Team Clara

OpenCV 2021 Competition

Computer Vision based Theremin Simulator and Practice Aid System

Final Report

Elisa Andrade, Jorge Chong

Jul 31st, 2021

Table of Contents

1 Problem

The Theremin is an electronic musical instrument invented by Lev Termen in 1920. It has two antennas, one for setting volume and one for tone (frequency). The tone of the apparatus is the result of a process of heterodynization of a signal at a reference frequency with a variable frequency oscillator, whose frequency depends on the distance from the hand to the tone antenna. One of the difficulties in learning the theremin is that there is no direct cue about where to locate the hand to produce the correct note. For example instruments like the piano or the guitar provide physical and visible feedback on the notes to be played. In the Theremin the only feedback provided is the tone produced. Not many people are gifted by the ability of perfect pitch and therefore playing the instruments requires a very steep learning curve. It would be nice to have a tool that considers human factors to accelerate learning and enjoyment.

In this regard, technologies like computer vision and augmented reality would make a big contribution. An ideal solution would have to understand the positions of the hands with respect to the antennas and at the same time provide visual cues about where to locate the hands in order to play the real note. To us, an ideal solution would provide visual feedback using augmented reality through devices like Apple glasses or Google glasses.

The solution will have to understand the player hands location and gestures, and here is where computer vision sensors like the OAK-D becomes a key part of the system. Algorithms for hand tracking and gesture recognition could be run on the device and interact with the system for providing feedback to the user. We have chosen these augmentations for the theremin because it will allow students to visualize notes spatially and correlate movements and notes, and in doing so working toward improving their musical intelligence.

We believe assisting technologies will be important part in musical education in the upcoming years. We envision use cases for children with special educational needs like children with autism spectrum syndrome. The use of a technology based on AI, that provides guidance and feedback will be key in their education since many of those children are sensitive to interaction with others including teachers who are not aware of their needs. Pictograph based educational systems are important to their development and inclusion. This vision is what compelled us to explore the problem more thoroughly and push the limits of what the OAK-D device can do.

2 Solution

Our proposed solution consists of 3 big parts. The first subsystem consists of a simulator of the theremin, based on computer vision. The second part is a calibration subsystem where we intent to adapt the dynamics of the virtual theremin to the player properties (hand sizes, body position, etc). Finally the practice subsystem will asses correctness and provide feedback to the player. A GUI that provides visual feedback is also an important element. A way to achieve this in an integrated product would consider devices with augmented reality capabilities.

We have made some compromises in order to present a proof of concept. As of July 31st we have completed the Theremin Simulator, we have researched the way the calibration should work. The recognition system would work on hand landmark detection. This part is in a rudimentary stage and requires the integration of a second camera in a single system. The whole system proved to be challenging and we are still working toward the final system. In the next sections we present the way our solution works and the challenges we have encountered through out the development of the project.

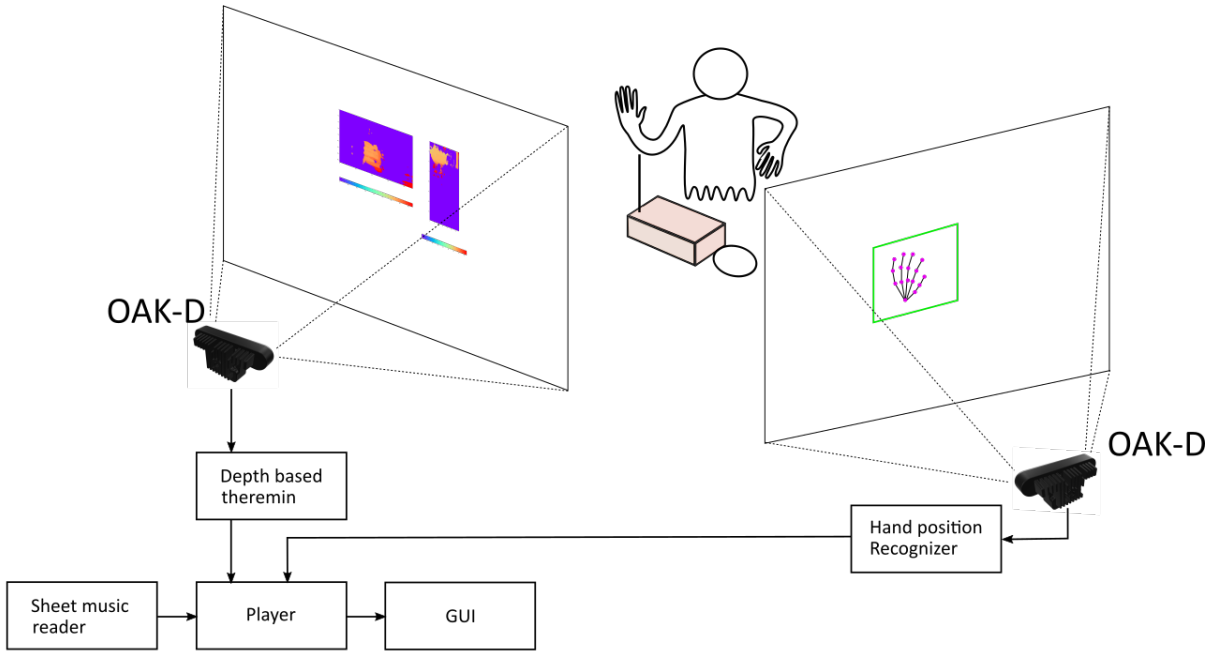

Figure 2.1: Block diagram of the ideal solution.

3 Subsystems

In this section we present details about our solution.

3.1 Theremin Simulator

The idea of building a simulator was twofold: first of all it would be amazing to use a computer vision system to have a virtual theremin and not depend on a real one, secondly we wanted to copy the working of the tone antenna in order to figure out a precise mapping between location and frequency, because that is the only way to provide visual feedback through a user interface (GUI or Augmented Reality).

3.1.1 Initial Ideas

At the beginning of the project we based our research in the implementation of the Mediapipe Hands model [Zhang et al., 2020] by geaxgx [geaxgx, 2021b], for the Openvino framework. This is a hand tracking system based on neural networks with excellent results. We found out this model is appropriate for tasks where you need to recognize gesture and/or follow hands at reasonable speeds.

The idea was to translate the geometry of the hand to a distance to an antenna (virtual or real) such that it could be mapped to a frequency value. The geometry and location of the hand were obtained by fetching hand landmarks detected by the neural network. Nevertheless, we have found three problems with this approach:

- It runs at around 11 to 13 frames per second, which introduce a lot of latency (90 ms). This is too high for a musical instrument. Our intention is to control frequency at high enough rate to provide a realistic and smooth effect.
- Landmarks detection prove to be too unstable. This can be solve by filtering, but in doing so we run the risk of introducing more processing latency. We also found that even if setting confidence thresholds at higher levels there were false detections, for example detecting the right hand not in the correct location but on different locations of the body which, if used in real time, produced frequency jumps and not the results we wanted.
- Finally, we wanted to locate the hand in space, and in order to do that we tried to use a spatial calculator node `SpatialLocationCalculator`, but this proved to be challenging because landmarks define small regions of interest and produced a too noisy registry of the z position in space.

3.1.2 Final Solution

We figured out that using just a depth map of the scene would be key to approach the problem. First it can run at higher frame rate (ideally 120 fps). It can also provide a more accurate geometric representation of the hand. The problem here is to figure out a mapping between geometry and distance to an antenna that be as faithful as a real theremin. This proved to be a more difficult problem than expected.

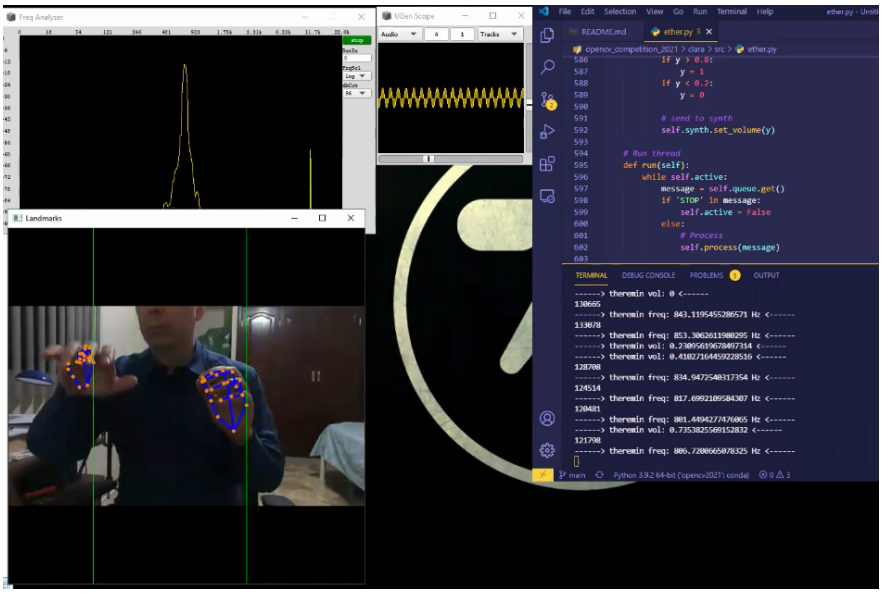

Figure 3.1: Early attempt using hand landmark model in device.

To tackle the problem we built tools to capture and replay depth maps (the depth output from a `StereoDepth` node). We realized that the depth stream output by the OAK-D could be constrained to a region of interest on the host side. These are precisely the regions where the hands move to play the instrument, and are very constrained regions. We built tools to figure out where to set the limit of those regions. There should be a spatial (x, y) rectangular region and a z_{min} and a z_{max} thresholds for each hands. We decided to constraint the z dimension such that the right hand (which controls the tone) is beyond the tone antenna and before the torso of the player (from the point of view of the camera).

This preliminary configuration we called ROI calibration and it is done in various steps. The purpose is to locate the tone antenna, the torso, and define right and left hand ROIs:

- For torso location we used the spatial calculator with a manual control to move the ROI to the correct location, chosen by the user. This determines the z_{maz} location.
- For antenna location, initially we prepared a small dataset of theremin images in order to train a antenna detector but it was too small so we resorted to something quicker, simpler and precise: Aruco markers. We located one on the antenna and used depth estimation for the z_{min} location.
- For the left and right hand ROIs, in the beginning we experimented with hand tracking based on Mediapipe, but as previously mentioned it provided to be too imprecise for this use case. We resorted to the Aruco markers as for the antenna location.

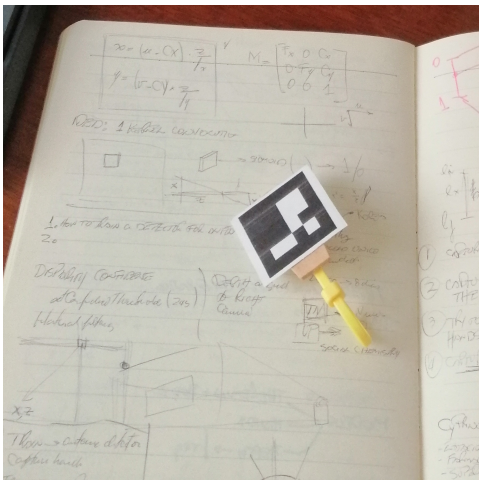

Figure 3.2: Marker for hands ROIs.

Figure 3.3: Marker for antenna.

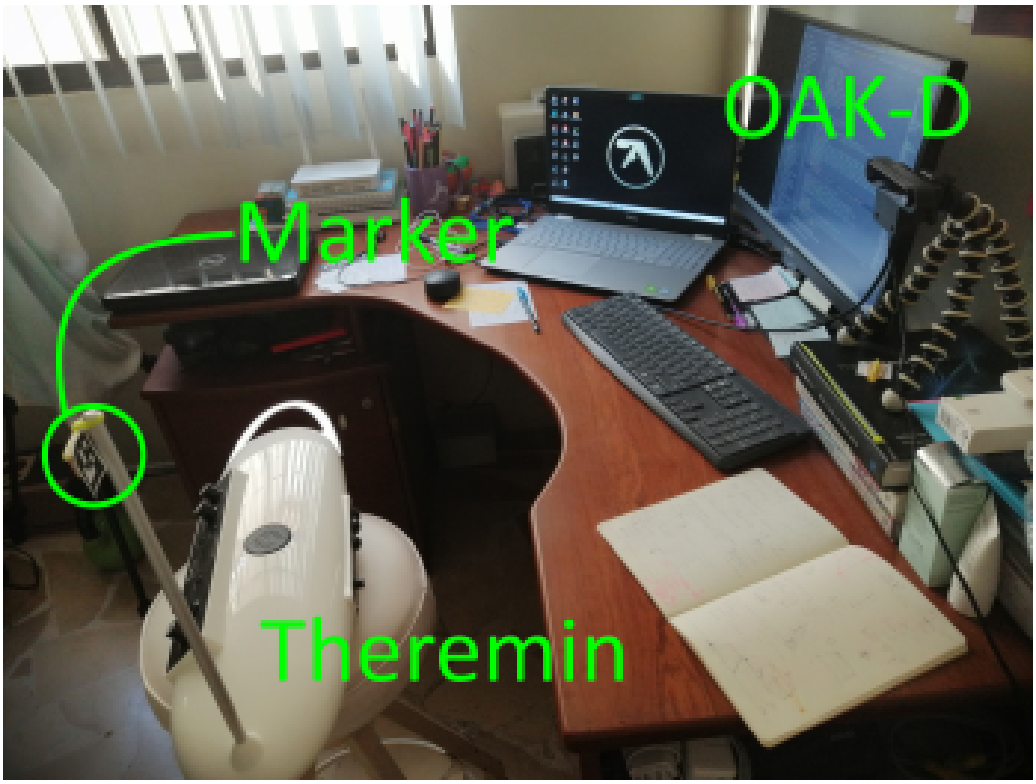

Figure 3.4: Setup.

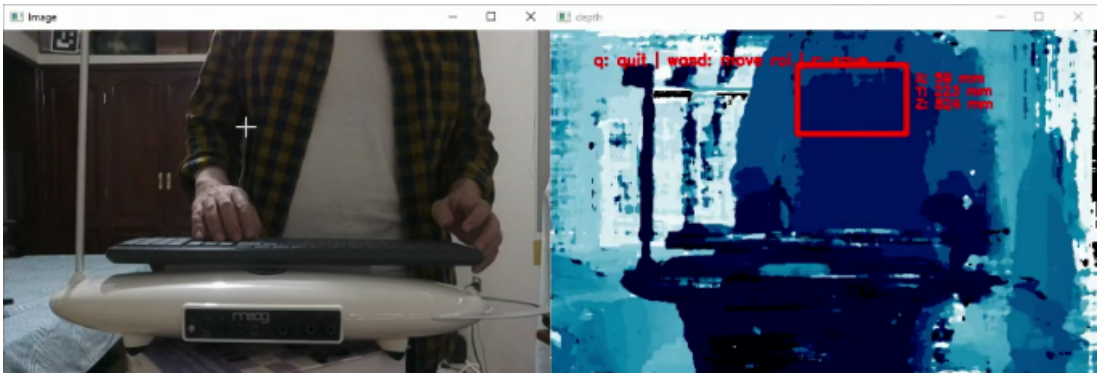

Figure 3.5: Body Location manually performed.

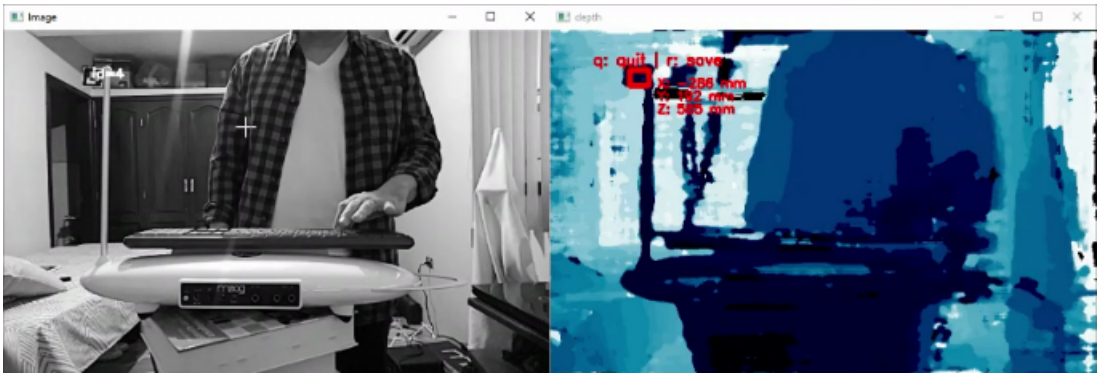

Figure 3.6: Antenna Location with marker.

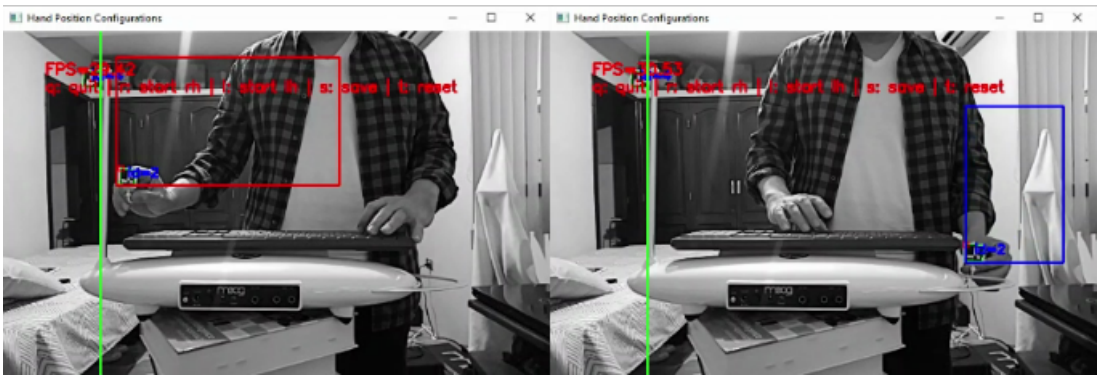

Figure 3.7: Left and Right hands regions.

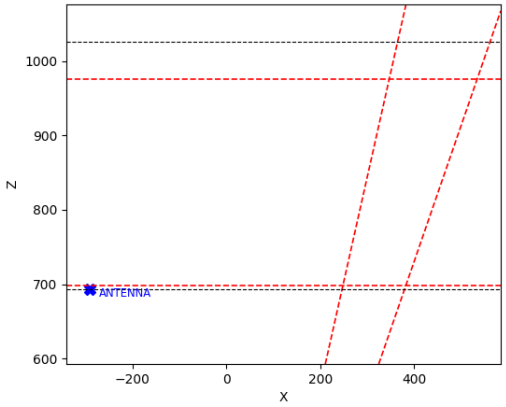

Figure 3.8: Left hand ROI.

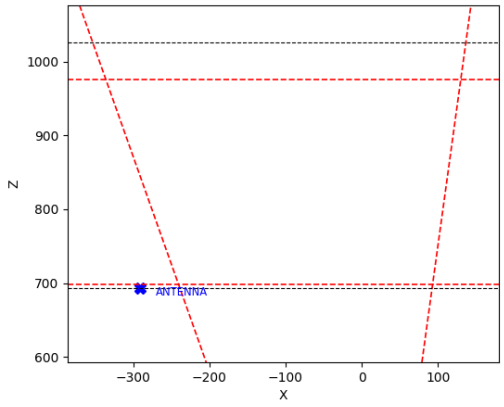

Figure 3.9: Right hand ROI.

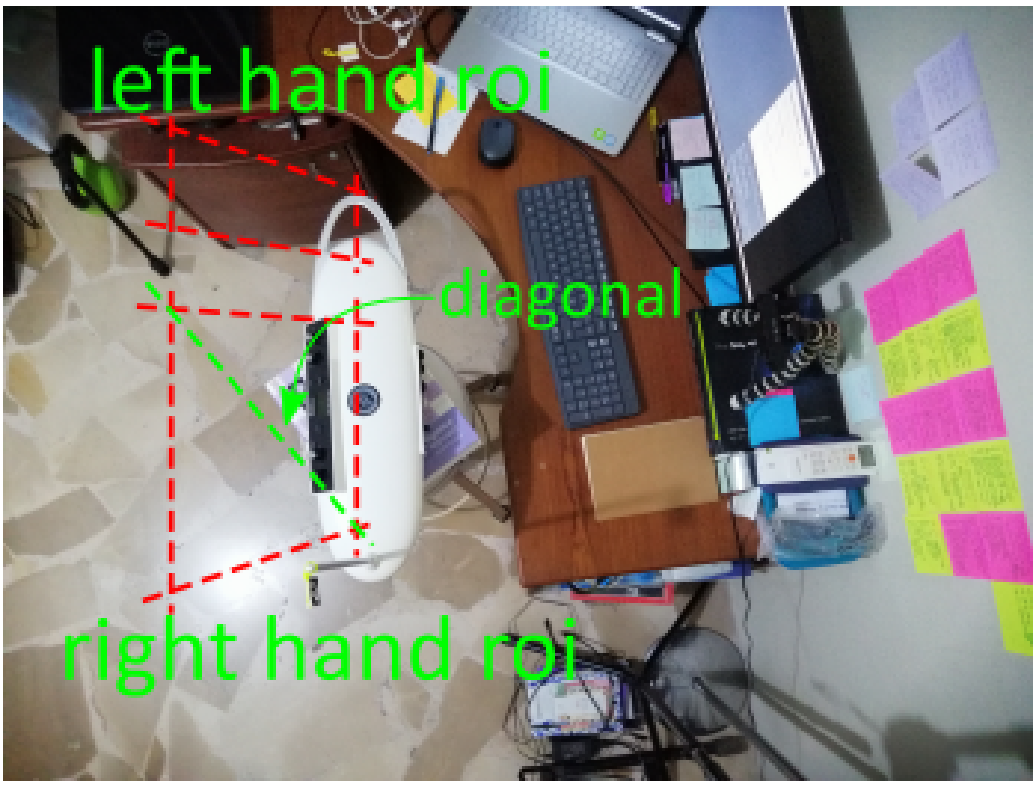

Figure 3.10: ROIs.

The whole configuration process is performed by a tool we created: `configurator.py`. In future work we want to explore a more automated way of doing this based on Blazepose [Bazarevsky et al., 2020]. There is a nice implementation for DepthAI by geaxgx [geaxgx, 2021a] we want to explore.

With the ROIs properly configured, we did perform data collection and subsequent analysis of depth frame captures. At first we considered machine learning to translate from depth to frequency. This would require more work as we needed to capture audio produced in a real theremin and synchronize it with the depth frames produced by the camera. At that time we have not received our theremin, so in order to make progress we resorted to consider simpler approaches first.

By using the intrinsics of the camera, we calculate the (x, y, z) coordinates from the depth map inside the region of interests, creating point cloud data for each hand. For the right hand for example the point cloud is projected to the plane XZ. Since the depth map is noisy, filtering is performed by deleting outliers in x and z coordinates. A simple statistical rule using IQR (inter quartile range) can be performed with acceptable results. Finally a centroid in the XZ plane is used to calculate the distance to the antenna (which is determined in the configuration step using the marker). This distance is transformed to a frequency through a linear or exponential mapping. Further filtering is added to consider a more precise location of the fingers. By filtering out z values that are too far away from the point of view of the camera, for example if z is higher than a threshold (the mean z_{mean}) be get a centroid more correlated with finger movements.

An additional idea we came upon was to project the centroid along the diagonal formed by the antenna and the most extreme point (closer to the body and far away from antenna) of the right hand ROI. In figure 3.10 there is a depiction of the ROIs and the diagonal we refer to (the green diagonal). For the left hand we followed a simpler approach by just projecting in YZ plane and taking the centroid after outlier removal.

In the next figures we present some of our findings. In figures 3.11 and 3.12 we see captures of the right hand using `datasette_depth_calibration.py`. As shown, we threshold z values based on the ROIs defined in the calibration phase. Figure 3.13 shows multiple views of the point cloud obtained by converting the depth frame to (x, y, z) coordinates. In Figure 3.14 we see histograms of the X and Z values. Figures 3.15 and 3.16 present the projected points on XZ plane for the right hand and the projected points on YZ plane for the left hand. We can see the centroids in the right hand projection: the red point is the centroid considering all the points inside the region, the green one is the centroid after outlier removal and the blue one is the centroid after discarding z values beyond a threshold. Observe that z values are highly discrete due to limitations of the depth map bit depth.

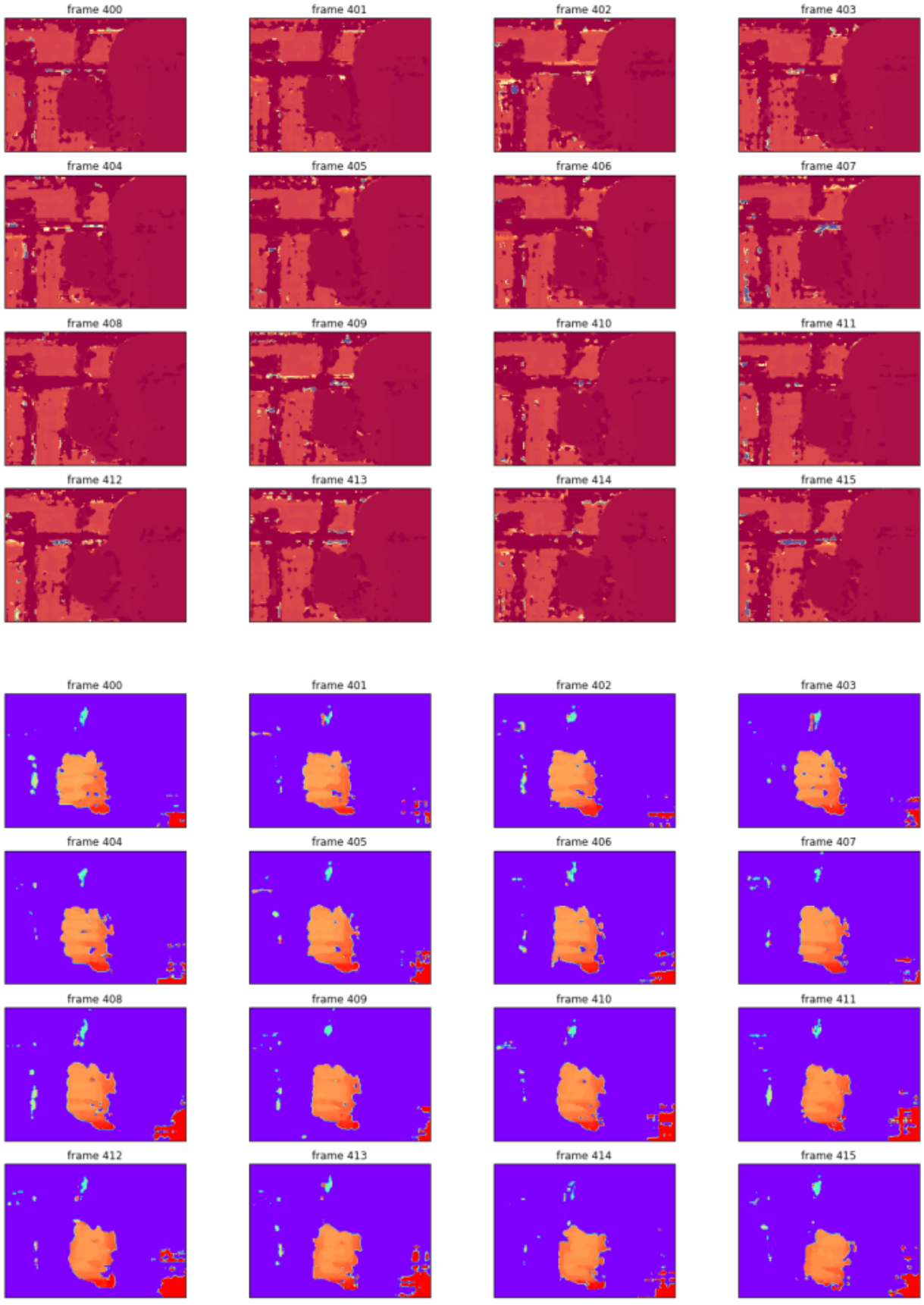

Figure 3.11: Right hand frames capture before/after applying threshold (1).

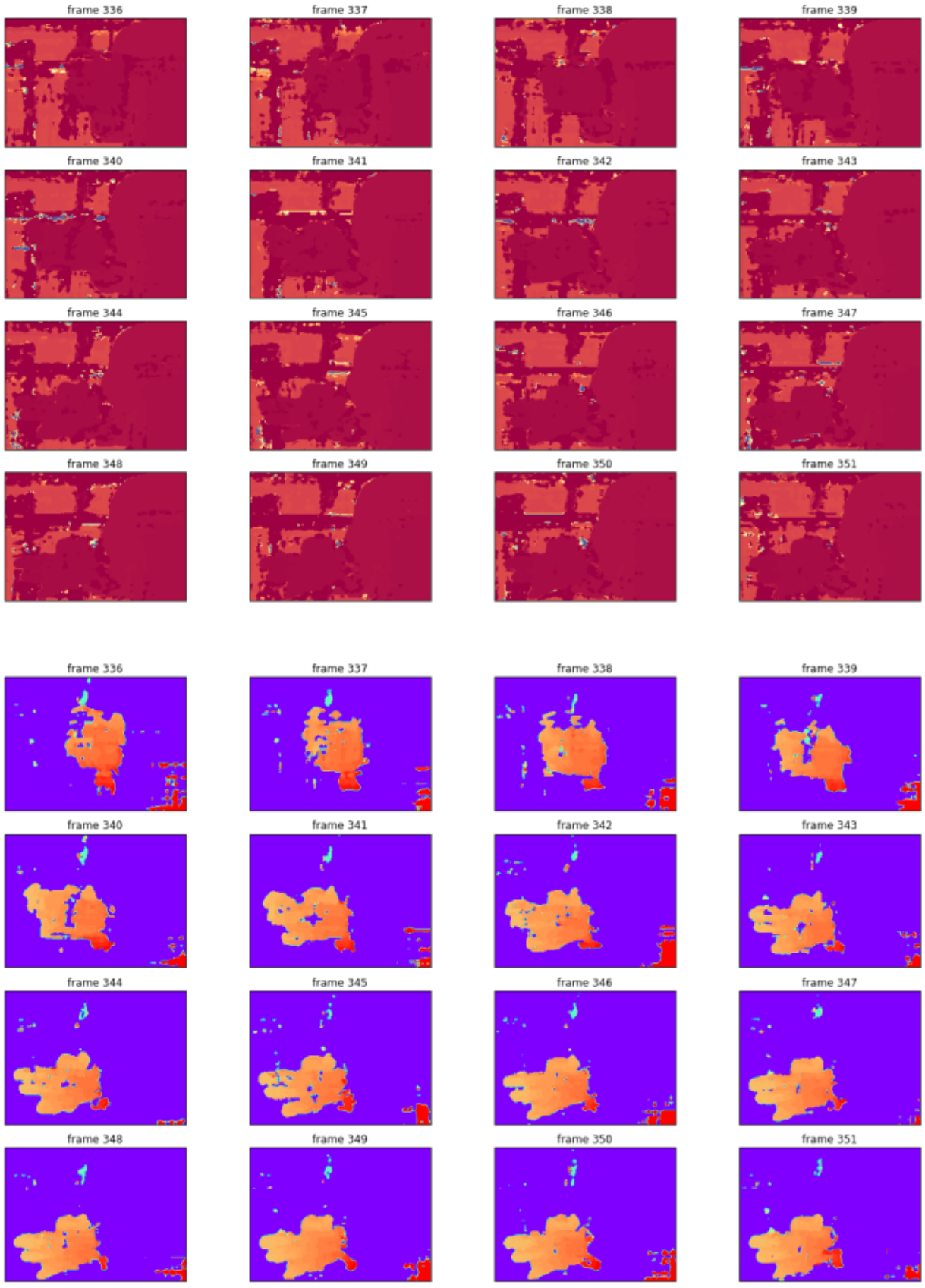

Figure 3.12: Right hand frames capture before/after applying threshold (2).

Figure 3.13: Point Cloud of a capture.

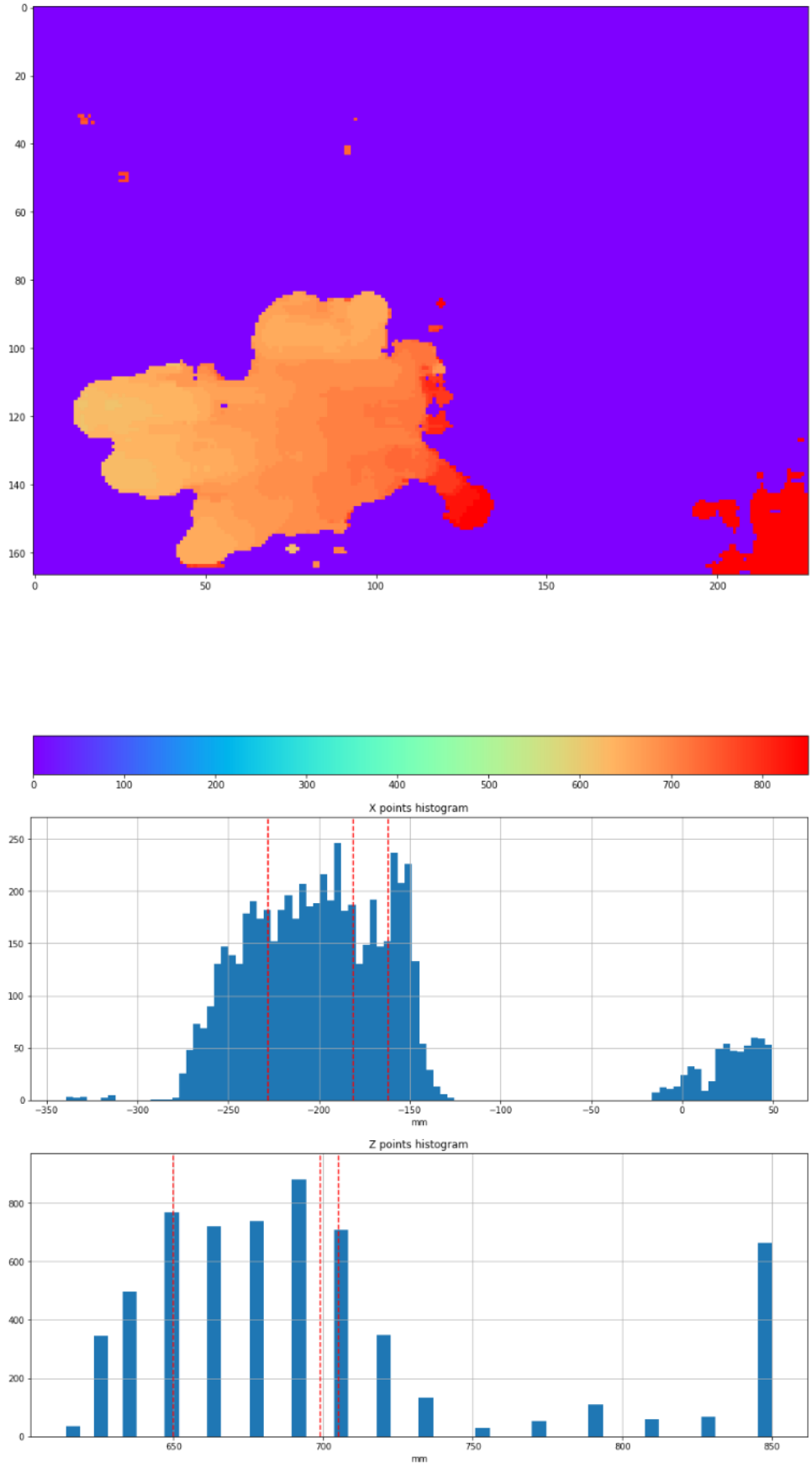

Figure 3.14: A capture and statistics in X and Z dimensions.

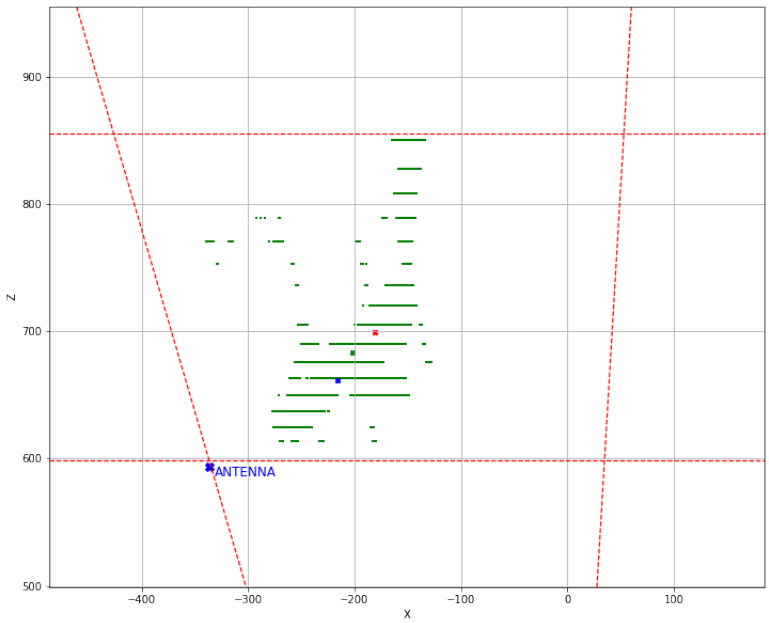

Figure 3.15: Right hand projection on XZ plane

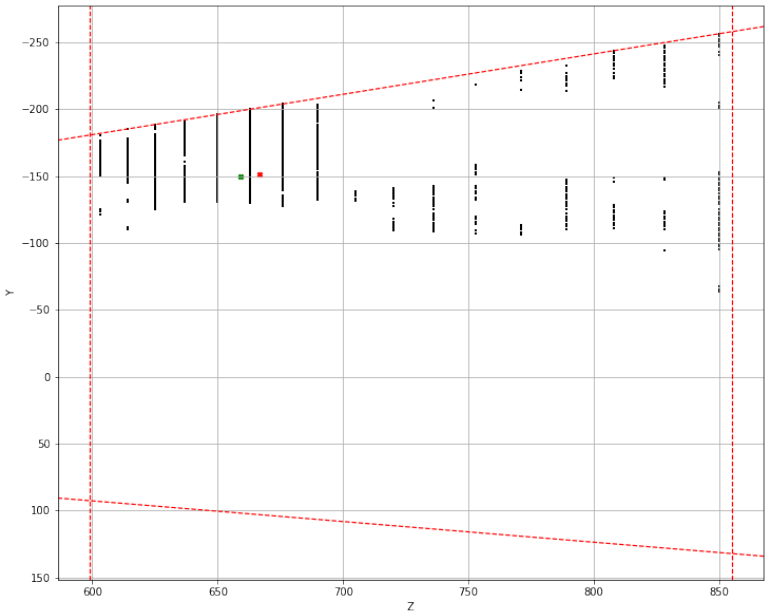

Figure 3.16: Left hand projection on YZ plane

3.1.3 The simulator

We are using Supercollider to receive frequency and volume values from our program and produce the sound accordingly. We have a visualizer that shows only the ROIs and process frames: `roi_depth_visualizer.py` that can be run after configuration. There is a `Threads` based version of the instrument: `ether_demo.py`. A newer version using `Multiprocessing` is in `ether.py`, but it is still under construction.

The sound generation is done by using a Supercollider script which receive frequency and volume parameters by OSC messages using UDP. Another approach would be to emulate directly the physics of the theremin by simulating an heterodyne circuit. We have found some interesting ideas in a forum for theremin related stuff [ThereminWold, 2021], but we did not advance the implementation.

Frequency values are the result of translating the distance from point cloud projection (XZ) centroid to antenna location, to a frequency value. Distance is converted to a `[0-1]` range which then is converted to a point in an equal-tempered scale for a number of octaves. For volume values a similar approach is performed but there is no distance calculation, only the average inside the y_{min} and y_{max} values defined by left hand ROI.

In the next figures we see an `ether_demo.py` capture, at around 70 fps (even though we set 120 fps using the Depthai framework). In Figure 3.17 we see an screenshot of the system in action. The frame rate was severely hurt by the screencast recording software so it shows 25 fps even though it can run at 60-70 fps if screencast recording software is not running. In Figures 3.18 and 3.19 we compare two captures of the spectrogram of a Theremini frequency sweep and our simulator frequency sweep. We note that the Theremini has harmonics over the fundamental frequency and therefore its characteristic sound. This can easily be improved by using a better sound generator (for our demo we use a sine tone generator). A more concerning issue can be seen in our simulator: it has a discrete nature. We have identified three possible causes related to this behaviour:

- The first one is the frame rate. After a lot of experimentation, we found that at 30 fps we hear discretized frequency jumps from one position to another. At 60 and 70 fps the situation improves but we still perceive it sometimes. One technical problem we had was the difficulty to make the device operates at 120 fps. We think this is in part due to usb connectivity with our computer. Also displaying visual frames at the same time hurts latency. A solution we are still developing is to use multiple processes: one for streaming depth, one for processing the stream and transform it into frequency and volume messages, and one processing the stream and display everything in a GUI. The GUI is still missing in our demo. The new version is in `ether.py` and will consider it.
- Noise is another issue. The depth map is by nature very noisy, which produce frequency jumps in distant points of the spectrum. This is the reason we perform a little bit of filtering and averaging. A way to improve this would be to consider filtering either at the disparity/depth map level or at the output frequency level. Some alternatives we have seen on the Depthai Discord channel are the use of WSL, and bilateral filters. At the output level we can also use filtering. All those options involve counting on higher frame rates to have a good enough responsiveness. One final idea is that the frequency

producing mechanism run at higher rates and send a stream of interpolated values between 2 successive frequency values to smooth things out. Figure 3.20 shows how it should work.

- A final culprit is the highly discrete nature of the depth capture (z values). One way to solve this is to try the `subpixel` option and consider treatment to noisy depth registry.

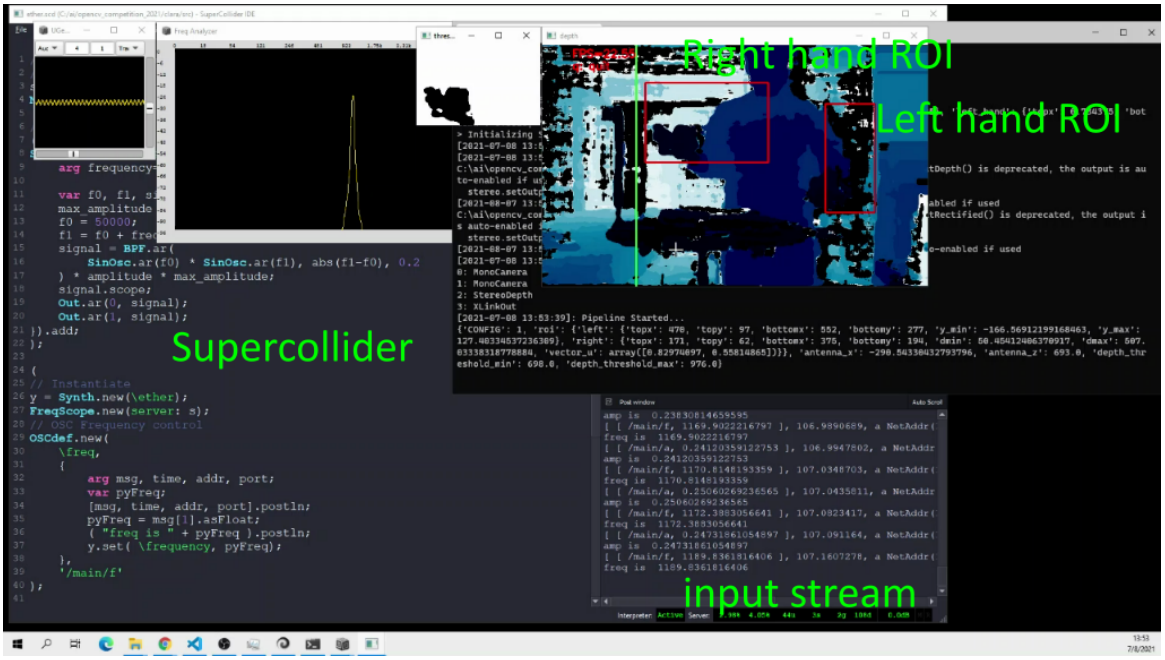

Figure 3.17: A capture of the simulator in action.

Of course this approach is not fully faithful to the real theremin, but it is a starting point. It produces acceptable sounds. Combined with a better sound generator, it can be improved. And also our solution can be used as a music controller. We plan to do more research in this area and we are developing tools for capturing depth frames and audio produced by a real theremin and synchronize them for a data driven solution. An idea here is to use machine learning to come up with a function to translate depth to frequency capable of smoothly interpolating between frequency points. Some ideas to try are neural networks, kernel methods (support vector regression) and mixture of gaussians, combined with regularization. Interesting and related research we have found: Hand PointNet, a 3D based hand pose estimation by [Ge et al., 2018], which promises a 48 fps real time estimation, Hand segmentation and fingertip tracking in [Nguyen et al., 2020], and Hand segmentation from single depth image as in [Li et al., 2016].

The previous issues would require further work. It would be nice to process almost all the algorithms on the device such that it would only stream the frequency and volume values. A highly optimized `C++` port would be preferable as well for future versions.

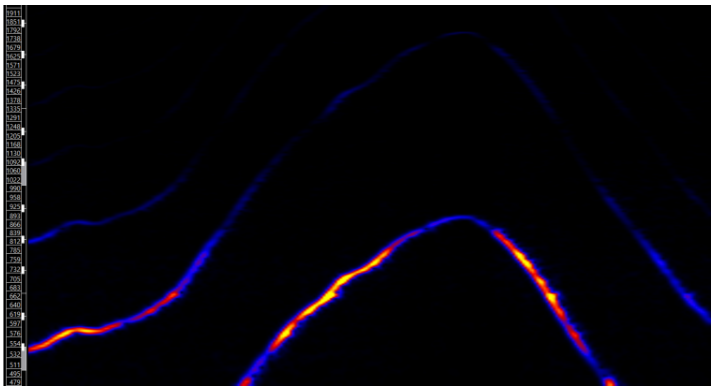

Figure 3.18: A frequency sweep spectrogram by the Theremini.

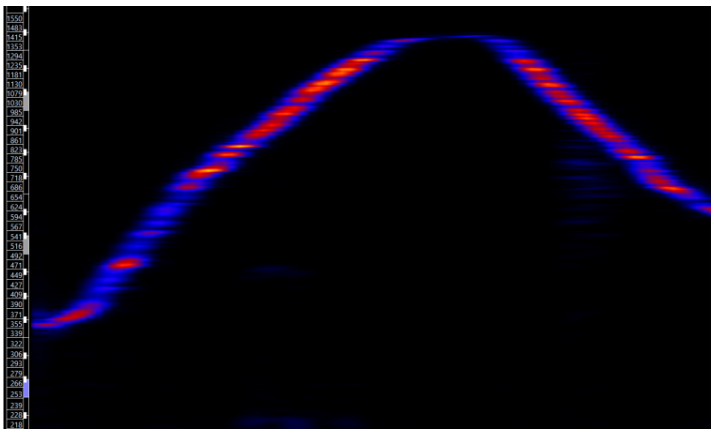

Figure 3.19: A frequency sweep spectrogram by our theremin simulator.

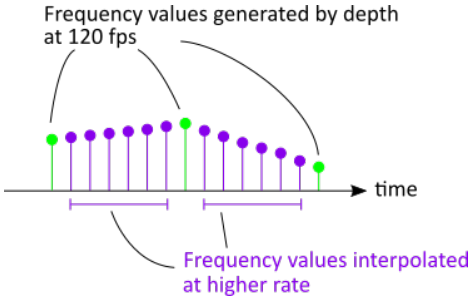

Figure 3.20: Interpolation idea.

3.2 Calibration

As mentioned previously one of the purpose of building a theremin simulator is to try to copy the dynamics of the theremin. This topic involves research in two areas: the physics of the theremin or a way to simulate it, and the method for playing the device, in particular the way to play notes. We are still developing the former one. The latter proved to be a highly complex endeavour. Here we describe our findings.

One method, developed by Carolina Eyck [Eyck, 2006], proved to be a proper starting point. The method is based on correlating 8 hand positions to notes in a scale (the easiest one is a major scale, but this could be adapted to any scale and a musical piece). Theses positions are hand configurations that produce the correct frequency. Since Eyck's method is very systematic in the way the configurations are made, we thought that these could be learn by a data driven approach.

Nonetheless, we found two difficulties:

- The theremin we experimented on is a digital one created by Moog, it is called Theremini, and it is considered an entry level one. The key point in Eyck's method is that a scale is played by progressively going from a closed hand position to an open one, rotating on the wrist while doing this. We found it very hard to adapt Eyck's method to the Theremini. In part because an analog theremin has a knob that allows to expand or compress the mapping between distance and frequency. This is the way an analog theremin can be calibrated according to the hand of the player in relation to Eyck's method so that you can have a scale by moving from a close to an open position. The Theremini has a calibration procedure but is very hard to make it correspond to a scale in the way analog ones can. Due to limitations having access to an analog theremin, we decided not to adopt this method.

- The second issue, has to do with the nature of the method itself and its fitness for a computer vision solution. Even though there are 8 positions in a scale, some positions are very similar to each other, making harder to distinguish between each other. Despite this difficulty we explored a nice approach by cortictechnology [michaelhwn, 2021], which is a system for sign language recognition, and tried to adapt it to our problem. Further development needs to be done in order to build a bigger dataset for proper learning of each positions.

Due to the mentioned considerations, we resorted to constraint the problem to recognizing fewer positions (3 or 4) and building on top of of geaxgx implementation of Mediapipe hand landmark detection [geaxgx, 2021b]. The idea is this: in a calibration step we ask the user to perform 3 or 4 predetermined positions and using the landmarks data, train a positions recognizer. We are still developing this idea and at the moment we have built tools to collect a dataset of landmarks (`datasette_hand_recorder.py`). Features can be added by computing the angle of rotation with respect to the wrist landmark and the separation of 3 finger tips, convex hull, and others. A simple approach to a fast recognizer would be a nearest neighbour classifier. Further inquiry will be performed in this regard.

3.3 Practice System

The practice system depend heavily on the previous subsystems. In particular, in order to signal the user the location of the hand for the next note, a faithful representation of the real theremin should be achieved first. The feedback system should be based on a user interface, preferably based on a 3D representation. It needs to display the current note, the position of the next note, and also the region in space where the hand should be located. Point clouds would be a nice visualization addition.

For GUI we have experimented with tools to reproduce captured depth frames datasets, to see how it could impact performance. Two of these tools are in `datasette_player.py` and datasette_player_matplotlib.py. For a better GUI, frameworks like pygame, Kivy, Unity should be considered.

4 Conclusions, Applications and Future Work

So far we have achieved a proof of concept for our virtual theremin or theremin simulator. Our ideal solution would have all the 3 subsystems. Though each of the subsystems turned out to be a big problem by itself. We underestimated the effort required for the whole project. Nonetheless we enjoyed researching it and exploring solutions.

One difficulty we faced was having access to a real theremin. Many of our expectations and assumptions turned out to be wrong when we finally set our hands on a real one and started playing with it. And this is a key point in any AI or CV based solution. One should understand the domain of the problem and explore it thoroughly with domain experts.

After months of analysing theremin methods and our implementation of a virtual theremin, we are considering creating our own method that utilizes fewer and simpler hand positions but will depend heavily on visual feedback about where to locate the hand. This, in turn, should depend on augmented reality technologies for the user experience. Another advantage of proposing a simpler method is that we can focus on researching a more faithful theremin simulator such that we depend more on visualization of note location and less on gestures. For this reason we think more research has to be done in order to improve the virtual theremin dynamics and faithfulness.

In retrospective what we achieved is the creation of a theremin which depend heavily on computer vision for its inner working. We appreciate the introduction to the market of a sensor for computer vision that provides disparity and depth for the DIY and the open source community at an affordable price tag. We believe it opens the possibilities for the creation of musical interfaces and the research in musical education.

References

[Bazarevsky et al., 2020] Bazarevsky, V., Grishchenko, I., Raveendran, K., Zhu, T., Zhang, F., and Grundmann, M. (2020). Blazepose: On-device real-time body pose tracking.

[Eyck, 2006] Eyck (2006). Theremin method. `https://www.carolinaeyck.com/method`.

[Ge et al., 2018] Ge, L., Cai, Y., Weng, J., and Yuan, J. (2018). Hand pointnet: 3d hand pose estimation using point sets. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[geaxgx, 2021a] geaxgx (2021a). Depthai blazepose. `https://github.com/geaxgx/depthai_blazepose`.

[geaxgx, 2021b] geaxgx (2021b). Depthai hand tracker. `https://github.com/geaxgx/depthai_hand_tracker`.

[Li et al., 2016] Li, M., Sun, L., and Huo, Q. (2016). Precise hand segmentation from a single depth image. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2398–2403.

[michaelhwn, 2021] michaelhwn, y. (2021). American sign language (asl) recognition using hand landmarks. `https://github.com/cortictechnology/hand_asl_recognition`.

[Nguyen et al., 2020] Nguyen, D. H., Do, T. N., Na, I.-S., and Kim, S.-H. (2020). Hand segmentation and fingertip tracking from depth camera images using deep convolutional neural network and multi-task segnet.

[ThereminWold, 2021] ThereminWold (2021). Heterodyn circuit. `http://www.thereminworld.com/Forums/T/30562/lets-design-and-build-a-simple-analog-theremin`.

[Zhang et al., 2020] Zhang, F., Bazarevsky, V., Vakunov, A., Tkachenka, A., Sung, G., Chang, C.-L., and Grundmann, M. (2020). Mediapipe hands: On-device real-time hand tracking.