



Figure 1. Superimposed representation of .PNG layers.

mapping guarantees a deterministic and faithful graphic to traditional music notation translation.

At this point, the data is finally ready to be converted into a string using the Bach's library tree format. This allows the data from Xnk to be opened in MaxMSP and later through any mainstream music editor.

Link: [Video of Xnk demonstration](#)

2.2 Detailed Explanation

Xnk uses the following C++ libraries: OpenCV2, LibPNG.

2.2.1 Preparing the .PNG Images

Xnk interprets individual .PNG images as independent voices. This gives the composer the power to think about each image as either: individual instruments or an ensemble of instruments. Because Xnk preserves the independence of layers, each image will correspond to an individual voice in the output. If the user inputs N number of images, the output will have N number of voices.

The .PNG images can be created using tools such as Adobe Illustrator, Concepts, Inkscape, etc. When using these tools, the user needs to export each layer individually and store it as an isolated layer. This will ensure that every layer is of the same size. See Fig. 1.

2.2.2 User-Defined Parameters

As Fig. 2 shows, Xnk gives its users control over the following parameters:

- Maximum Output (highest) Pitch.
- Minimum Output (lowest) Pitch.
- MIDI Velocity.
- Target Length.

The parameters Maximum and Minimum Pitch define the pitch range of the output. These parameters expect the user to input values in midicents. If not defined, they default to a range from 10800 to 2100.

The MIDI Velocity parameter defines a uniform MIDI velocity for the output. It takes integer values ranging from 0 to 127.

The Target Length parameter is used to calculate the stretch factor that will be used for scaling the events' onsets and duration. This parameter expects the user to input the desire length of the output in milliseconds.

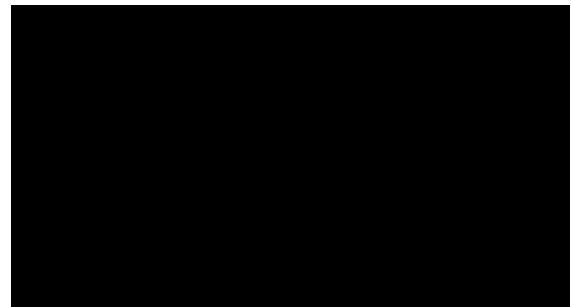


Figure 2. Xnk' Parameter Input.

2.2.3 Pre-Processing Preparation

Before any processing takes place, Xnk reads and stores the User-Defined Parameter values as well as the filepath list that contains the location of the .PNG images.

2.2.4 Image Processing

1. An empty vector is created that will be used as a container that will store the binarized images.
2. Xnk iterates through each string element in the filepath vector created on the previous step. For each filepath in the filepath vector:
 - Loads the image found at the iterated filepath to memory.
 - Binarization of the image. This is done so that Xnk only deals with 1s and 0s. This makes further processing simpler and more efficient.
 - Adds the binarized image data to the binarized image container vector created before the start of the iteration loop.
 - Cleans up memory preparing the next iteration.
3. Creates an empty vector that will store vectors containing the Voice Event Data. The nested vector structure is used to preserve the independence of each inputted .PNG image.
4. Once all the images are binarized and stored to memory, Xnk starts the event identification process by iterating through every binarized image stored at the binarized image container vector. For each binarized image in the binarized image container vector:
 - Initializes an empty vector that will serve as a local event container for the event data belonging the current binarized image iteration.



Figure 3. Output in MaxMSP.

While programs such as Finale or Sibelius are powerful tools, they are focused on editing music notation, not composing music. To do the latter, there are alternatives such as the Bach Library [1] for MaxMSP [2], OpenMusic [3], MaxScore [4], SPEAR [5], among others. Additionally, a similar concept was adopted by Hyperscore [6].

While SPEAR also deals with graphics to create sonorities, it is centered on spectral analysis of pre-existing sound files. In Hyperscore's case, it gave its users visual representation of music parameters to create music. Arguably, Hyperscore interpreted graphical notation. In contrast, Xnk parses graphical notation.

2.4 Current Limitations

At the current stage of development, Xnk has the following limitations:

- Only accepts .PNG images as inputs.
- MIDI Velocity is arbitrarily decided, not deducted
- The dependance on external graphic editors to create scores and parts does not allow Xnk to natively do graphic edits to the .PNG images. Furthermore, Xnk's relies on MaxMSP for further processing, playback, and MusicXML/MIDI export methods.
- The algorithm that extracts event data from binarized images needs further improvement.
- Not able to recognize glissandos as individual events. They are interpreted as sequences of events.
- Extreme sensitivity creates a substantial amount of redundant data.

2.5 Future Development

Currently, there are four main priorities for future development. In order of importance, they are:

- Create a native graphic notation editor.
- Improve the event extraction algorithm.
- Create a native playback engine.
- Create a native music notation editor.

2.5.1 Native Graphic Notation Editor

Developing a Native Graphic Notation Editor will be a huge step forward in Xnk's development. This will enhance the Xnk's workflow by eliminating the need to constantly alternate between Xnk and the user's graphic editor

of his choice. Rather, the user will be able to make edits directly in the Xnk environment.

Developing this feature will prepare the path for in-depth user defined MIDI velocity handling as well as Computer Assisted composition AI algorithms.

In its current state, Xnk has a graphic editor that is at an early stage of development. The current challenge with the editor environment within Xnk is that it needs to be able to expand horizontally towards positive x infinitely while, at the same time, supporting infinite zoom. This is currently being done by treating graphics as vectors in a similar way in which Adobe's Illustrator [7] or TopHatch's Concepts [8] do it.

Once the previously discussed features of the editor environment are developed, specific graphic editing in the context of Xnk's goals will be created.

2.5.2 Native Playback Engine

Playback engines are indispensable for music software. As mentioned before, Xnk currently relies on MaxMSP to handle playback. Developing a Native Playback Engine will improve Xnk's functionality, practicality and workflow.

This playback engine must be powerful, scalable, polyphonic, and microtonal friendly in nature. One of the current questions is whether to use MIDI.

The MIDI protocol was not designed to natively work with microtones. While there are ways around that, they can be impractical in their application and decrease the overall power of the playback engine. The most appealing solution would be to create a native Xnk's playback protocol that would allow for polyphonic microtonality without relying on workarounds. In addition to being able to handle microtonality natively, this theoretical new protocol must be able to be translated into commonplace formats and protocols such as MIDI or MusicXml.

2.5.3 Native Music Notation Editor

The creation of a Native Music Notation Editor will make Xnk an powerful and versatile music composition program.

One of the most important and challenging aspects of developing this music editor will be implementing the logic behind rhythmic quantizing tools.

Contrasting Xnk editor with the ones present in mainstream music editors, Xnk will be composition focused

Figure 4. Hands2MIDIChannels raw input video.

rather than engraving focused. The goal of Xnk is to provide its users with a flexible and dynamic composition environment.

2.6 Closing Thoughts

Xnk is not intended to replace industry-standard music editors. In the case of Finale [9] and Sibelius [10], they are both powerful computer engraving programs. However, they lack the degree of flexibility and functionality needed for a pure composition software.

Ideally, Xnk will become an intermediate step between music sketching and a finalized score.

3. HANDS2MIDICHANNELS

3.1 Overview

Hands2MIDIChannels is a python script capable of routing MIDI events to specific channels based on the hand that triggered the event. This is done by cross-referencing the original MIDI file with a video that captured the performer's hand movements over the keyboard.

It takes a MIDI file and a corresponding video as inputs. After the video file passes through a number of pre-processing steps, it is manually synchronized with the video recording. This is done so that Hands2MIDIChannels knows the specific moment in which a MIDI event is triggered in the corresponding video.

At this point, further pre-processing steps take place in preparation for the cross-reference stage. During the cross-reference, Hands2MIDIChannels iterates through the events on the MIDI file as shown in the video and calculates which hand most likely triggered the iterated event.

After Hands2MIDIChannels assigns a hand to every relevant event in the MIDI file, it creates a new MIDI file where the events are routed to specific channels based on the hand that triggered the event.

Link: [Video of Hands2MIDIChannels demonstration](#)

3.2 Detailed Explanation

Hands2MIDIChannels uses the following Python libraries: OpenCV2, Numpy, Mediapipe, MIDO, Pillow, and TQDM.

3.2.1 Initial Video Preparation

The goal of the Initial Video Preparation stage is to make the video recording as computer-vision analysis-friendly as possible. At the end of this stage, Hands2MIDIChannels obtains a video where the area of interest shows the keyboard as a perfect rectangle. This makes further processing straightforward.

1. Store Input Paths
 - Hands2MIDIChannels stores the paths for the MIDI and the corresponding video file.
2. Initial Video Crop
 - User defines the overall area of interest that clearly shows the performer's hands as well as the keyboard using an interactive interface. See Fig. 4 and 5.
3. Set Keyboard Top Bound
 - User defines the keyboard top bound using an interactive interface.

Figure 5. Hands2MIDIChannels input cropped video focusing on the area of interest.

Figure 6. Hands2MIDIChannels input video after pre-processing and transformation.

4. Set Keyboard Bottom Bound
 - User defines the keyboard bottom bound using an interactive interface.
5. Keyboard Bound Calculation
 - Hands2MIDIChannels uses the previously gathered information to calculate the side bounds of the keyboard.
6. Hand Bounds Calculation
 - Hands2MIDIChannels calculates the hand bounds in respect to the keyboard bounds.
7. Calculate 2D Transformation Matrix
 - To account for image distortion commonly created by lenses, Hands2MIDIChannels calculates a 2-dimensional transformation matrix that will make the keyboard top and bottom bounds be as parallel as possible over the x-axis. See Fig. 6.
8. Set Y Levels for Black and White Keys
 - User defines the bottom bounds for both: the black and white keys. Note that thanks to the previous 2D Transformational Matrix, this bound is parallel to the x-axis.

3.2.2 Video-MIDI Synchronization

Synchronizing video and MIDI as well as abstracting the individual onsets for each MIDI event is important because this will help Hands2MIDIChannels locate the exact place in time on the video where an inquired MIDI event takes place.

1. Set Sync Onsets
 - User inputs the first and the last onset of events in the MIDI file as they are shown in the video.

2. Individual Onset Calculation

- By using the first and last onset, Hands2MIDIChannels calculates the individual place in time where a MIDI event is shown at the video.

3.2.3 Final Preparations

During the Final Preparations Stage, Hands2MIDIChannels calculates the contours corresponding to every piano key. Then, the user is presented with an interactive interface that allows finetuning the contours. These contours will be used in the Cross-referencing Stage that follows.

1. Hands2MIDIChannels creates an approximation of the contours corresponding to every individual white and black keys.
2. The user adjusts the contours so that they accurately represent the keyboard as shown in the video.
 - Because the video is transformed by a 2-Dimensional Matrix, the keys in the keyboard tend to have slightly different widths. Modifying the contours' placement over the x-axis accounts and corrects for this.
3. The final keyboard' keys contours are created. MIDI note information is attached.

3.2.4 Cross-Referencing

During the cross-referencing stage, Hands2MIDIChannels iterates through the MIDI events and their corresponding onsets in the video. It calculates which hand most likely triggered the iterated event. See Fig. 7. For each MIDI Event:

- Get MIDI note number.

Figure 7. Hands2MIDIChannels recognizing hand data from a still frame. Inquired MIDI event is highlighted in green.

- Get video onset.
- Get keyboard key contour corresponding to the inquired MIDI note number.
- Load the inquired video onset as an image.
- Calculate which hand most likely triggered the event.
- Store result.

Hands2MIDIChannels uses logical gates to determine if a note was played with the right or left hand. Currently there are 4 stages in determining the guess. If the algorithm arrives to a guess at any stage, it skips the remaining stages.

The stages are:

- Check if no hands are detected in the video. In this case, Hands2MIDIChannels routes the event to an excluded MIDI channel. This stage is often triggered by a low frame rate video input or by Mediapipe failing to recognize hand information.
- Check if there is only one hand detected in the video. In this case, Hands2MIDIChannels assigns the detected hand as the hand that triggered the inquired MIDI event.
- Check if there is a hand located inside the inquired key contour area (that was calculated in the Final Preparations stage of the program). If this case is True, the guess will be the hand that is inside the inquired key area.
- Calculates the hand closest the inquired key area. This is the last resource. In this case, the guess is the hand closest to the inquired key area.

3.2.5 Create Output

In the Create Output stage, Hands2MIDIChannels creates a new MIDI file where the events are routed to specific channels based on the hand that triggered the event. This is done by duplicating the original MIDI file and updating the channel attribute for the MIDI events. See Fig. 8.

3.3 Justification

Creating a legible piano score from a MIDI recording is a laborious task. This is because most DAWs and music editors record MIDI events to a single channel by default. This in turn routes all events to a single music staff. To fix

this, there are two common solutions that industry standard software present: manual channel assignation or arbitrary choice of split point.

The manual channel assignation method involves the user changing the MIDI channel for each MIDI event manually. This can be a practical solution for short recordings but it can quickly become an enormous task when dealing with larger durations.

The second method, arbitrary choice of split point, involves the user selecting a note as the split point. The notes below the split point will be routed to an individual MIDI channel while the notes above the split point will be routed to another MIDI channel. For simple purposes, this works great. However, for contemporary music and improvisations, this method does not work since there isn't a constant split point most of the time.

Hands2MIDIChannels offers an alternate solution that strives to involve minimal user input and improve efficiency. At the same time, Hands2MIDIChannels opens the door for accessibility capabilities. Blind improvisers and composers will be particularly benefited by this program since it drastically decreases the need for external intervention.

3.4 Current Limitations

Hands2MIDIChannels aims to decrease user involvement, however, it still relies heavily on it. Additionally, processing speed has room for improvement. While it remains faster than any other method, processing times for long files (28000+ events) is around 2:30:00 (hr:mn:sc). Furthermore, Hands2MIDIChannels only works with MIDI keyboards. It does not support acoustic instruments.

Finally, the guess accuracy has room for improvement. Errors created by Mediapipe not recognizing hand landmark information or a slight synchronization error in the video create inaccurate guesses.

3.5 Future Developments

Future developments for Hands2MIDIChannels aim to achieve the following:

1. Improve the hand-guess algorithm.
2. Support acoustic pianos recordings.
3. Improve time efficiency.
4. Decrease user intervention as much as possible.

Figure 8. Before and after Hands2MIDIChannels processing comparison. MIDI channels are represented by colors.

To improve the hand-guess algorithm in the future, it would be useful to create a machine learning model from scratch that specializes in recognizing hand information from a frame. Another possible option would be to add additional steps in the video pre-process stage with the goal of making the visual material as compatible as possible with Mediapipe. Yet another possible path is to implement a function that will audit the guesses and calculate their accuracy. Based on this probability, the function will be able to correct the guesses if they need it.

Supporting acoustic pianos video recordings as a singular input for Hands2MIDIChannels is a challenge, yet it is a theoretically possible challenge. By abstracting pitch information from the recorded signal using Fast Fourier transformations, comparing it to the corresponding video frame, then having a machine learning model recognize the hand data in that frame, and finally, output an accurate MIDI file with events sorted in two channels, Hands2MIDIChannels would be able to support acoustic piano video recordings.

To improve time efficiency, the code needs to be reorganized, translated to C++, and designed to decrease user involvement as much as possible. Hands2MIDIChannels aims to be an efficient tool that is able to save its users' time by automating the hand assignment process as much as possible.

3.6 Closing Thoughts

Hands2MIDIChannels gives its users a time-efficient tool that allows them to improvise/compose freely at the keyboard with the certainty that they will have access to an organized MIDI file ready for time quantization. Because of its nature, Hands2MIDIChannels thrives as both: as a

standalone software, and as an external function capable of interacting with existing DAWs and music editors.

Finally, Hands2MIDIChannels is of particular help for blind composers and improvisers.

4. CONCLUSION

This paper introduces two novel software tools independently developed by the author, offering innovative solutions for optimizing creative workflows in music composition and improvisation.

Addressing specific challenges overlooked by current industry-standard software, Xnk presents a deterministic, reliable, and scalable framework for the translation of graphic elements into traditional music notation. Conversely, Hands 2MIDIChannels streamlines the hand assignment process for keyboard instrument recordings.

Beyond their technical advantages, both tools contribute to unparalleled accessibility solutions, catering to individuals with disabilities. Moreover, they play a pivotal role in democratizing the music creation process, extending its reach to a broader and more diverse population.

In summary, these tools not only advance the field of digital music notation and transcription but also have the potential to significantly impact accessibility and inclusivity within the realm of music creation, marking a noteworthy stride towards a more universally accessible and democratized musical landscape.

Acknowledgments

I would like to extend my heartfelt thanks to Stella Gitelman Willoughby, whose unwavering support and insightful advice have been instrumental throughout the course of

