.

The finger numbers with which the note should be played are also displayed in the note sheet and the respective finger is haptically addressed by Dexmo. The display of finger numbers is not possible in cases where less than seven notes are generated for one hand, but guidance can still be provided in those cases.

On the error diagram, the user can see his learning progress of a repeatedly practiced music piece. If the difficulty of the next piece remains the same (same number of bars, number of notes maximum in each bar, same pitches and note lengths), the error values are still displayed in the same diagram. If the difficulty of the task changes, the diagram is reset.

The errors, user settings, played notes for each attempt and MIDI files are also stored, allowing the user to go back to one of the last practiced tasks and play it again.

6 Discussion and further development

There are some areas of the project that can be improved. The probably most important extension is a differentiated error evaluation of the played task. The error calculated so far serves mostly as a prototype, since only the lengths of the notes are compared, independent of pitch, timing and other factors. Therefore, the calculation of differentiated error types such as timing, duration, amount of notes, note pitch, loudness, velocity and others could be incorporated in the future. Furthermore the position of Dexmo could be useful for determining errors in the fingering of the pieces. So far the Dexmo Client provides its position, however this is not used in the main application.

Once different error types are created, the generation of the next task could be adapted to them. For example the tempo of the next task could be reduced if the timing error is high etc. The guidance could also be adapted to the error and a machine learning component could be used to select the best possible guidance and task for a specific user.

The guidance could also be revised further. Instead of the current inwards guidance when a note starts and the outwards impulse when a note ends, an inwards impulse could also be added that stops as soon as the user moves his finger in order to avoid excessive guidance. One could also start the guidance of the movement towards the note just before the actual note, so that the note is played for the entire duration without delay. Also, the guidance could raise the finger slightly before lowering it to the piano to make the movement more natural and announce it better. Another variable which could be varied is the strength of the guidance. However, these adjustments to the guidance would need to be evaluated regarding their effectiveness because Adams et al. were able to show that feedback as hint is better for learning than real finger movement, because fine finger movements etc. are the most difficult to transfer with a robot [6]. If many different modes where implemented in the tool, it could also be used to evaluate which type of feedback is the most effective and useful for the user.

The selection of pitches could also be adjusted, so that only notes suitable for certain musical styles (e.g. blues) are created in a generated piece of music. This could be useful to get a reasonable sound despite randomly generated notes.

Another development could be to show the user in the GUI at which note they are in the actual task. Visual feedback in the GUI could also supplement the haptic feedback on Dexmo, as our literature research has shown that this combination is far more useful than haptic feedback alone.

An additional training mode could also be introduced to wait until the correct note is played before continuing the exercise. This could help beginners to get used to a task and ease the initial difficulty.

Even though all data (the MIDI file, the user settings, the played notes and the error) is stored in an XML file, this file is not read when the task is repeated by the user. This feature could be added to show the user his previous error and to give him an incentive for improvement.

It could also be considered to integrate the project into an existing program in which a live display of the task or of the current note may already exist, for example in open source programs like *PianoBooster* [17], which however was written in C++ and therefore is currently not compatible.

References

[1] dextarobotics. `https://origin.dextarobotics.com/en-us/`. [Online; accessed September-2020].

[2] `https://www.youtube.com/watch?v=Sif7cY8qwjM`. [Online; accessed September-2020].

[3] flowkey GmbH. `https://www.flowkey.com/en`. [Online; accessed September-2020].

[4] OnlinePianist. `https://www.onlinepianist.com/`. [Online; accessed September-2020].

[5] K. Ann Renninger and Alexandra List. *Scaffolding for Learning*, pages 2922–2926. Springer US, Boston, MA, 2012.

[6] Richard J Adams, Daniel Klowden, and Blake Hannaford. Virtual training for a manual assembly task. *Haptics-e, The electronic journal of haptics research*, 2001.

[7] Gabriele Wulf, Charles H Shea, and Sabine Matschiner. Frequent feedback enhances complex motor skill learning. *Journal of motor behavior*, 30(2):180–192, 1998.

[8] Amaya Becvar Weddle and James D Hollan. Professional perception and expert action: Scaffolding embodied practices in professional education. *Mind, Culture, and Activity*, 17(2):119–148, 2010.

[9] Dan Morris, Hong Tan, Federico Barbagli, Timothy Chang, and Kenneth Salisbury. Haptic feedback enhances force skill learning. In *Second Joint EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (WHC'07)*, pages 21–26. IEEE, 2007.

[10] David Feygin, Madeleine Keehner, and R Tendick. Haptic guidance: Experimental evaluation of a haptic training method for a perceptual motor skill. In *Proceedings 10th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. HAPTICS 2002*, pages 40–47. IEEE, 2002.

[11] Marie-Hélène Milot, Laura Marchal-Crespo, Christopher S Green, Steven C Cramer, and David J Reinkensmeyer. Comparison of error-amplification and haptic-guidance training techniques for learning of a timing-based motor task by healthy individuals. *Experimental brain research*, 201(2):119–131, 2010.

[12] Graham Grindlay. Haptic guidance benefits musical motor learning. In *2008 Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 397–404. IEEE, 2008.

[13] Laura Marchal Crespo and David J Reinkensmeyer. Haptic guidance can enhance motor learning of a steering task. *Journal of motor behavior*, 40(6):545–557, 2008.

[14] `http://lilypond.org/index.html`. [Online; accessed August-2020].

[15] Marco Musy. `https://github.com/marcomusy/pianoplayer`. [Online; accessed August-2020].

[16] `https://www.midi.org/specifications-old/item/table-1-summary-of-midi-message`. [Online; accessed September-2020].

[17] `https://github.com/captnfab/PianoBooster`. [Online; accessed September-2020].