E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

GuidingHaptics

Máster Universitario en
Ingeniería Informática

Trabajo Fin de Máster

Stefan Donkov Bogdanov

Asier Marzo

Pamplona, 18/09/2023

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Acknowledgements

I want to thank all the members of the UPNALAB research group for making this group as welcoming and friendly as it is, as well as their assistance in the development and testing necessary for the completion of this project.

Summary

A user study comparing different guiding devices was conducted to evaluate the mental workload of their usage. Several devices were evaluated, among them, GuidingHaptics which vibrates asymmetrically to produce a pulling sensation in the hand, a vibration band on the user's wrist whose stimuli must interpreted as a direction, and finally a pair of headphones that produce beeping sounds as means of guidance.

The guiding devices and their respective drivers were first constructed, and the software necessary for their operation implemented. Tuning was required to produce sufficiently strong, perceptible, and interpretable stimuli.

For the user study, a program was implemented to control the guiding devices and collect the subject's objective data. The users were required to fill in a NASA Task Load Index (NASA-TLX) questionnaire. A statistical analysis on the subjective and objective data was later conducted.

Keywords

Haptics, Guidance, Asymmetric Vibration, Mental Workload, Pulling Sensation, Pulling Illusion, Voice Coil, Linear Resonant Actuator (LRA)

Index

Contents

State of the Art

The work done aims to expand the knowledge in asymmetrically vibrating devices, capable of inducing a pulling illusion for guidance. To do so, an investigation into the mental workload caused by such devices has been conducted.

The pulling illusion is extensively present in the literature [1], [2], reporting how different parameters, like phase amplitude and frequency of its signals, affect its perceived magnitude.

There are different methods to create guiding cues. For example, the use of flywheels [3] to induce a force towards an arbitrary direction, or the use of vibration motors attached to the arm or wrist to allow directional or rotational guidance [4]–[7].The sensation produced by the pulling illusion, has also been explored for the creation of guiding devices in several previous papers [8], [9].

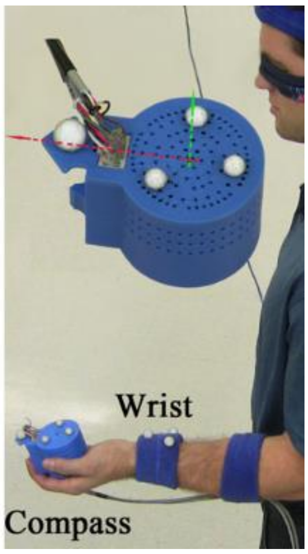


Fig. 0.1 Device used in [3]

Fig. 0.2 Device used in [2]

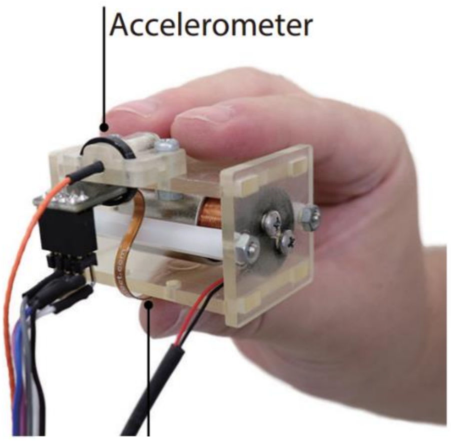


Fig. 0.3 Device used in [1]

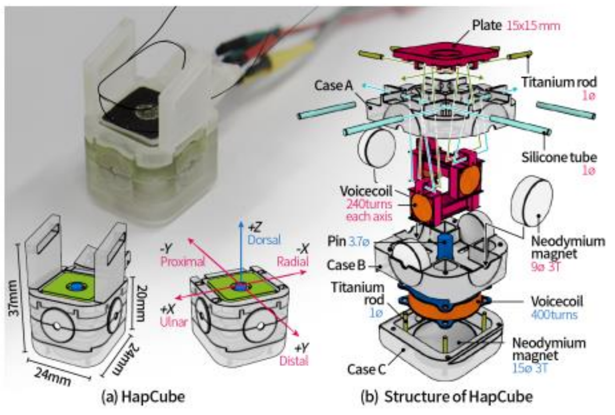


Fig. 0.4 Device used in [9]

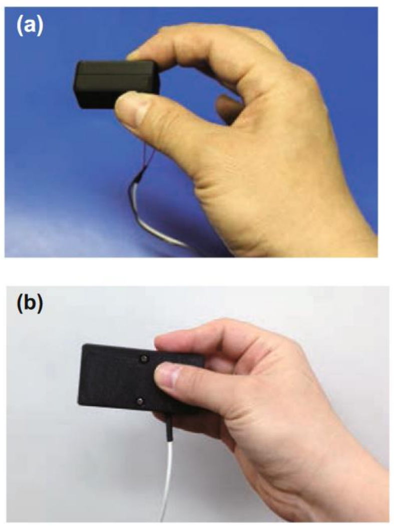


Fig. 0.5 Devices used in [8]

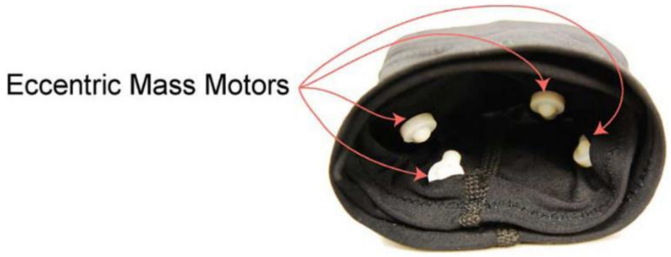


Fig. 0.6 Device used in [4]

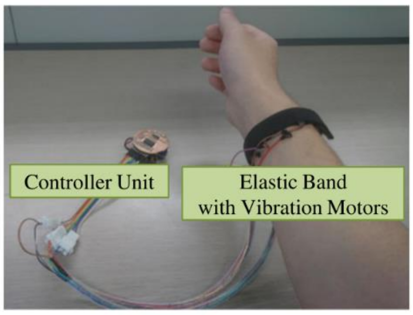


Fig. 0.7 Device used in [6]

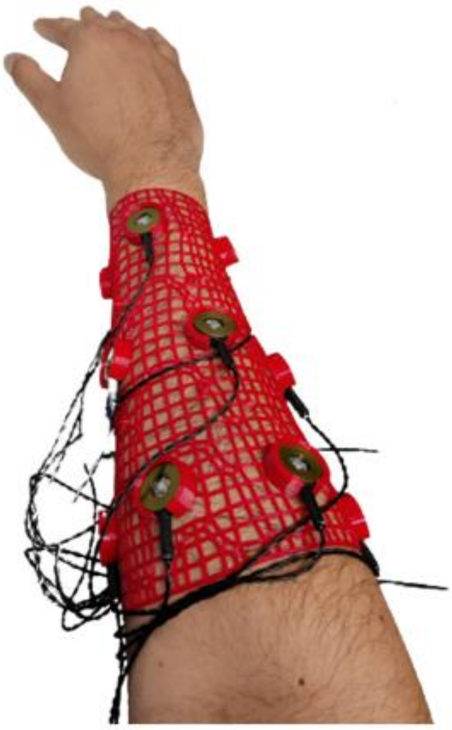


Fig. 0.8 Device used in [5]

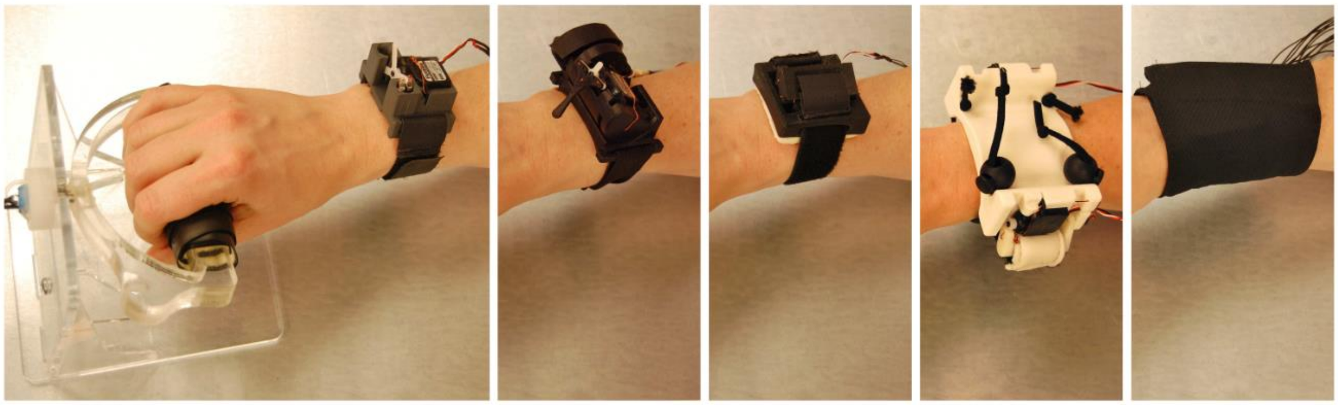


Fig. 0.9 Devices used in [7]

However, there were no conclusions found about if, the use of pulling-illusion-based devices over traditional methods of guiding, could have any advantages. Methods to compare these devices, both subjective and objectively, can be used to determine which guiding method is superior or preferred by users. For instance, NASA-TLX and SWAT questionnaires [10], [11], subjectively measure the mental workload produced while using guiding devices. On the other hand, objective measurements such as TCT and accuracy on a secondary task [12], are used to address the users' performance.

When testing mental workload, an appropriate secondary task must be selected in order not to cause unwanted interference with the main task. In this case, for the primary task of guidance, there are several appropriate secondary task that can be applied, such as arithmetic operation solving [13], and Stroop effect tests [14].

Methodology

Assembly of the devices

GuidingHaptics

The guiding cues that this device will employ are based on the “pulling illusion” produced when a person’s fingertips are subjected to an asymmetric vibration [15]. Several commercial solutions capable of generating the pulling sensation exist, among them voice coils and Linear Resonant Actuators (LRAs).

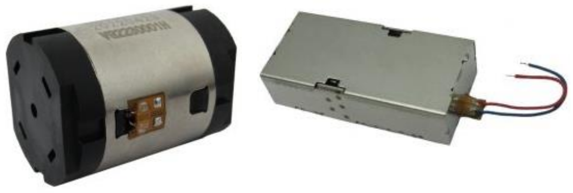


Fig. 1 Voice coil actuator and LRA [12]-[13]

The signals that produce the pulling illusion have already been studied in several different papers [16]–[19]. To generate these signals a signal generator [20] was first used, however its output power was not enough to drive the actuators directly. An attempt to amplify the signal with a digital audio amplifier was made but its minimum output power was too high for the actuators. As a final solution, an Arduino Nano [21] was used to generate the signals programmatically in combination with a L298N module [22] to allow for an amplification from 5V up to 40V.

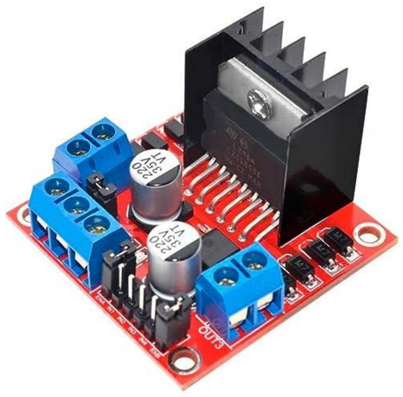


Fig. 2 L298N module [8]

The code implemented for the Arduino Nano allows for a selection between several different signals proposed by previous literature, as well as the input of a custom waveform generated by means of WavePainter [23], a web UI that allows the painting of custom signals.

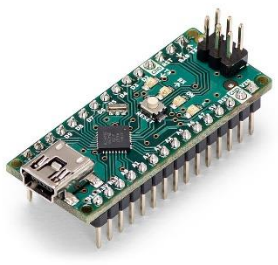


Fig. 3 Arduino Nano [7]

Several different actuators [24]–[27] were tested in order to find the one that produces the best illusory sensation of force. Some of them were rapidly discarded since the vibration was not tangential to the skin, and others were able to produce a feeling of directionality when being held but didn’t produce a pulling sensation. The actuators shown in Fig. 1 are the ones that were able to produce a perceptible force, being the one from the voice coil the strongest of the two.

By means of Red Pitaya [28] and an analog accelerometer module [29], a partial analysis on the frequency response of the voice coil was conducted. The phase of the output signal of the actuator was not able to be captured by the device, therefore, only the amplitude of the vibrations captured by the accelerometer was used, and the graph in Fig. 6 was obtained. This graph shows that the peak amplitude produced by the actuator is when driven between 60Hz to 70Hz, which coincides with the manufacturer's specifications for the resonant frequency.

Fig. 4 Red Pitaya [14]

A measurement of the response of the actuator to the different waveforms proposed by the related work was also conducted, obtaining the results shown in Fig. 7. Some of the signals produce a symmetric vibration of the actuator which does not produce the desired pulling sensation, and the first signal shown in the figure produces the most asymmetric vibration, this being the reason it was chosen for future tests.

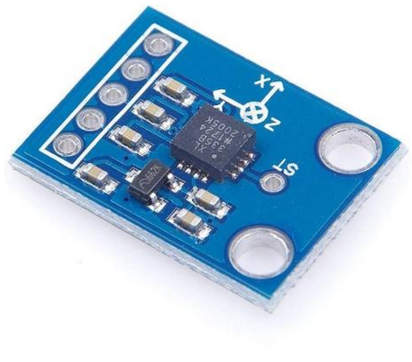


Fig. 5 ADXL335 accelerometer [15]

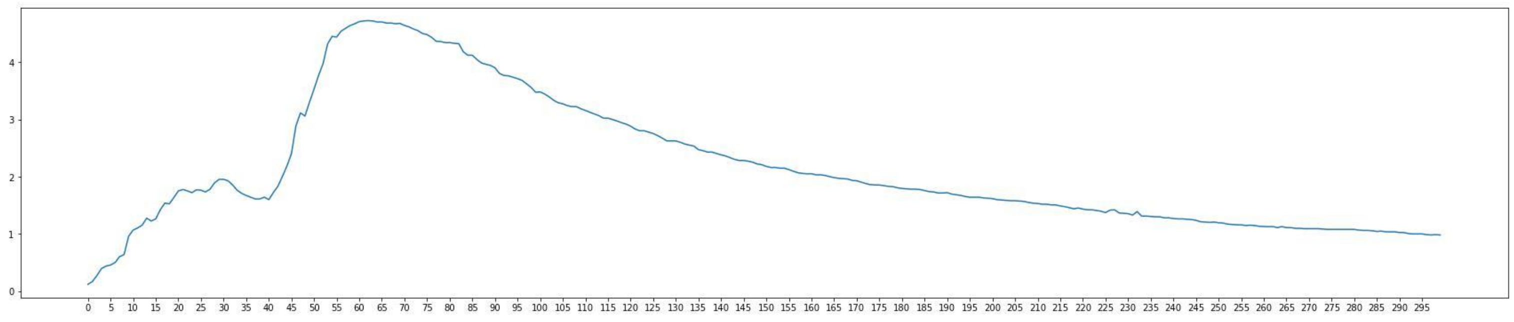


Fig. 6 Frequency response graph

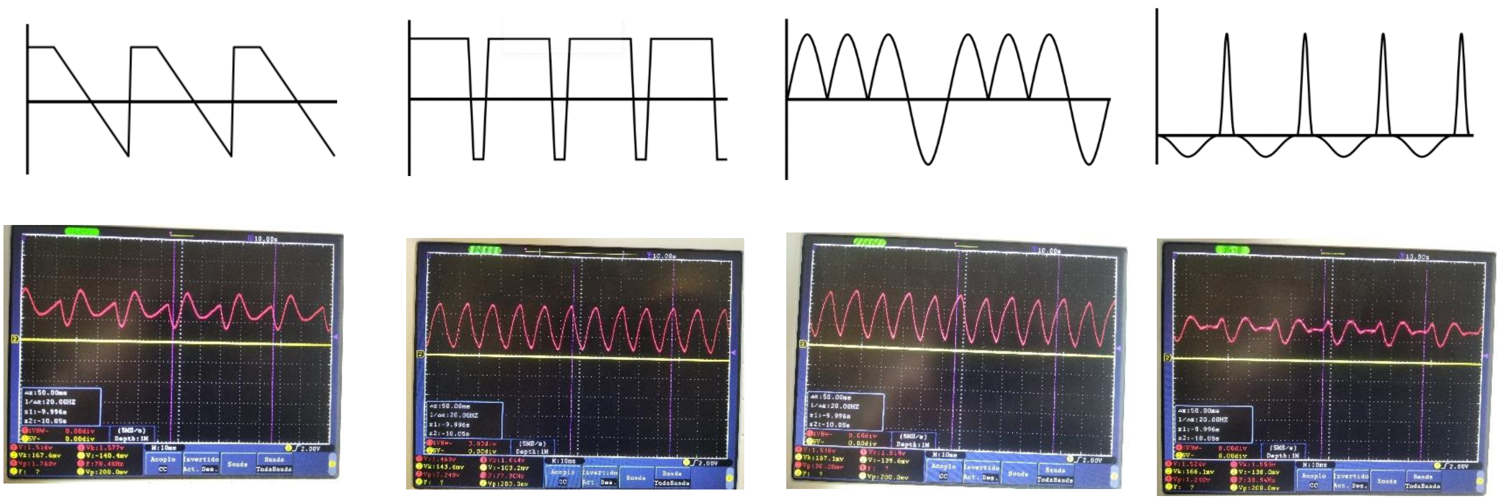


Fig. 7 Input signals (top) and outputs (bottom) of the accelerometer captured with oscilloscope.

According to literature the optimum frequency of vibration to induce the pulling sensation is between 10Hz and 50Hz [16], as it stimulates the Meissner corpuscles which are capable of perceiving the direction of the vibrations [4].

This presents a problem since the actuators should have a resonant frequency between 10 and 50Hz to maximize the amplitude of the vibrations at the frequencies perceived by the mechanoreceptors responsible for the pulling sensation. However, no such commercial actuators exist.

A preliminary user study was conducted to decide the optimum holding position of the device as well as the frequency producing the strongest pulling force according to the subjects. From the data shown in Fig. 8, the frequency of 38Hz was chosen for future studies and experiments, as it was the one which was felt as the strongest by the highest number of participants.

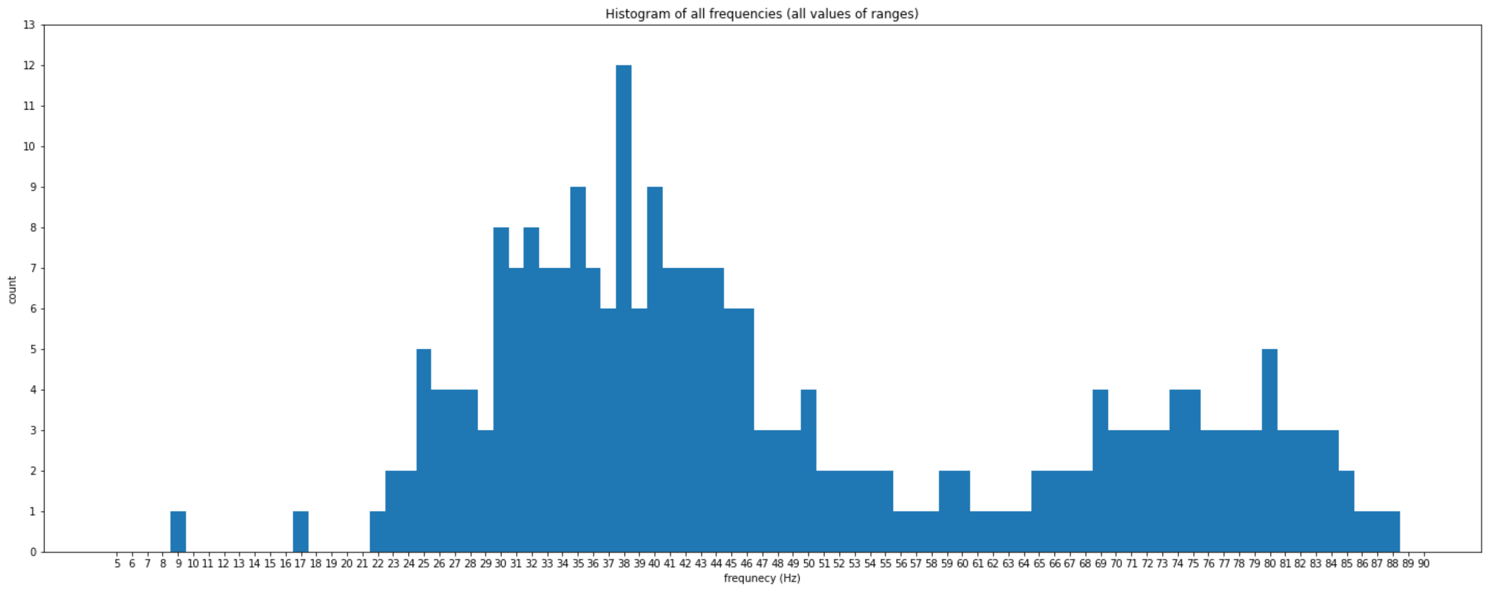


Fig. 8 Histogram of frequencies which produce the strongest pulling sensation reported by the participants.

It is to be noted that there are 2 distinct groups of frequencies producing the strongest pulling sensation. This is believed to be because the actuators resonant frequency is around 70Hz, thus producing the highest amplitude vibrations near this frequency, and, since the directionality of the vibrations can be perceived at those frequencies, subjects might have confused the feeling of directionality with the pulling sensation. The subjects should not feel pulled at frequencies near 70Hz as the mechanoreceptors responsible for the pulling illusion are not stimulated [16].

As for the holding position that provides the strongest pulling sensation, and the one used in following studies, the depicted in Fig. 9 was chosen, as it maximizes the contact with the fingertips and produces the vibration tangentially to the contact surface. And according to some users the sensation was stronger when holding the actuator that way, while other users reported it felt similar if not equal in strength to the other holding position (Fig. 10) studied.

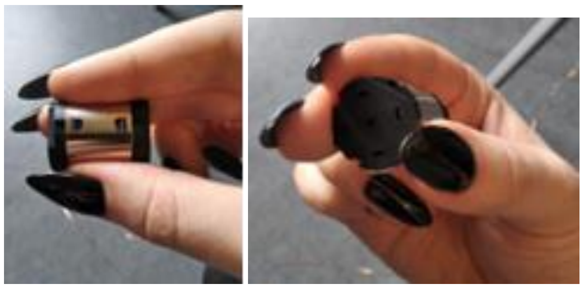


Fig. 9 Holding position chosen.

A modification of the device was necessary to improve the feedback of the device in one of the vibrating directions, since as reported by the users, and objectively measured with an accelerometer, one of the directions produced a weaker pulling sensation (Fig. 11). It is observed that for opposite input signals, the outputs of the actuator are not exactly opposite, we see that the two peaks of amplitude in the left output have a greater difference in magnitude than the peaks of the right output.

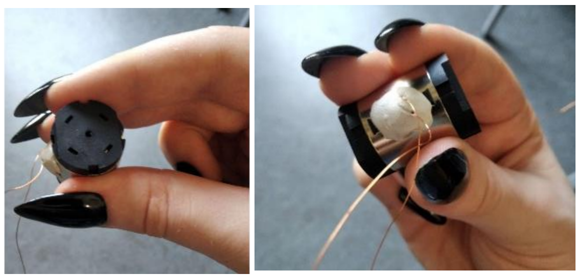


Fig. 10 A different holding position.

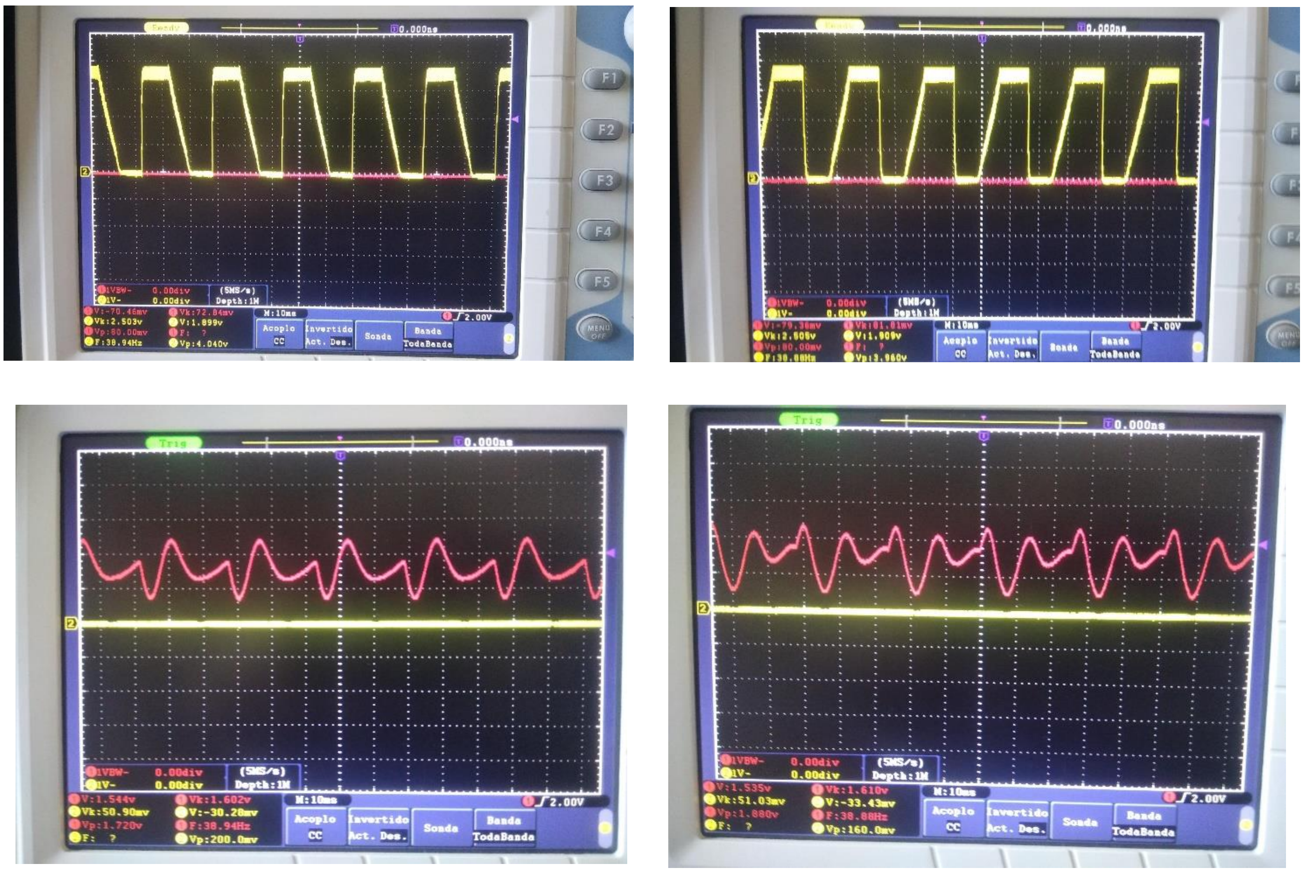


Fig. 11 Difference of asymmetry in the outputs of the actuator.

It is suspected that the cause of this could be an irregularity in the actuator making it not respond as expected. An attempt to remedy was made, taking inspiration from the DIY actuator used in WAVES[18], an emulation of the rubber membranes was made using the membranes from a pair of earphones depicted in Fig. 12, as a substitution to the magnets inside the voice coil responsible for keeping the oscillator centered, see Fig. 13.

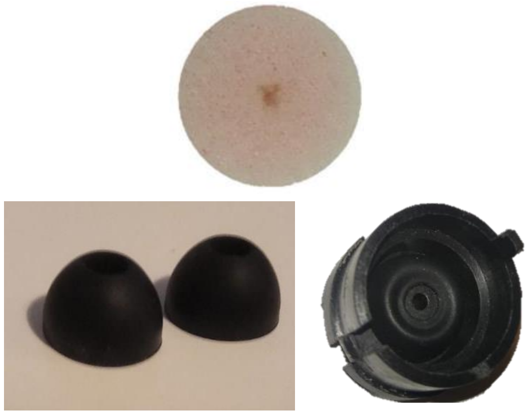


Fig. 12 Membranes used in WAVES [4] (top), earphone membranes (left), modification (right)

Discarded assemblies

There were several other designs tested to attempt to achive a stronger pulling sensation and a higher number of degrees of freedom.

The solution that was previously proposed only has 1 DoF of vibration. A logical train of thought would lead to the use of 2 or more actuators perpendicular to each other to allow for 2 or 3 DoFs, or for actuators to be controlled by pairs to create the sensation of torque enabling haptic feedback up to 6 DoFs. This however proved to be a very difficult endeavour, since just coupling actuators together leads to increases in weight and changes to their resonant frequencies, which both have a negative effect in the percieved strength of the pulling sensation, making it less if not completely imperceptible. Other problems arise when using more actuators, like the fact that it is difficult if not impossible to have a finger contacting all actuators and the other fingers contacting just one. The pinching of the actuator seemed to be key in order to induce a pulling sensation, and a holding position for 3 actuators was not possible. Despite the previously exposed hurdles, a device that allows for 2 DoF of vibration was constructed (Fig. 14). Hoewever it did not produce pulling sensation as strong as just using a single actuator. It is to be noted that it did produce a feeling of directionality similar to the one resulting from driving a single actuator at its resonant frequency.

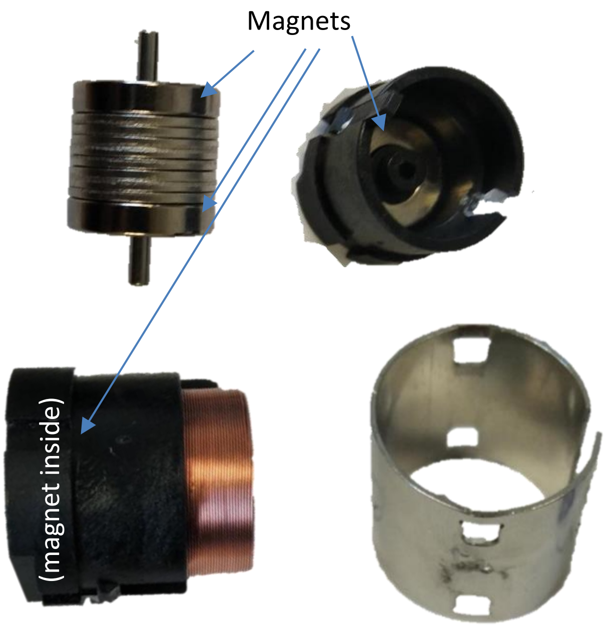


Fig. 13 Disassembled voice coil. Oscillator (top left), right casing (top right), left casing with coil (bottom left), metal casing holding all together (bottom right)

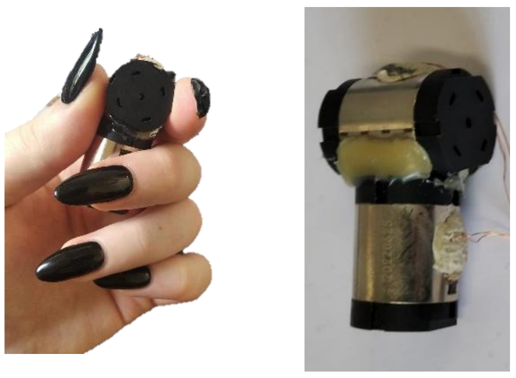


Fig. 14 2DoF device on the right, and its holding position, thumb contacting both actuators, index and middle fingers on different actuators.

Efforts in the increase of the percieved pulling force lead to an attempt in enlarging the contact area between the actuator and the subject’s skin. The first solution tried was encasing the actuator in mouldable plastic [30], see Fig. 15, this lead to a significant increase in the amount of skin in contact with the vibrating surface, as the entire hand could be wrapped around the vibrating surface. In spite of this, there was no increase in the strength of the sensation, and in fact it could not be felt at all. The cause of this could be the increase in weight changing the resonant frequency of the system and thus making the vibration produced by it no loger assymmetric. It is possible that encasing the voice coil causes the oscillator inside it to not move as freely as normal. The oscillator could have to compress the air when moving thus resulting in an atenuated vibration instead of a sudden “push” to one side and a slower move to the other.

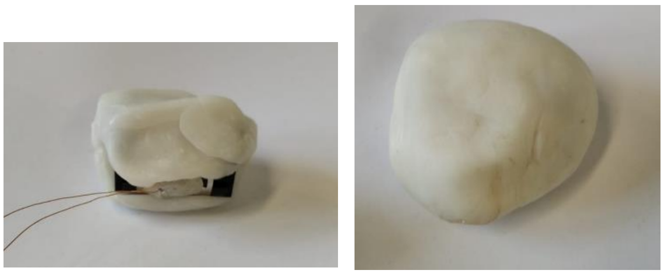


Fig. 15 Actuator partially encased in plastic (left), and fully encased (right)

3D printed casings, shown in Fig. 16, were also studied but did not yield a device capable of inducing the pulling sensation, probably because of the same reasons described previously.

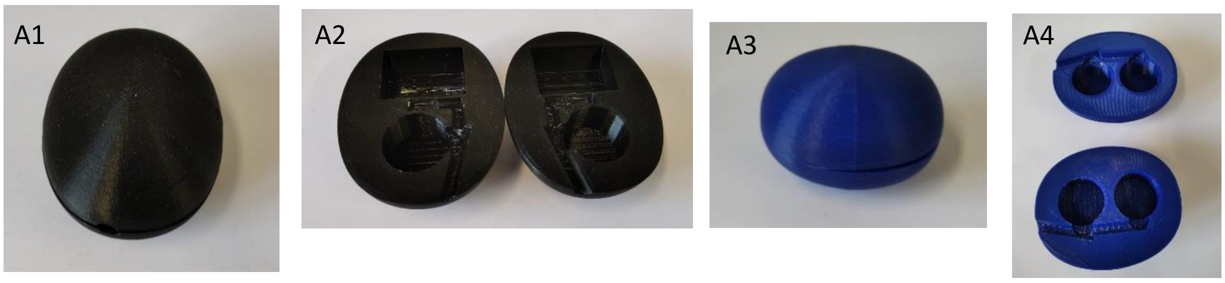


Fig. 16 Different casings for the actuators. A1, A2 was intended for a 2DoF device, while A3, A4 was intended to increase the strength of the sensation by using 2 actuators.

Another solution proposed the use of a surface speaker [31], see Fig. 17, in combination with a DIY handle made from a threaded rod and mouldable plastic, check Fig. 18. This configuration does allow for a perception of the pulling sensation, and is somewhat stronger than the one produced by the voice coils. It even allows for a 2 or 2.5 DoF of vibration, not reaching 3DoF since one of the directions would have to oppose gravity, which is a much stronger force than the one induced by the device. However, it has the disadvantage of being too cumbersome, in the sense that it weights considerably more than a voice coil, and this weight is offset to one side, causing an unwanted torque in the hand when being held.

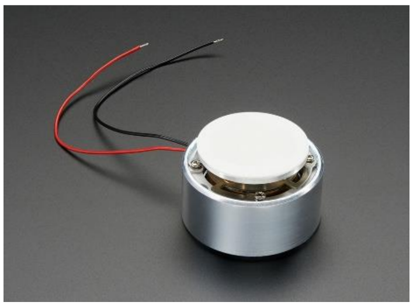


Fig. 17 Surface speaker [17]

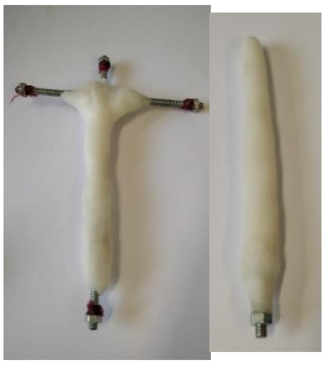


Fig. 18 Two different DIY handles

Vibration Band

The assembly the device (Fig. 19) is based on one of the designs shown in [7], where some eccentric mass motors around the user's wrist inside a streachable fabric. Nonetheless, there are several changes made to that design in order to adapt it for what, from now on will be related to as "Vibration Band".

Firstly, the actuators were exchanged for LRAs instead of ERM, since they don't have a ramp up and ramp down of intensity of vibration, thus allowing for a faster response time. The number of actutators was reduced to 2 the minimum needed for 1DoF guiding device. The positions of the actuators were also changed to be at each of the sides of the arm (Fig. 19). This was a decision made to facillitate the future user study. GuidingHaptics requires the fingers holding the actuator to be perpendicular to the wrist and parallel to the ground, this position forces ulna bone to be facing the ground while the radius faces up. If the actuators where instead placed at each sides of the wrist, the vibration would be felt on the top side or bottom side when the arm is in the position described previously, and this would not be a reasonable way of comparing the different devices in the user study, as the vibrations top or bottom would have to be interpreted as left or right, instead of left or right directly. With the modified position both the Guiding Haptics device, and the Vibration Band would produce a sensation to the left or right, without requiring the subject to further interpret the stimuli.

The driver for both of the LRAs was the same system used for Guiding Haptics, only the code running on the Arduino Nano was modified to generate a sinusoidal signal and allow individual control of the LRAs.

The velcro strip was necessary to ensure good contact between the actuators and the subject's skin, as if they were to be hanging loosly the haptic vibration might not be perceptible.

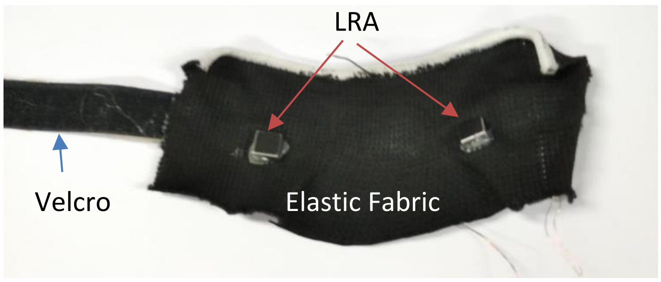


Fig. 18 Vibration band device.

Audio Device

This device uses a pair of over-the-ear stereo headphones to provide guiding cues to the subject. To do so, they make a "beep" sound in subject's ears. If the user has to be guided towards the right, the sound cue will only be produced in the right ear. On the other hand, if the user must go towards the left, then the stimuli will be produced in the left ear.

One of the reasons this method was selected is because it causes less mental workload when compared to other audio based cues like spoken voice commands [32]. Another justification for the use of headphones comes from the fact that the user will be moving left or right when following the guiding cues. The use of headphones ensures that the user will always have the same reference frame for left and right, since the headphones will always be on the same postion on their head. If instead, a pair of stereo spekears were to be used, the frame of reference of the user would not remain constant, as the user would move with respect to the speakers.

The software implementation was done in Python, making use of the Pynput library [33]. It was directly integrated with the code used in the implementation of the user study.

User Study

The goal of this user study is to discover if the use of asymmetrically vibrating guiding devices brings any benefits regarding to the mental workload experienced by the subjects during the guiding experiment, when compared to other haptic or audio based guiding devices.

To do so the subjects will have to follow the guiding cues of the different devices until they reach different targets distributed in an alternating manner along a 1 dimensional line. While paying attention to the guidance, users will also have to perform different secondary tasks that aim to increase the user's mental workload.

The time to complete the different tasks (TCT), all the positions of the users during the whole study, the accuracy, and the reaction time in the secondary tasks will be tracked in order to have enough empirical data to identify possibly significant differences.

The users will also have to fill in a NASA-TLX [34] questionnaire in order to gather subjective data about their experience with the devices. The users will also have to score the devices from 1 to 10 and rank them in order of preference. There will also be an open comments section.

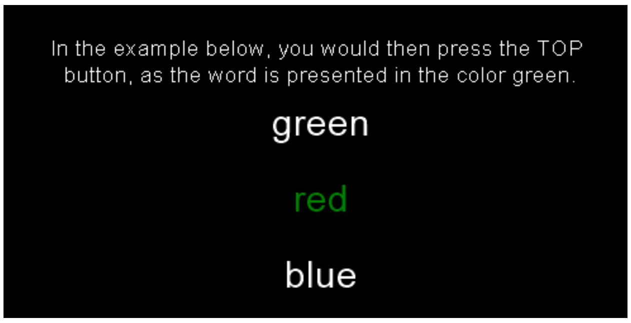
Conditions

The user study consists of 3 conditions, guidance with GuidingHaptics, the vibration band and the audio device, referred to as Sound Cues in the following figures. There is no downtime between changing from one condition to the following, since the subjects will be required to wear or hold all the devices simultaneously, even though only one of them will be providing stimuli during each condition. Before starting the experiment, the participants received instructions detailing the different conditions and tasks that they will have to perform. The conditions were counterbalanced using a Latin square to avoid order effects.

Trials

Each of the conditions will have 3 tasks associated with it, the trials will always be in the same order, it being the following:

1. No secondary task: the subject will just have to follow the guiding cues.
2. Stroop task: while being guided, the user will also have to perform a secondary task based on the Stroop effect [14]. There will be two options to select between for the color in which the letter of the word presented is written (Fig. 19)
3. Math task: different simple mathematical operations will be presented simultaneously with the guiding cues. The subject will attempt to select the correct answer from the two options presented (Fig. 20).

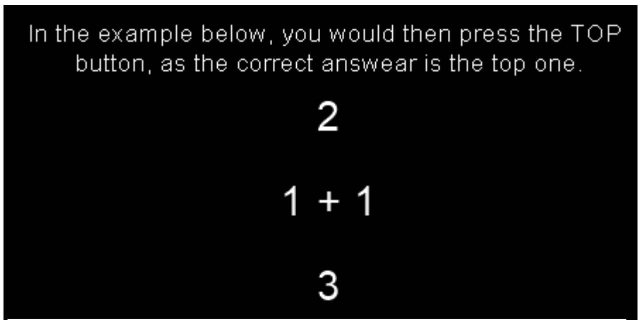

In the example below, you would then press the TOP button, as the word is presented in the color green.

green

red

blue

Fig. 19 Stroop task example.


In the example below, you would then press the TOP button, as the correct answear is the top one.

2

1 + 1

3

Fig. 20 Math task example.

Implementation

The implementation of this user study required a system to track the positions in 3D space of the participants. For this, we used an Optitrack V120:Duo [35] which allows a 6DoF tracking of an IR reflective marker positioned on the actuator used for the Guiding Haptics device (Fig. 23).

Fig. 21 OptiTrack system.

Fig. 22 Reflective marker.

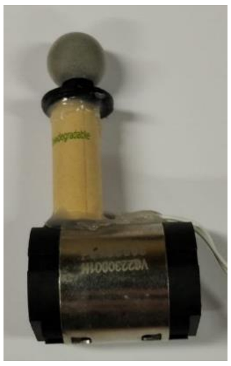


Fig. 23 GuidingHaptics device with reflective marker on top of a carboard tube.

The code for the user study was entirely implemented with Python. It makes use of several different libraries:

- Python-NatNet [36]: allows for the communication between the Python and the Motive software [37] necessary to operate the OptiTrack system.
- Pygame [38]: used to control the beeping sounds for the Audio device.
- Pynput [33]: handles button presses of the user as well as comunication between the main program from the user study and the one for the secondary tasks.
- Pyserial [39]: required for the communication between Python and the microcontrollers responsible for generating the signals for the actuators of the GuidingHaptics and Vibration Band devices.
- Psychopy [40]: was the library used in the program responsible for the stroop and math secondary tasks.

The code for the guidance control and data collection for the different conditions was implemememented from scratch. The 3D tracking runs parallel to the main program,on a separate thread, and updated a global variable with the current position of the user. The main program controlls different threads that send messages, through serial, to the microcontrollers responsible for signal generation. This way we can control the guiding cues to allow the user to advance through the fixed sets of markers. As the user reaches the target with their hand, the program will automatically change the guiding cue to guide the participant to the next target in the set.

The program also handles the emulation of keypresses necessary for the communication with the program responsible for handling the secondary tasks. This is necessary as it was not possible to integrate this program in the main one as *psychopy*, the library used in its implementation, does not allow for it to be run on a thread. Also, to select the correct answer in the secondary tasks, the users have to press a button (Fig. 24), which is done by using another Arduino Uno to send a serial message to the main program when one of the buttons of the remote is pressed.

On the other hand, for the secondary tasks, we used the code from Stroopy [41] as a foundation, and iterated over it to add the math task, modify the stroop task, and implement automated control for starting and stopping the different trials for each condition.

All the data gathered from the users is stored into CSV files to simplify its future analysis.

The figures below depict the whole setup used to conduct the user study.

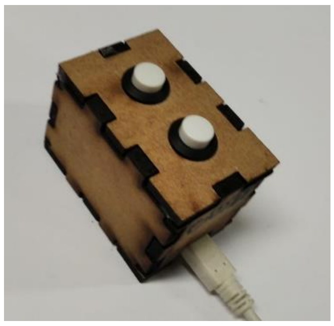


Fig. 24 Remote for selecting the secondary task’s answers.

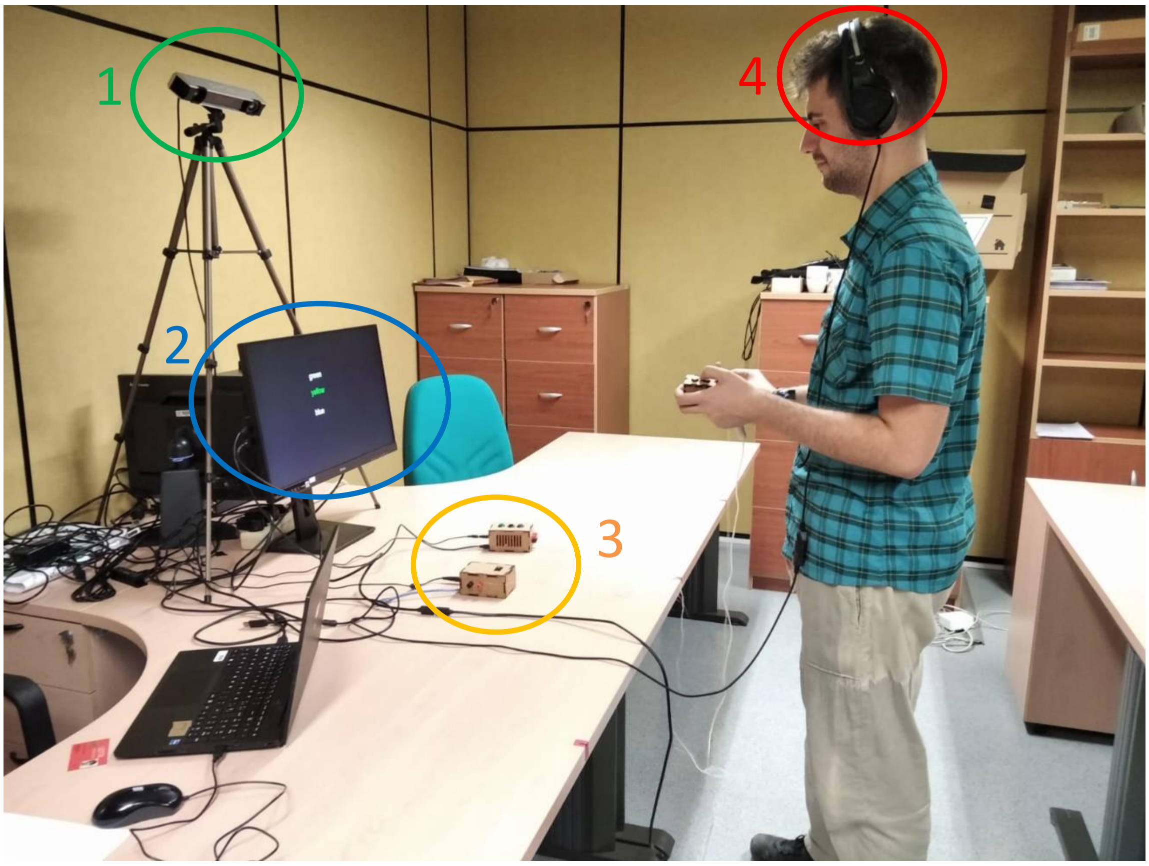


Fig. 25 Setup for the user study. 1- OptiTrack system. 2- Screen where the secondary task is shown. 3- Drivers for the guiding devices. 4- Headphones for the sound guiding device

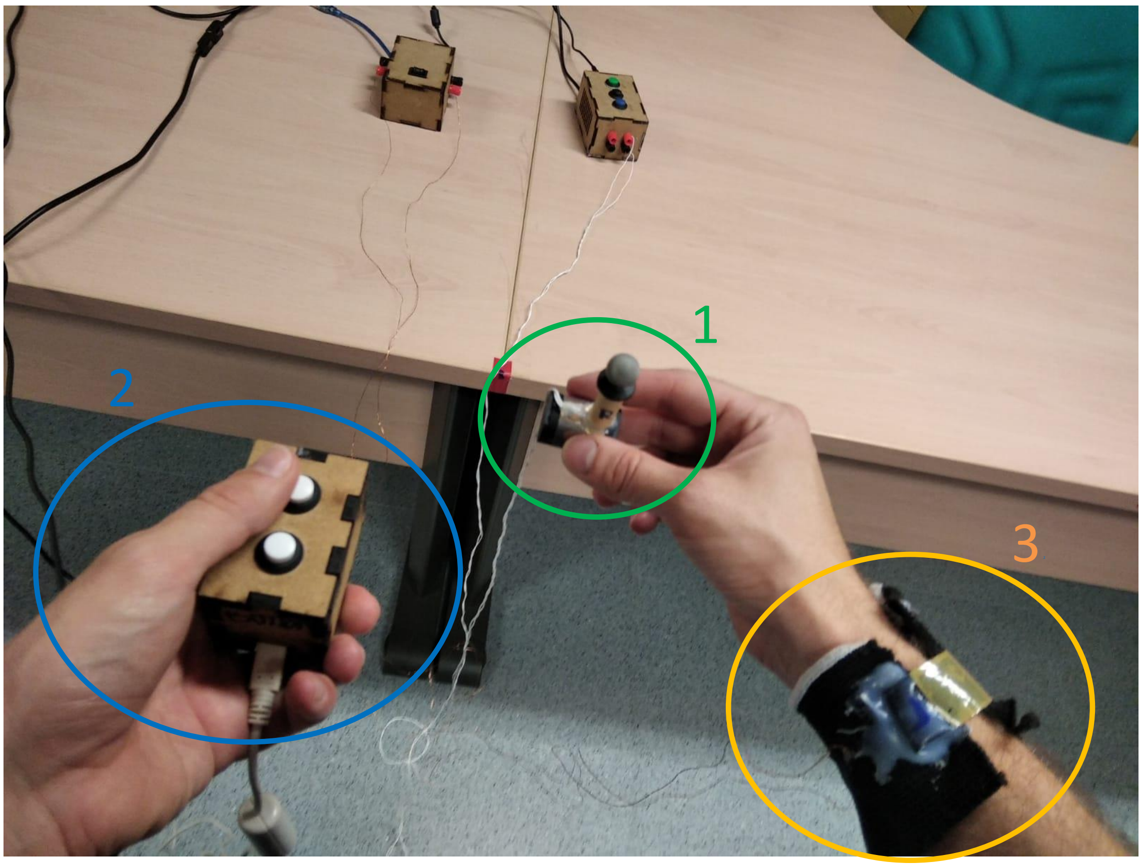


Fig. 26 The different devices on the user's hands and arm. 1- GudingHaptics device. 2- Remote to select answers. 3- Vibration band device.

Results and Analysis

The main and secondary tasks CSV files, for each user, generated during the user study had to be preprocessed to remove faulty or erroneous data. To do so, the *pandas* [42] python library was used to load the CSV files one by one. Once loaded, the timestamps for the ends of each trial were compared on both the CSV file for the main task, and the one for the secondary task. For each trial, if the secondary task file has timestamps later than that of the first file, they are removed.

Some files, the ones from user 1 and 13, had to be discarded since for user 1 the math task did not display properly, and for user number 13 the data was not recorded properly and was rendered unsalvageable.

Proceeding with the data processing, the corrected CSVs for each user are loaded one by one. For each device and trial the TCT, the reaction time and inefficiency of the main task, and the timestamps used for the calculations are stored into a python list to avoid having to load the files again. The first six targets in all trials are not included in the calculation of the statistics as they are considered to be learning trials for the user to get used to the device and readjustment trials between interruptions when changing secondary tasks. An analysis of the variance was later performed on the data.

As for the subjective data about the user's mental workload and experience, the data of the questionnaires was collected and the users with invalid data, 1 and 13, were also discarded. The data was later analyzed using analysis of variance.

Figures with the movements of the users and the reaction times for each target were also plotted, that way corrections to the reaction times could be manually done if there are errors in its calculation caused by occlusions in the tracking systems or the user confusing the guiding cues. See the figures below for some examples.

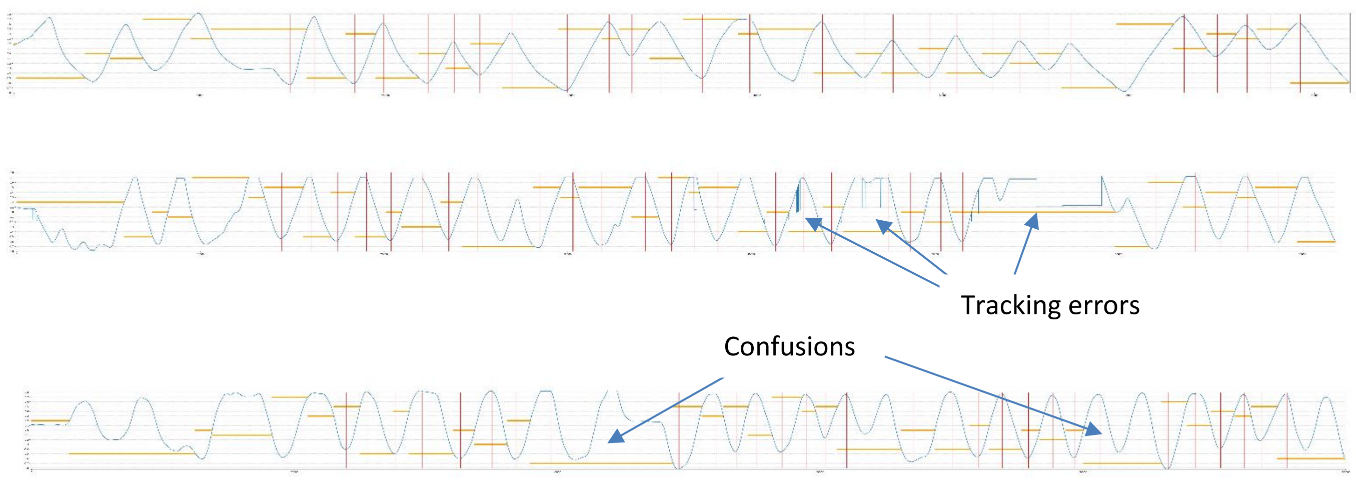
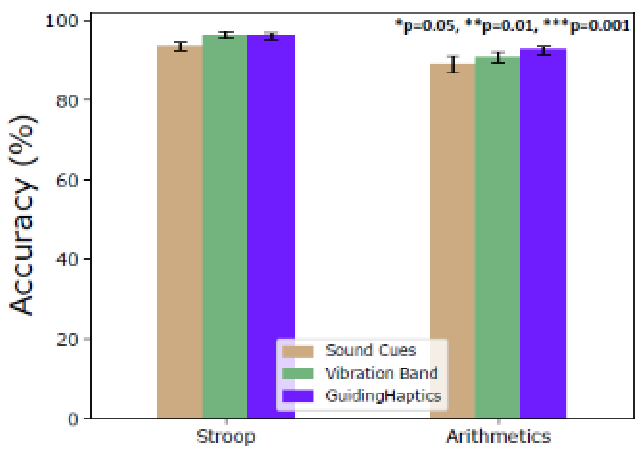
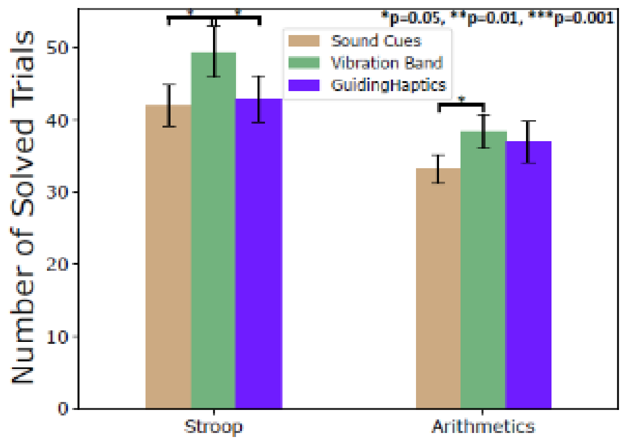
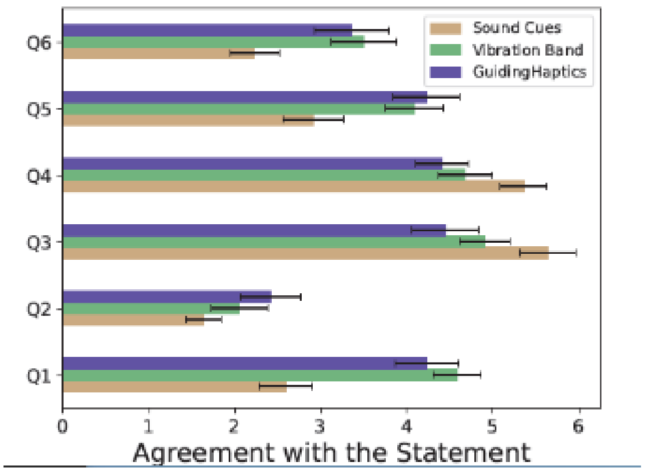


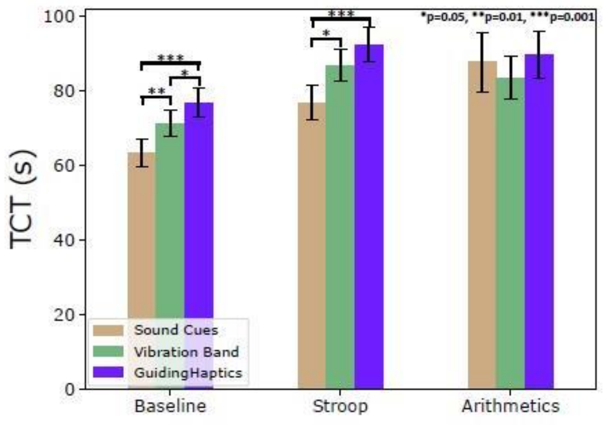
Fig. 27 Plots of the user's movement (blue), reaction times (red) and targets (orange). Top plot shows a tracking with no major occlusions or confusions. Middle shows a plot with major tracking errors and bottom one depicts a plot with confusions of the user.

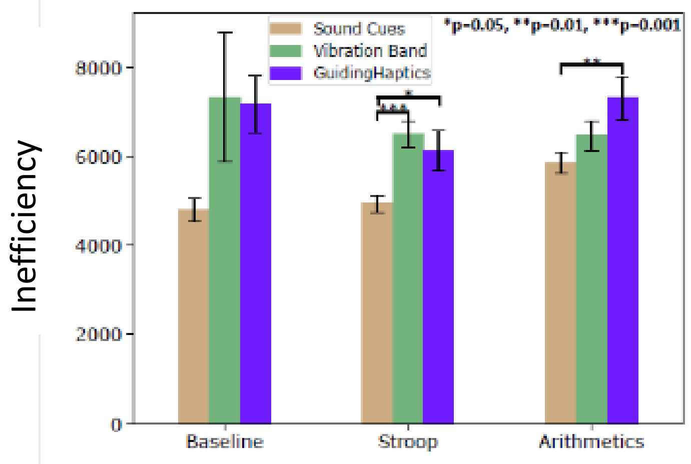
The figures below show plots statistics of the empirical measurements as well as the subjective experiences of the users.

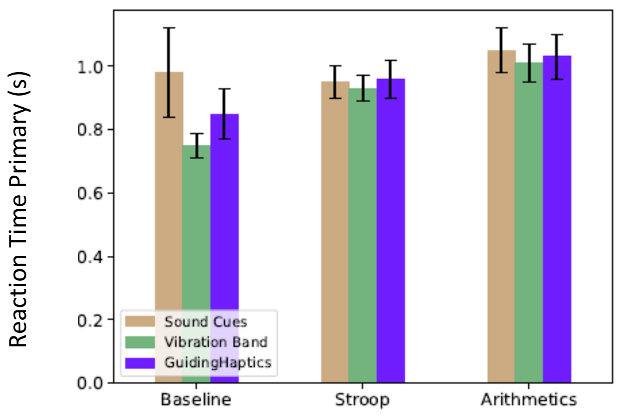












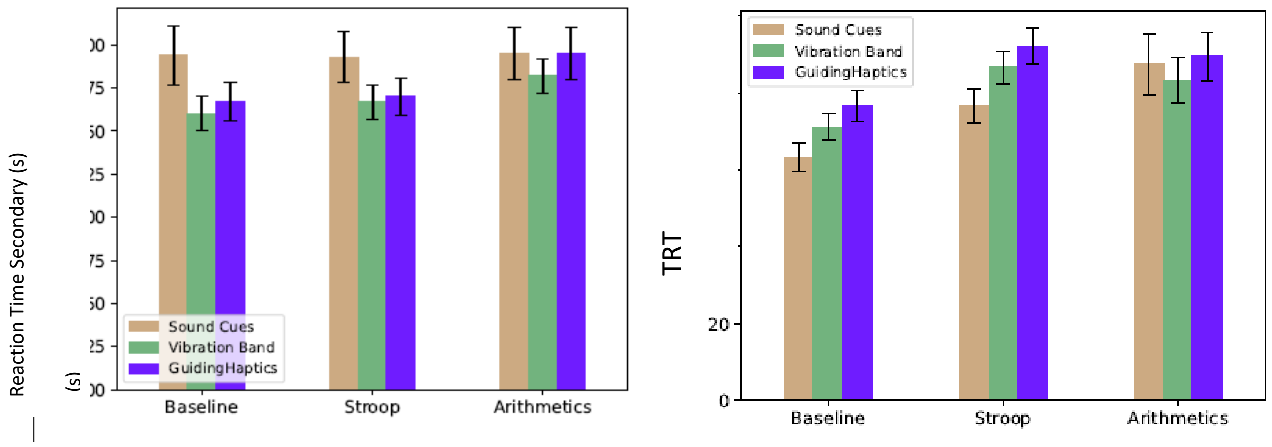


Fig. 28 Plots with the different statistics calculated.

From this analysis and plots we can draw the several conclusions

The sound based guiding device allows for a significantly faster completion time for guiding tasks, while the vibration based devices cause an increase in the completion time by a significant amount, being GuidingHaptics the slowest method. Reasons for this could be the familiarity the users have with the devices. Nonetheless, when performing a sencondary task, there are no significant differences in completion time, possibly meaning that, when it comes to guidance with more realistic conditions, the devices perform similarly. No significant differences in the math task could also suggest it was inappropriate for the secondary to the guidance task, because maybe it interfered or caused too much mental workload.

GuidingHaptics and vibration band cause significant inefficiencies in the users movement when compared to the audio device, resulting in a longer distance traveled by the user to complete the target course. However there are no significant differences in efficiency when comparing both of the vibration based devices. This may indicate that the sound based device produces more accurate cues, or that it's stimuli are processed faster by the user.

The vibration band device allows for a significant increase in the amout of secondary task trials solved, when compared to the other devices. The cuase of this could be the device causing less mental workload than the other ones, making the users focus more on the secondary task.

Finally a list with the most recurrent comments of the open comments section.

GuidingHaptics (14 users commented):

- Hard to know where it’s guiding (4)
- At the beginning it’s hard to know the correct direction (3)
- One direction felt clearer than the other (3)
- Guided by the change of the type of vibration or different sound it made, not by the change of direction (3)

- Intuitive (2)

Vibration Band (17 users commented):

- One direction felt clearer than the other (5)
- Hard to know where it’s guiding (3)
- The hardest one [compared to the other conditions] (2)
- Best guidance [compared to the other conditions] (2)

SoundCues (9 users commented):

- The best one to be guided with [compared to the other two conditions] (5)
- Easy to follow [the guiding] (3)
- The least comfortable [of all conditions] (2)

Bibliography

[1] T. Tanabe, H. Endo, and S. Ino, “Effects of Asymmetric Vibration Frequency on Pulling Illusions,” *Sensors*, vol. 20, no. 24, p. 7086, Dec. 2020, doi: 10.3390/s20247086.

[2] T. Tanabe, H. Yano, and H. Iwata, “Evaluation of the Perceptual Characteristics of a Force Induced by Asymmetric Vibrations,” *IEEE Trans. Haptics*, vol. 11, no. 2, pp. 220–231, Apr. 2018, doi: 10.1109/TOH.2017.2743717.

[3] J.-P. Choiniere and C. Gosselin, “Development and Experimental Validation of a Haptic Compass Based on Asymmetric Torque Stimuli,” *IEEE Trans. Haptics*, vol. 10, no. 1, pp. 29–39, Jan. 2017, doi: 10.1109/TOH.2016.2580144.

[4] K. Bark *et al.*, “Effects of Vibrotactile Feedback on Human Learning of Arm Motions,” *IEEE Trans. Neural Syst. Rehabil. Eng. Publ. IEEE Eng. Med. Biol. Soc.*, vol. 23, no. 1, pp. 51–63, Jan. 2015, doi: 10.1109/TNSRE.2014.2327229.

[5] H. Elsayed, M. Weigel, J. Semsch, M. Mühlhäuser, and M. Schmitz, “Tactile Vectors for Omnidirectional Arm Guidance,” in *Augmented Humans Conference*, Glasgow United Kingdom: ACM, Mar. 2023, pp. 35–45. doi: 10.1145/3582700.3582701.

[6] J. V. Salazar Luces, K. Okabe, Y. Murao, and Y. Hirata, “A Phantom-Sensation Based Paradigm for Continuous Vibrotactile Wrist Guidance in Two-Dimensional Space,” *IEEE Robot. Autom. Lett.*, vol. 3, no. 1, pp. 163–170, Jan. 2018, doi: 10.1109/LRA.2017.2737480.

[7] A. A. Stanley and K. J. Kuchenbecker, “Evaluation of Tactile Feedback Methods for Wrist Rotation Guidance,” *IEEE Trans. Haptics*, vol. 5, no. 3, pp. 240–251, 2012, doi: 10.1109/TOH.2012.33.

[8] T. Amemiya and H. Gomi, “Distinct Pseudo-Attraction Force Sensation by a Thumb-Sized Vibrator that Oscillates Asymmetrically,” in *Haptics: Neuroscience, Devices, Modeling, and Applications*, M. Auvray and C. Duriez, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 88–95. Accessed: Nov. 23, 2022. [Online]. Available: http://link.springer.com/10.1007/978-3-662-44196-1_12

[9] H. Kim, H. Yi, H. Lee, and W. Lee, “HapCube: A Wearable Tactile Device to Provide Tangential and Normal Pseudo-Force Feedback on a Fingertip,” in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, Montreal QC Canada: ACM, Apr. 2018, pp. 1–13. doi: 10.1145/3173574.3174075.

[10] B. Cain, “A Review of the Mental Workload Literature,” *English*, p. 35, Jul. 2007.

[11] L. L. Di Stasi, A. Antolí, M. Gea, and J. J. Cañas, “A neuroergonomic approach to evaluating mental workload in hypermedia interactions,” *Int. J. Ind. Ergon.*, vol. 41, no. 3, pp. 298–304, May 2011, doi: 10.1016/j.ergon.2011.02.008.

[12] L. Longo, “Experienced mental workload, perception of usability, their interaction and impact on task performance,” *PLOS ONE*, vol. 13, no. 8, p. e0199661, Aug. 2018, doi: 10.1371/journal.pone.0199661.

[13] X. Wu and Z. Li, “Secondary Task Method for Workload Measurement in Alarm Monitoring and Identification Tasks,” Jul. 2013, pp. 346–354. doi: 10.1007/978-3-642-39143-9_39.

[14] “Stroop effect,” *Wikipedia*. Jun. 27, 2023. Accessed: Sep. 09, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Stroop_effect&oldid=1162193994

[15] T. Amemiya, H. Ando, T. Maeda, T. Amemiya, H. Ando, and T. Maeda, “Kinesthetic Illusion of Being Pulled Sensation Enables Haptic Navigation for Broad Social Applications,” in *Advances in Haptics*, IntechOpen, 2010. doi: 10.5772/8701.

[16] J. Rekimoto, “Traxion: a tactile interaction device with virtual force sensation,” in *Proceedings of the 26th annual ACM symposium on User interface software and*

technology, St. Andrews Scotland, United Kingdom: ACM, Oct. 2013, pp. 427–432. doi: 10.1145/2501988.2502044.

[17] T. Tanabe, H. Yano, and H. Iwata, “Properties of proprioceptive sensation with a vibration speaker-type non-grounded haptic interface,” in *2016 IEEE Haptics Symposium (HAPTICS)*, Apr. 2016, pp. 21–26. doi: 10.1109/HAPTICS.2016.7463150.

[18] H. Culbertson, Julie M. Walker, J. Walker, M. Raitor, and A. M. Okamura, “WAVES: A Wearable Asymmetric Vibration Excitation System for Presenting Three-Dimensional Translation and Rotation Cues,” pp. 4972–4982, May 2017, doi: 10.1145/3025453.3025741.

[19] N. Sabnis, “Pseudo forces from asymmetric vibrations can provide movement guidance,” 2021, Accessed: Sep. 05, 2023. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid%3A88e0f900-1682-4ac8-8583-0a9c013a6380

[20] “Arbitrary Function Generators.” https://www.tek.com/en/datasheet/afg310-afg320 (accessed Sep. 05, 2023).

[21] “Arduino Nano,” *Arduino Official Store*. https://store.arduino.cc/products/arduino-nano (accessed Sep. 05, 2023).

[22] “OcioDual Controlador L298N Motores DC PAP Driver Stepper Doble puente H para Electrónica Robótica Proyectos Raspberry PIC AVR : Amazon.es: Industria, empresas y ciencia.” https://www.amazon.es/OcioDual-Controlador-Motores-Driver-Stepper/dp/B07YNR5KWP/ref=asc_df_B07YNR5KWP/?tag=googshopes-21&linkCode=df0&hvadid=628580889748&hvpos=&hvnetw=g&hvrand=6753073358422671351&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=1005503&hvtargid=pla-1875528526707&psc=1 (accessed Sep. 05, 2023).

[23] “InigoEzcurdia/WavePainter,” *GitHub*. https://github.com/InigoEzcurdia/WavePainter (accessed Sep. 05, 2023).

[24] “VLV200634A - 160hz Rectagular LRA Linear Vibration Motor,” *Vybronics*, Jun. 27, 2022. https://www.vybronics.com/linear-lra-vibration-motors/v-lv200634a (accessed Sep. 05, 2023).

[25] “VL32158H-L25 - Generates 5G @ 100hz LRA Linear Vibration,” *Vybronics*, Dec. 10, 2021. https://www.vybronics.com/linear-lra-vibration-motors/v-l32158h-l25 (accessed Sep. 05, 2023).

[26] “Solenoid Motor Vibration VG2230001H,” *Vybronics*, Feb. 24, 2022. https://www.vybronics.com/coin-vibration-motors/lra/v-g2230001h (accessed Sep. 05, 2023).

[27] “LRA Coin Vibration Motor - VG1040003D,” *Vybronics*, May 06, 2019. https://www.vybronics.com/coin-vibration-motors/lra/v-g1040003d (accessed Sep. 05, 2023).

[28] “Red Pitaya - Swiss Army Knife For Engineers,” May 03, 2021. https://redpitaya.com/ (accessed Sep. 05, 2023).

[29] “HiLetgo 2pcs ADXL335 3-Axis Accelerometer Angular Transducer Sensor GY-61 Accelerometer Sensor Angular Transducer Sensor Analog Output 3-5V for Arduino : Amazon.es: Industria, empresas y ciencia.” https://www.amazon.es/HiLetgo-ADXL335-Accelerometer-Angular-Transducer/dp/B081YTDWXS/ref=sr_1_1?crid=2ORKWJO7ZZQ1B&keywords=accelerometer+analog+output&qid=1673614029&sprefix=accelerometer+analog+output%2Caps%2C860&sr=8-1 (accessed Sep. 05, 2023).

[30] “Polydoh plástico moldeable + 6 libre paquetes de colorear gránulos, plástico, 500g (también conocido como polimorph, plastimake o instamorph) : Amazon.es: Hogar y cocina.” https://www.amazon.es/pl%C3%A1stico-moldeable-polimorph-plastimake-instamorph/dp/B01MZE4LYK?th=1 (accessed Sep. 06, 2023).

[31] A. Industries, “Large Surface Transducer with Wires - 4 Ohm 5 Watt.” https://www.adafruit.com/product/1784 (accessed Sep. 06, 2023).
[32] S. Holland, D. Morse, and H. Gedenryd, “AudioGPS: Spatial Audio Navigation with a Minimal Attention Interface,” *Pers. Ubiquitous Comput.*, vol. 6, Sep. 2002, doi: 10.1007/s007790200025.
[33] “pynput Package Documentation — pynput 1.7.6 documentation.” https://pynput.readthedocs.io/en/latest/ (accessed Sep. 08, 2023).
[34] “NASA-TLX,” *Wikipedia*. Jul. 22, 2023. Accessed: Sep. 09, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=NASA-TLX&oldid=1166550915
[35] “V120:Duo - An optical tracking system in a single, plug-and-play package,” *OptiTrack*. http://optitrack.com/cameras/v120-duo/index.html (accessed Sep. 09, 2023).
[36] M. Edwards, “Overview.” Jun. 22, 2023. Accessed: Sep. 09, 2023. [Online]. Available: https://github.com/mje-nz/python_natnet
[37] “Motive - In Depth,” *OptiTrack*. http://optitrack.com/software/motive/index.html (accessed Sep. 09, 2023).
[38] “GettingStarted - pygame wiki.” https://www.pygame.org/wiki/GettingStarted (accessed Sep. 10, 2023).
[39] “pyserial: Python Serial Port Extension.” Accessed: Sep. 10, 2023. [MacOS :: MacOS X, Microsoft :: Windows, POSIX]. Available: https://github.com/pyserial/pyserial
[40] “Home — PsychoPy®.” https://www.psychopy.org/ (accessed Sep. 10, 2023).
[41] Erik, “Stroopy.” Jul. 19, 2023. Accessed: Sep. 10, 2023. [Online]. Available: https://github.com/marsja/stroopy
[42] “pandas - Python Data Analysis Library.” https://pandas.pydata.org/ (accessed Sep. 12, 2023).

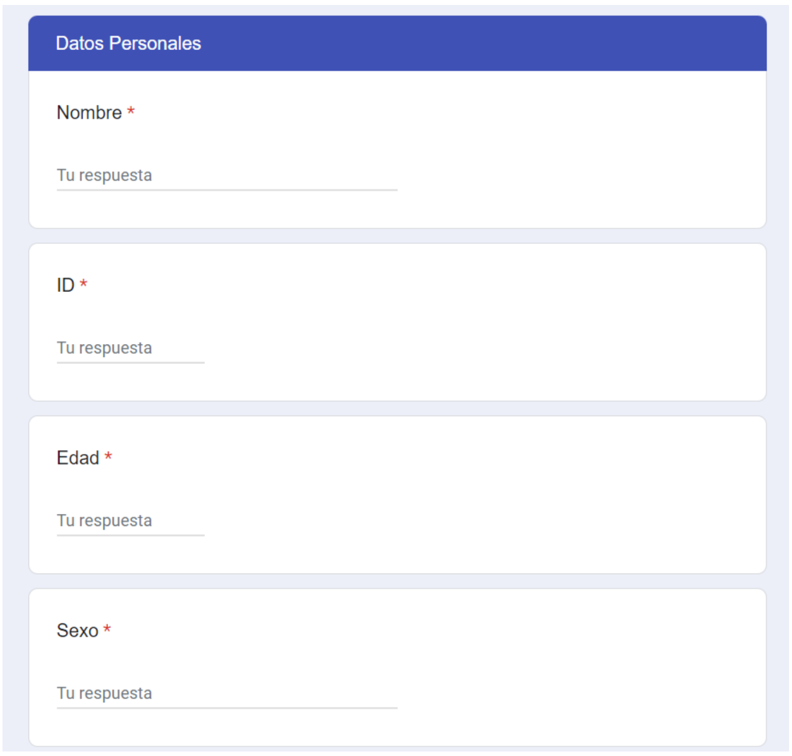
Annex

Arduino code for signal generation

```
#include <avr/sleep.h>
#include <avr/power.h>
#define N_SAMPLES 256
#define HALF_SAMPLES (N_SAMPLES/2)
#define MAX_VAL 80 //this is in uS, it is the period of the PWM freq minus some time for calculations
#define HALF_VAL (MAX_VAL/2)
#define PWM_FR 20000
#define N_SIGNALS 6
//square, sin, triang, sawRise, sawFall, custom
uint8_t signals[N_SIGNALS][N_SAMPLES]; //each sample is: 2 bits port masks | 6 bits microswait (0 to 63 at most)

inline uint8_t packSample(uint8_t val) {
  if (val >= HALF_VAL){
    return 0b01000000 | (val-HALF_VAL);
  }else{
    return 0b10000000 | (HALF_VAL-val);
  }
}

//wave vlues obtained with wavepainter (the wave looks like this --> ‾\)
const uint8_t cappedSawtooth[] =
{98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98
,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98
,98,98,98,98,98,98,98,98,98,98,98,98,98,98,98,99,99,98,96,95,94,94,93,92,91,91,91,90,90,88,87,86,86,86,85,83,84,83,82,81,81,80
,80,79,78,78,78,77,76,76,75,75,74,73,73,71,72,71,70,69,69,68,67,67,67,66,66,65,64,64,63,62,63,61,61,60,60,60,58,57,56,57,55,54
,54,53,53,51,51,50,50,50,49,48,48,47,47,46,45,45,44,44,43,42,42,42,41,40,39,39,38,38,37,38,37,36,36,35,35,34,34,33,32,32,31,31
,31,29,29,28,29,27,27,26,26,24,24,23,22,22,22,20,20,18,18,18,16,15,15,14,14,13,12,11,12,10,10,9,8,8,7,6,6,5,5,4,3,3,3,2,2,2,2,2};


void fillInSignals() {
  for (int i = 0; i < N_SAMPLES; ++i) {
    signals[0][i] = packSample( i < HALF_SAMPLES ? MAX_VAL : 0 ); //square wave
    const double v = cos( i * 2.0 * PI / N_SAMPLES ); //sinusoidal
    signals[1][i] = packSample( (byte) ((v + 1.0) / 2.0 * MAX_VAL) ); //map from -1,1  to  0,MAX
    signals[2][i] = packSample( abs( i*MAX_VAL/N_SAMPLES*2 - MAX_VAL ) ); //triangular
    //signals[3][i] = packSample( i * MAX_VAL / N_SAMPLES ); //saw-tooth rise
    //signals[4][i] = packSample( (N_SAMPLES-i) * MAX_VAL / N_SAMPLES ); //saw-tooth fall
    signals[3][i] = packSample( cappedSawtooth[i] * MAX_VAL / 100); // ‾\ wave
    signals[4][i] = packSample( (100-cappedSawtooth[i]) * MAX_VAL / 100); //  /‾  wave
    signals[5][i] = packSample( 0 ); //custom starts empty
  }
}

void setup() {
  fillInSignals();

  DDRC = 0b00000011; //set pins A0 to A1 as outputs
  PORTC = 0b00000000; //output low in all of them

  // initialize timer1 to interrupt at PWM_FR (usually 20kHz)
  noInterrupts();          // disable all interrupts
  TCCR1A = TCCR1B = 0;
  TCNT1  = 0;
  OCR1A = (F_CPU / PWM_FR);
  TCCR1B |= (1 << WGM12);   // CTC mode
  TCCR1B |= (1 << CS10);    // 1 prescaler, no prescaling
  TIMSK1 |= (1 << OCIE1A);  // enable timer compare interrupt
  interrupts();             // enable all interrupts

  // disable ADC
  ADCSRA = 0;

  // turn off everything we can
  power_adc_disable ();
  power_spi_disable();
  power_twi_disable();
  power_timer0_disable();

  //power_usart0_disable();
  Serial.begin( 57600 );

  //buttons
  pinMode( 2, INPUT_PULLUP );
  pinMode( 3, INPUT_PULLUP );
  pinMode( 4, INPUT_PULLUP );
  pinMode( 5, INPUT_PULLUP );

  /*for(int i = 0; i < N_SAMPLES; i++){
    Serial.print( signals[0][i] ); Serial.print(',');
    Serial.print( signals[1][i] ); Serial.print(',');
    Serial.print( signals[2][i] ); Serial.print(',');
    Serial.print( signals[3][i] ); Serial.print(',');
    Serial.print( signals[4][i] ); Serial.println();
  }*/
}

uint32_t indexShift24 = 0;
uint32_t indexIncShift24 = 4294967;
int currentSignal = 1;
ISR(TIMER1_COMPA_vect) { // timer compare interrupt service routine
  static uint8_t portMask = 0b00000001;
  static uint8_t microsHigh = HALF_VAL;
  PORTC = portMask; //switch on the corresponding pins
  delayMicroseconds(microsHigh);
  PORTC = 0b00000000; //all off

  //increase the index. overflow makes it cycle
  indexShift24 += indexIncShift24;

  //get sample
  const uint8_t sample = signals[currentSignal][indexShift24 >> 24];
  //extract pin mask
  portMask = sample >> 6;
  //extract wait micros
  microsHigh = sample & 0b00111111;
}

inline void stopSignalGen(){
  TIMSK1 &= ~(1 << OCIE1A); //disable the timer interrupt
}
inline void signalGen(int signalType, float fr){
  currentSignal = signalType % N_SIGNALS;
  TIMSK1 |= (1 << OCIE1A); //enable timer interrupt
  indexIncShift24 = (uint32_t) ( (float)((uint32_t)1<<24) * (float)N_SAMPLES * fr / (float)PWM_FR);
}

bool prevButtonPressed = false;
```

```
void loop() {
  //0 -> off
  //1 SIGNAL_TYPE FR -> start emitting SIGNAL_TYPE[0,1,2,3,4,5] with FR
  //2 val1 val2 ... val256 -> send a custom signal. 256 numbers from 0 to 100 are expected after the number 2
  if (Serial.available()) {
    const int command = Serial.parseInt();
    Serial.println( command );
    if (command == 0){
      stopSignalGen();
    }else if (command == 1){
      const int signalType = Serial.parseInt();
      const float fr = Serial.parseFloat();
      signalGen( signalType, fr );
    }else if (command == 2){
      stopSignalGen();
      for( int i = 0; i < N_SAMPLES; i++){
        const int val = Serial.parseInt();
        signals[5][i] = packSample( val * MAX_VAL / 100);
      }
    }else if (command == 3){
      stopSignalGen();
      Serial.read();
      for( int i = 0; i < N_SAMPLES; i++){
        const int val = Serial.read();
        signals[5][i] = packSample( val );
      }
    }

    while ( Serial.read() != '\n'); //skipt until finding the new line character
    Serial.println("9");
  }

  if (digitalRead(2) == LOW && prevButtonPressed == false){
    signalGen(3, 36.4);
    prevButtonPressed = true;
  }else if (digitalRead(3) == LOW && prevButtonPressed == false){
    signalGen(4, 36.4);
    prevButtonPressed = true;
  }else if (digitalRead(4) == LOW && prevButtonPressed == false){
    if (random(2) == 0){
      signalGen(3, 36.4);
    }else{
      signalGen(4, 36.4);
    }
    prevButtonPressed = true;
  }else if (digitalRead(5) == LOW && prevButtonPressed == false){
    stopSignalGen();
    prevButtonPressed = true;
  }else{
    prevButtonPressed = false;
  }
}
```

Questionnaire



Datos Personales

Nombre *

Tu respuesta

ID *

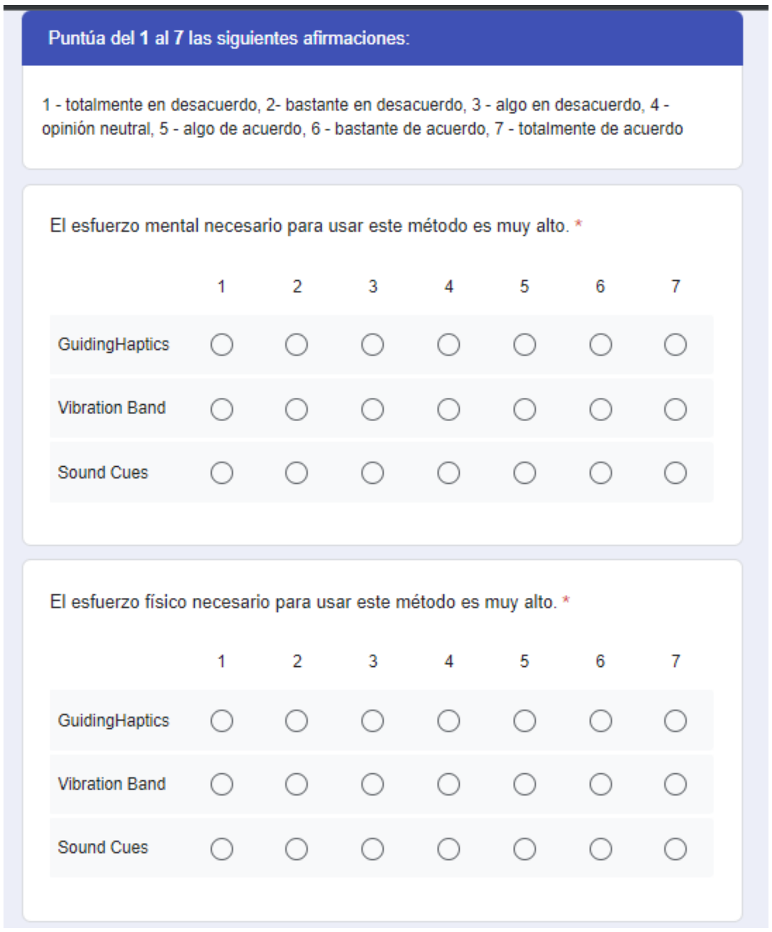
Tu respuesta

Edad *

Tu respuesta

Sexo *

Tu respuesta



Puntúa del 1 al 7 las siguientes afirmaciones:

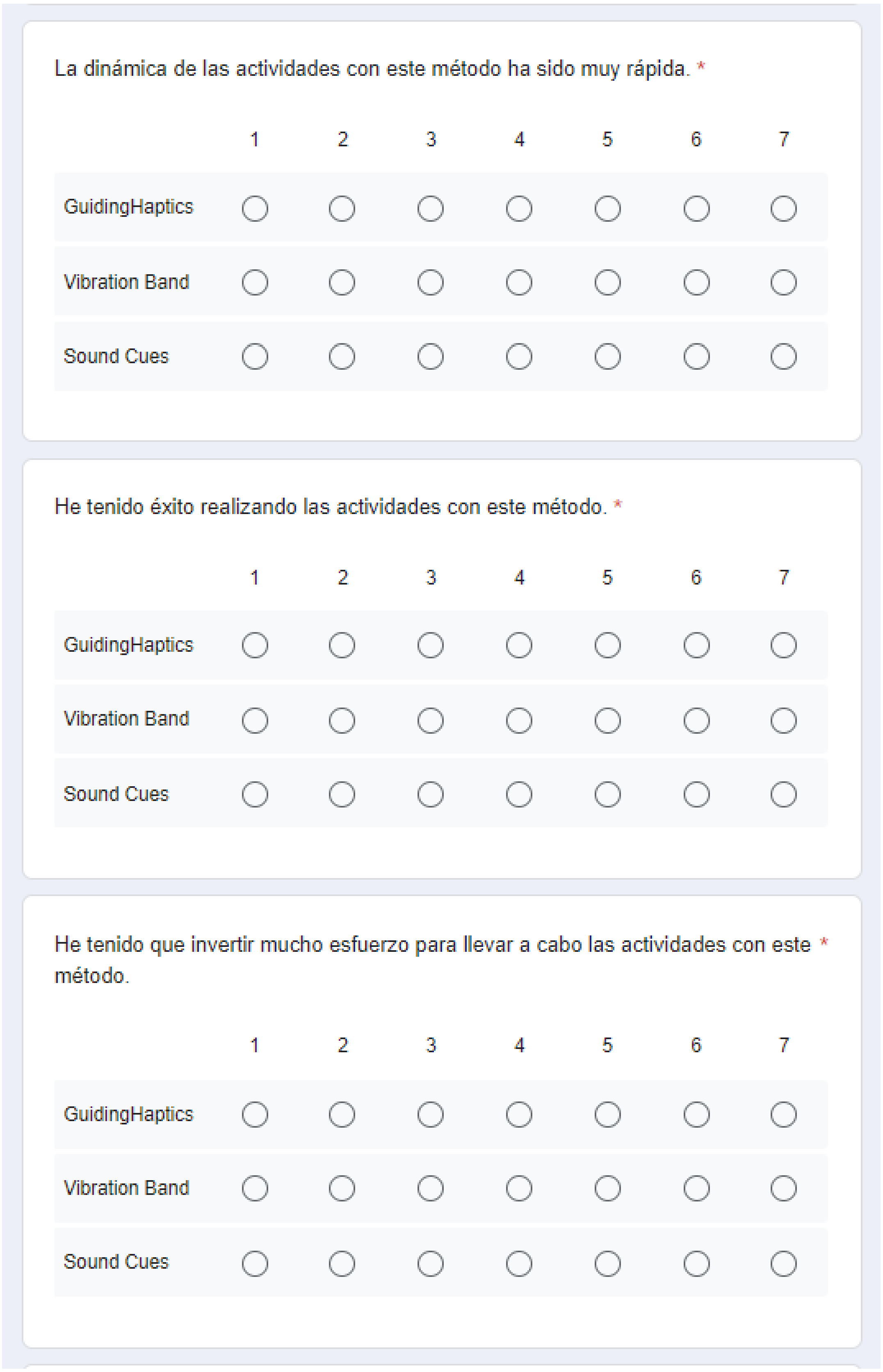
1 - totalmente en desacuerdo, 2- bastante en desacuerdo, 3 - algo en desacuerdo, 4 - opinión neutral, 5 - algo de acuerdo, 6 - bastante de acuerdo, 7 - totalmente de acuerdo

El esfuerzo mental necesario para usar este método es muy alto. *

	1	2	3	4	5	6	7
GuidingHaptics	○	○	○	○	○	○	○
Vibration Band	○	○	○	○	○	○	○
Sound Cues	○	○	○	○	○	○	○

El esfuerzo físico necesario para usar este método es muy alto. *

	1	2	3	4	5	6	7
GuidingHaptics	○	○	○	○	○	○	○
Vibration Band	○	○	○	○	○	○	○
Sound Cues	○	○	○	○	○	○	○



La dinámica de las actividades con este método ha sido muy rápida. *

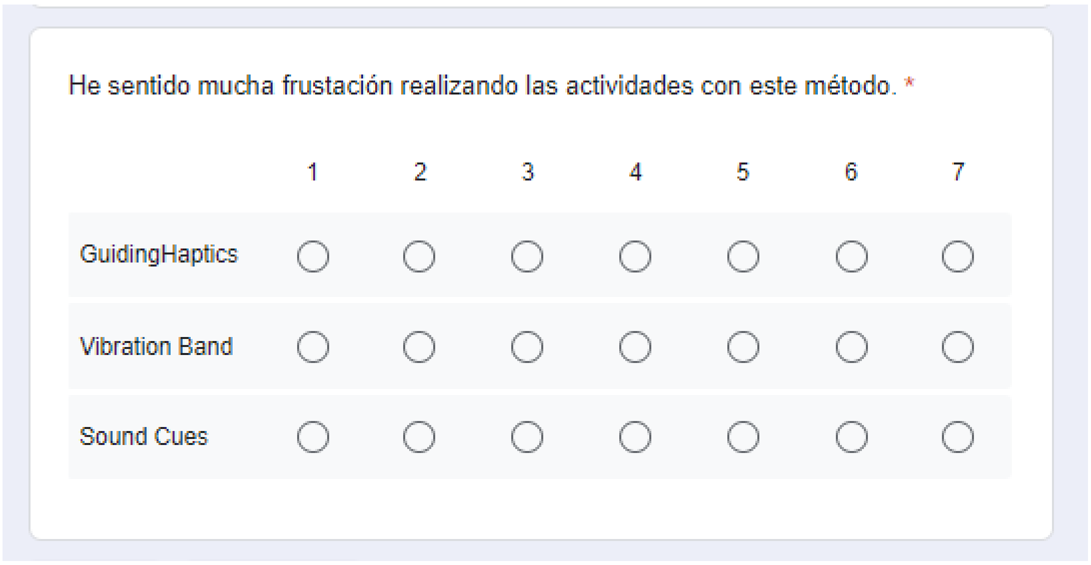
	1	2	3	4	5	6	7
GuidingHaptics	○	○	○	○	○	○	○
Vibration Band	○	○	○	○	○	○	○
Sound Cues	○	○	○	○	○	○	○

He tenido éxito realizando las actividades con este método. *

	1	2	3	4	5	6	7
GuidingHaptics	○	○	○	○	○	○	○
Vibration Band	○	○	○	○	○	○	○
Sound Cues	○	○	○	○	○	○	○

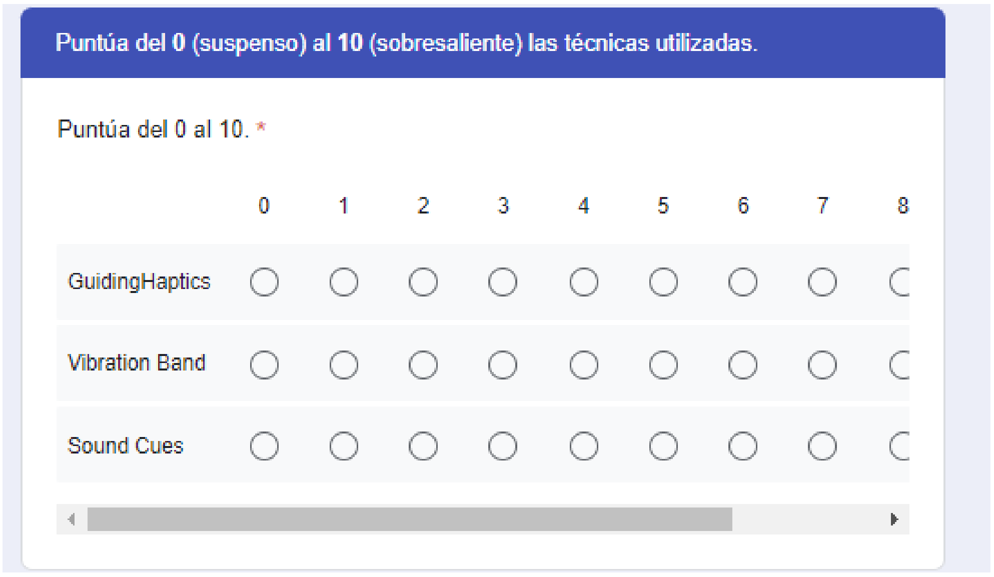
He tenido que invertir mucho esfuerzo para llevar a cabo las actividades con este método. *

	1	2	3	4	5	6	7
GuidingHaptics	○	○	○	○	○	○	○
Vibration Band	○	○	○	○	○	○	○
Sound Cues	○	○	○	○	○	○	○



He sentido mucha frustación realizando las actividades con este método. *

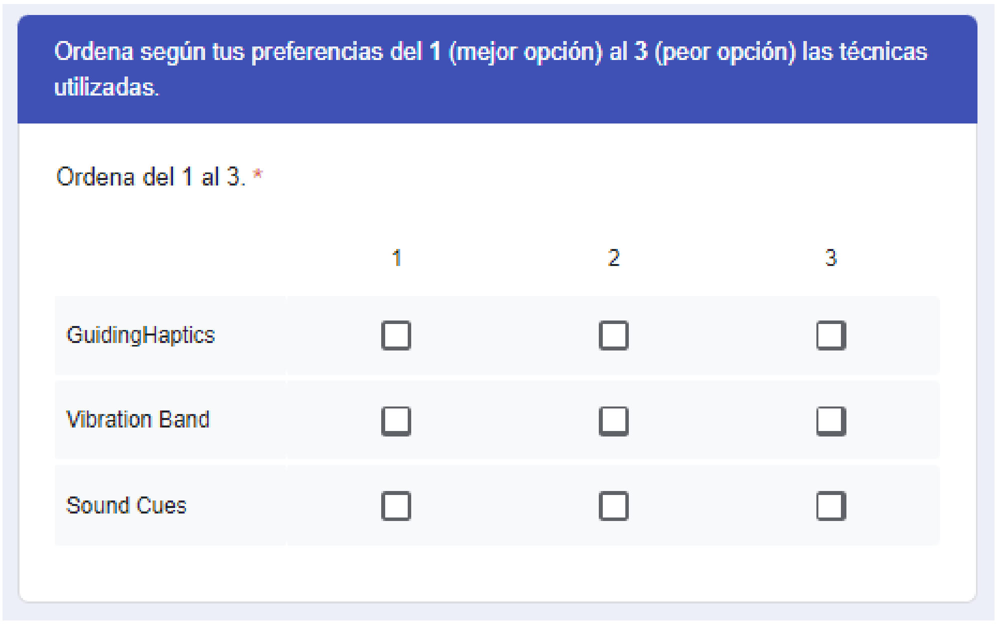
	1	2	3	4	5	6	7
GuidingHaptics	○	○	○	○	○	○	○
Vibration Band	○	○	○	○	○	○	○
Sound Cues	○	○	○	○	○	○	○



Puntúa del 0 (suspenso) al 10 (sobresaliente) las técnicas utilizadas.

Puntúa del 0 al 10. *

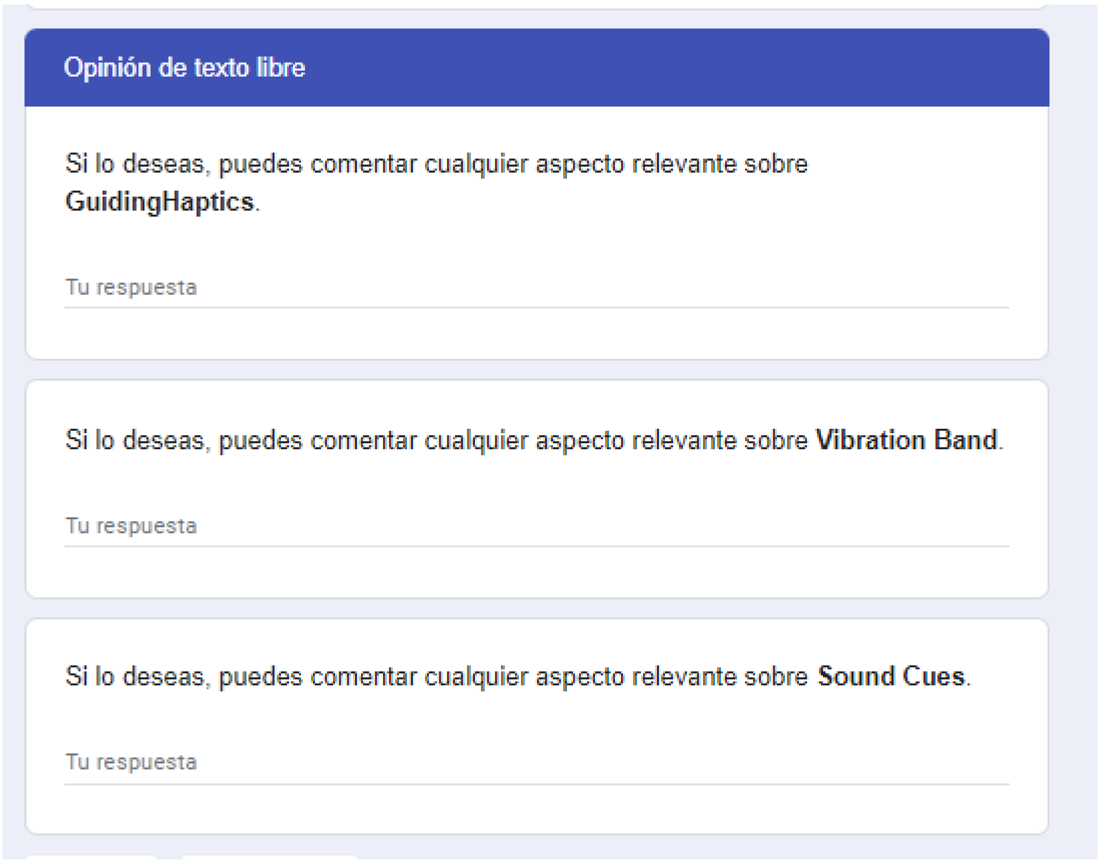
	0	1	2	3	4	5	6	7	8
GuidingHaptics	○	○	○	○	○	○	○	○	○
Vibration Band	○	○	○	○	○	○	○	○	○
Sound Cues	○	○	○	○	○	○	○	○	○



Ordena según tus preferencias del 1 (mejor opción) al 3 (peor opción) las técnicas utilizadas.

Ordena del 1 al 3. *

	1	2	3
GuidingHaptics	☐	☐	☐
Vibration Band	☐	☐	☐
Sound Cues	☐	☐	☐



Opinión de texto libre

Si lo deseas, puedes comentar cualquier aspecto relevante sobre **GuidingHaptics**.

Tu respuesta

Si lo deseas, puedes comentar cualquier aspecto relevante sobre **Vibration Band**.

Tu respuesta

Si lo deseas, puedes comentar cualquier aspecto relevante sobre **Sound Cues**.

Tu respuesta

Jupyter Notebook for the user study

See pages below.

In [ ]:
```
#!pip install jupyter_contrib_nbextensions
#!pip install jupyter_nbextensions_configurator
#!jupyter nbextensions_configurator enable --user
#!jupyter contrib nbextension install --user
#!jupyter nbextension enable codefolding/main
```

In [ ]:
```
# !pip install pygame
# !pip install pynput
# !pip install pyserial
# !pip install psychopy
```

In [ ]:
```
##### imports

from __future__ import print_function
import time
import attr
import numpy as np
import natnet

import threading

import asyncio

import serial

import csv
from datetime import datetime

from pynput.keyboard import Key, Controller
from pynput import keyboard

import pygame
from pygame.locals import *

import time
import random


# from psychopy import event, core, data, gui, visual

# from fileHandling import *
```

In [ ]:
```
#global vars




global prev
prev = None
global stop
stop = False
```

In [ ]:
```
#optitrack client
@attr.s
class ClientApp(threading.Thread):


    _client = attr.ib()
    _quiet = attr.ib()

    _last_printed = attr.ib(0)
    @classmethod
    def connect(cls, server_name, rate, quiet):
        if server_name == 'fake':
            client = natnet.fakes.SingleFrameFakeClient.fake_connect(rate=rate)
        else:
            client = natnet.Client.connect(server_name,logger=SilentLogger())
        if client is None:
            return None
        return cls(client, quiet)

    def run(self):
        if self._quiet:
            self._client.set_callback(self.callback_quiet)
        else:
            self._client.set_callback(self.callback)
        self._client.spin()

    def callback(self, rigid_bodies, markers, timing):

        """
        :type rigid_bodies: list[RigidBody]
        :type markers: list[LabelledMarker]
        :type timing: TimestampAndLatency
        """
        global prev


        if markers:
            if prev == None:
                print('Markers')
                for m in markers:
                    prev = m.position

            else:
                for m in markers:
                    prev = m.position


    def callback_quiet(self, *_):
        if time.time() - self._last_printed > 1:
            print('.')
            self._last_printed = time.time()


def main():


    global prev
    try:
        app = ClientApp.connect(None,10,False)
        app.run()

    except natnet.DiscoveryError as e:
        print('Error:', e)
```

In [ ]:

```
#silent logger
class SilentLogger(object):

    """Dummy logger implementation that just calls print."""

    def _log_impl(self, msg, *args):
        """Implementation function to make subclassing work."""
        print(msg % args)
    def _log(self, msg, *args):
        """Print msg % args."""
        self._log_impl(msg, *args)

    def _silence(self, msg, *args):
        return

    debug = _silence
    info = _log
    warning = _log
    error = _log
    fatal = _log
```

In [ ]:
```
def distance3D(a,b): #it's 1D actually
    return np.abs(a[2]-b[2])#np.sqrt((a[0]-b[0])**2+(a[1]-b[1])**2+(a[2]-b[2])**2)
```

In [ ]:
```
#Emulate key presses with 2 button controller

emul = 1
def emulate():
    controller = serial.Serial('COM7', timeout = 0, baudrate=115200)
    keybr = Controller()
    while True:
        key = controller.readline().decode()
        #print("jeje:"+key)
        if key != "":
            key = key[0]
            if(key == "1"):
                #print("x")
                keybr.press('x')
                keybr.release('x')
            else:
                #print("y")
                keybr.press('m')
                keybr.release('m')
        if emul == 0:
            break
    controller.close()
    controller.__del__()
```

In [ ]:
```
#function for printing the position of an optitrack marker
def fun2():
    global prev
    prev = None
    while not stop:
        time.sleep(0.5)
        print(prev)
```

In [ ]:
```
#to stop the emulator of key presses uncomment below and run
#emul = 0
```

In [ ]:
```
#run threads for optitrack client and key emulator
x = threading.Thread(target=main)
y = threading.Thread(target=emulate)

x.start()
y.start()
time.sleep(3)
```

In [ ]:
```
### ID of the user, change for a new user
subId = 25
```

```
orders = [[0,1,2],[0,2,1],[1,0,2],[1,2,0],[2,0,1],[2,1,0]]
print(orders[(subId-1) % len(orders)])
```

In [ ]:

```
####User study implementation RUN HERE
serBand = serial.Serial('COM9', timeout = 0, baudrate=4800)
serGH = serial.Serial('COM8', timeout = 0, baudrate=4800)
pygame.init()
pygame.mixer.set_num_channels(1)
left = pygame.mixer.Sound("./left-short.mp3")
right = pygame.mixer.Sound("./right-short2.mp3")
finished = pygame.mixer.Sound("./finished.mp3")
ready = pygame.mixer.Sound("./ready.mp3")
beep =  pygame.mixer.Sound("./beep.mp3")
channel = None


keyb = Controller()

xLine = 0 #x-axis line along which the user will be guided

minDistance = 0.01 #threshold of hitting target


targets = [-0.4,0.3,0,0.6,-0.3,0.5,-0.7,-0.5,-0.6,0.1,-0.4,0.0,-0.3,0.2,-0.1,0.4,-0.1,0.
           0.4,-0.2,0.7,-0.5,0.5,-0.3,-0.0,-0.5,0.3,-0.7,0.6,0.4,0.6]# -0.4,0.3,0 + seed

targets1 = [-0.2,0.1,-0.3,0.5,0.4,0.7,0.0,0.2,-0.7,0.0,-0.5,0.2,-0.3,0.3,-0.2,0.5,0.1,0.
            -0.3,0.7,0.3,0.7,-0.4,-0.2,-0.3,-0.1,-0.7,0.1,-0.6,0.0,-0.3]# -0.2,0.1,-0.3

targets2 = [0.2,-0.5,0,-0.1,0.7,0.3,0.5,-0.5, 0.4,-0.5,0.0,-0.3,0.2,-0.7,0.5,0.3,0.5,-0.
            -0.4,0.1,-0.4,-0.0,-0.2,0.0,-0.7,0.6,0.1,0.4,0.3,0.5,-0.6]# 0.2,-0.5,0 + see

targs = [targets,targets,targets2]
targs2 = [targets1,targets2,targets1]
targs3 = [targets2,targets1,targets]

soundMode = 1 #0 for words, 1 for beeps

mode = 0 #sound = 0, vibration band = 1, guiding haptics = 2

def auxWrite():
    global first
    if direction == "l":
        if first:
            serGH.write(str.encode('1 2 37\n'))
            time.sleep(0.3)
            serGH.write(str.encode('1 1 37\n'))
            first = False
        else:
            serGH.write(str.encode('1 1 37\n'))
    elif direction == "r":
        if first:
            serGH.write(str.encode('1 1 37\n'))
            time.sleep(0.3)
            serGH.write(str.encode('1 2 37\n'))
            first = False
        else:
            serGH.write(str.encode('1 2 37\n'))

def auxWrite2():
    global first
    if direction == "l":
        if first:
            serBand.write(str.encode('3\n'))
            time.sleep(0.3)
            serBand.write(str.encode('2\n'))
            first = False
        else:
            serBand.write(str.encode('2\n'))
    elif direction == "r":
        if first:
            serBand.write(str.encode('2\n'))
            time.sleep(0.3)
            serBand.write(str.encode('3\n'))
            first = False
        else:
            serBand.write(str.encode('3\n'))


def guide(first):
    if direction == "l":
        if mode == 0:
            #pygame.mixer.find_channel(True).play(left,-1)
            if soundMode == 0:
                channel.play(left,-1)
            else:
                channel.play(beep,-1)
            channel.set_volume(1.0, 0.0)
        elif mode == 1:
            #if first:
                #serBand.write(str.encode('2\n'))
            serialWriteThread2 = threading.Thread(target=auxWrite2)
            serialWriteThread2.start()
        elif mode == 2:
            serialWriteThread = threading.Thread(target=auxWrite)
            serialWriteThread.start()
    elif direction == "r":
        if mode == 0:
            #pygame.mixer.find_channel(True).play(right,-1)
            if soundMode == 0:
                channel.play(right,-1)
            else:
                channel.play(beep,-1)
            channel.set_volume(0.0, 1.0)
        elif mode == 1:
            if first:
                #serBand.write(str.encode('3\n'))
                serialWriteThread2 = threading.Thread(target=auxWrite2)
                serialWriteThread2.start()
        elif mode == 2:
            if first:
                serialWriteThread = threading.Thread(target=auxWrite)
                serialWriteThread.start()
            #serGH.write(str.encode('1 2 37\n'))

def endFinal():
    pygame.mixer.stop()
    if mode == 0:
        pygame.mixer.Sound.play(finished)
    keyb.press('q')
    keyb.release('q')
    serGH.write(str.encode('0\n'))
    serBand.write(str.encode('0\n'))
    serGH.close()
    serGH.__del__()
    serBand.close()
    serBand.__del__()

def end():
    pygame.mixer.stop()
    if mode == 0:
        pygame.mixer.Sound.play(finished)
    keyb.press('q')
    keyb.release('q')
    serGH.write(str.encode('0\n'))
    serBand.write(str.encode('0\n'))

def spam_q2():
    for i in range(8):
        keyb.press('q')
        keyb.release('q')
        time.sleep(0.05)

def spam_q():
    for i in range(8):
        keyb.press('q')
        time.sleep(0.002)
        keyb.release('q')
        time.sleep(0.05)
    time.sleep(0.3)
    keyb.press('w')
    keyb.release('w')


def waitSpace():
    with keyboard.Events() as events:
        for event in events:
            if event.key == keyboard.Key.space:
                break


def loop(condition,targets,final=False):
    global currentTarget
    global target
    global prevTarget
    global currentDirection
    global direction
    global counter

    change = True

    while True:#for x in range(1,len(targets)):
        guide(change)
        change = False
        direction = ""
        dist = distance3D(prev,(xLine,0,currentTarget))
        if dist < minDistance and target < len(targets):
            prevTarget = currentTarget
            target += 1
            if target >= len(targets):
                counter += 1
                file_write.writerow([counter,str(subId)+condition,datetime.now(),dist,cu
                if final:
                    endFinal()
                else:
                    end()
                break
            change = True
            currentTarget = targets[target]
            if prevTarget > currentTarget :
                direction = "l"
                currentDirection = "left"
            if prevTarget < currentTarget :
                direction = "r"
                currentDirection = "right"
        #print(distance3D(prev,(xLine,0,currentTarget)))
        file_write.writerow([counter,str(subId)+condition,mode,datetime.now(),dist,curre
        counter += 1
        time.sleep(0.0005)

    keyb.press('q')
    keyb.release('q')
    time.sleep(0.05)
    keyb.press('q')
    keyb.release('q')
    spam_qThread = threading.Thread(target=spam_q)
    spam_qThread.start()
    spam_qThread2 = threading.Thread(target=spam_q2)
    spam_qThread2.start()


orders = [[0,1,2],[0,2,1],[1,0,2],[1,2,0],[2,0,1],[2,1,0]]

order = orders[(subId-1) % len(orders)] #minus 1 beacuse they start at 1

currentTarget = targs[order[0]][0]#targets[0]
target = 0
prevTarget = 0
currentDirection = ""
direction = ""
first = True
modes = ["sound","vibration","guiding"]

counter = 1

if prevTarget > currentTarget :
    direction = "l"
    currentDirection = "left"
if prevTarget < currentTarget :
    direction = "r"
    currentDirection = "right"
with open('Data\\user'+str(subId)+'.csv', 'w+', newline='') as file:
    file_write = csv.writer(file)
    file_write.writerow(["counter","subId","mode","timestamp","distance","currentDirecti

    mode = order[0]
    channel = ready.play()
    #waitSpace()
    print("prepare condition: "+ modes[mode])
    time.sleep(2)
    waitSpace()


    ######First trial###########
    print("Starting condition 1 trial 1")
    loop("_1_1",targs[order[0]])
    spam_q2()
    waitSpace()
    time.sleep(3)
    waitSpace()
    #waitSpace_spam()
    #######Second Trial###########
    currentTarget = targs2[order[0]][0]#targets[0]
    target = 0
    prevTarget = 0
    currentDirection = ""
    direction = ""
    first = True
    if prevTarget > currentTarget :
        direction = "l"
        currentDirection = "left"
    if prevTarget < currentTarget :
        direction = "r"
        currentDirection = "right"


    print("Starting condition 1 trial 2")
    loop("_1_2",targs2[order[0]])
    spam_q2()
    waitSpace()
    time.sleep(3)
    waitSpace()
    #######THIRD TRIAL########
    currentTarget = targs3[order[0]][0]#targets[0]
    target = 0
    prevTarget = 0
    currentDirection = ""
    direction = ""
    first = True
    if prevTarget > currentTarget :
        direction = "l"
        currentDirection = "left"
    if prevTarget < currentTarget :
        direction = "r"
        currentDirection = "right"

    print("Starting condition 1 trial 3")
    loop("_1_3",targs3[order[0]])
    spam_q2()
    waitSpace()
    ################################################## END OF FIRST CONDITION ##########
    ############################################## START OF SECOND CONDITION ##########

    mode = order[1]
    print("prepare condition: "+ modes[mode])
    waitSpace()
    ######First trial############
    currentTarget = targs[order[1]][0]#targets[0]
    target = 0
    prevTarget = 0
    currentDirection = ""
    direction = ""
    first = True
    if prevTarget > currentTarget :
        direction = "l"
        currentDirection = "left"
    if prevTarget < currentTarget :
        direction = "r"
        currentDirection = "right"
    print("Starting condition 2 trial 1")
    loop("_2_1",targs[order[1]])
    spam_q2()
    waitSpace()
    #######Second Trial##########
    currentTarget = targs2[order[1]][0]#targets[0]
    target = 0
    prevTarget = 0
    currentDirection = ""
    direction = ""
    first = True
    if prevTarget > currentTarget :
        direction = "l"
        currentDirection = "left"
    if prevTarget < currentTarget :
        direction = "r"
        currentDirection = "right"


    print("Starting condition 2 trial 2")
    loop("_2_2",targs2[order[1]])
    spam_q2()
    waitSpace()
    #######THIRD TRIAL#######
    currentTarget = targs3[order[1]][0]#targets[0]
    target = 0
    prevTarget = 0
    currentDirection = ""
    direction = ""
    first = True
    if prevTarget > currentTarget :
        direction = "l"
        currentDirection = "left"
    if prevTarget < currentTarget :
        direction = "r"
        currentDirection = "right"

    print("Starting condition 2 trial 3")
    loop("_2_3",targs3[order[1]])
    spam_q2()
    waitSpace()
    ################################################## END OF SECOND CONDITION #########
    ############################################### START OF THIRD CONDITION ###########


    mode = order[2]
    print("prepare condition: "+ modes[mode])
    waitSpace()
    ######First trial###########
    currentTarget = targs[order[2]][0]#targets[0]
    target = 0
    prevTarget = 0
    currentDirection = ""
    direction = ""
    first = True
    if prevTarget > currentTarget :
        direction = "l"
        currentDirection = "left"
    if prevTarget < currentTarget :
        direction = "r"
        currentDirection = "right"
    print("Starting condition 3 trial 1")
    loop("_3_1",targs[order[2]])
    spam_q2()
    waitSpace()
    ######Second Trial##########
    currentTarget = targs2[order[2]][0]#targets[0]
    target = 0
    prevTarget = 0
    currentDirection = ""
    direction = ""
    first = True
    if prevTarget > currentTarget :
        direction = "l"
        currentDirection = "left"
    if prevTarget < currentTarget :
        direction = "r"
        currentDirection = "right"


    print("Starting condition 3 trial 2")
    loop("_3_2",targs2[order[2]])
    spam_q2()
    waitSpace()
    #######THIRD TRIAL#######
    currentTarget = targs3[order[2]][0]#targets[0]
    target = 0
    prevTarget = 0
    currentDirection = ""
    direction = ""
    first = True
    if prevTarget > currentTarget :
        direction = "l"
        currentDirection = "left"
    if prevTarget < currentTarget :
        direction = "r"
        currentDirection = "right"

    print("Starting condition 3 trial 3")
    loop("_3_3",targs3[order[2]],True)
    spam_q2()
    waitSpace()
    ################################################## END OF THIRD CONDITION ##########
```

Python code for the secondary tasks

See pages below.

```
In [ ]: # -*- coding: utf-8 -*-
        from psychopy import event, core, data, gui, visual
        from fileHandling import *
        from datetime import datetime
        from pynput import mouse as pynputMouse
        class Experiment:
            def __init__(self, win_color, txt_color):
                self.stimuli_positions = [[-.2, 0], [.2, 0], [0, 0]]
                self.win_color = win_color
                self.txt_color = txt_color

            def create_window(self, color=(1, 1, 1)):
                # type: (object, object) -> object
                color = self.win_color
                win = visual.Window(monitor="testMonitor",
                                    color=color, fullscr=False,screen = 1)
                return win

            def settings(self):
                experiment_info = {'Subid': '', 'Age': '', 'Experiment Version': 0.1,
                                   'Sex': ['Male', 'Female', 'Other'],
                                   'Language': ['English', 'Swedish', 'Spanish'], u'date':
                                       data.getDateStr(format="%Y-%m-%d_%H:%M")}

                info_dialog = gui.DlgFromDict(title='Stroop task', dictionary=experiment_info,
                                              fixed=['Experiment Version'])
                experiment_info[u'DataFile'] = u'Data' + os.path.sep + u'stroop.csv'
                experiment_info[u'DataFile2'] = u'Data' + os.path.sep + u'' + experiment_info["S
                if info_dialog.OK:
                    return experiment_info
                else:
                    core.quit()
                    return 'Cancelled'

            def create_text_stimuli(self, text=None, pos=[0.0, 0.0], name='', color=None):
                '''Creates a text stimulus,
                '''
                if color is None:
                    color = self.txt_color
                text_stimuli = visual.TextStim(win=window, ori=0, name=name,
                                               text=text, font=u'Arial',
                                               pos=pos,
                                               color=color, colorSpace=u'rgb')
                return text_stimuli

            def create_math_stimuli(self, text=None, pos=[0.0, 0.0], name='', color=None):
                '''Creates a text stimulus,
                '''
                if color is None:
                    color = "White"
                text_stimuli = visual.TextStim(win=window, ori=0, name=name,
                                               text=text, font=u'Arial',
                                               pos=pos,
                                               color=color, colorSpace=u'rgb')
                return text_stimuli

            def create_trials(self, trial_file, randomization='random'):
                '''Doc string'''
                data_types = ['Response', 'Accuracy', 'RT', 'Sub_id', 'Sex','Time_stamp']
                with open(trial_file, 'r') as stimfile:
                    _stims = csv.DictReader(stimfile)
                    trials = data.TrialHandler(list(_stims), 1,
                                               method="random")
                [trials.data.addDataType(data_type) for data_type in data_types]
                return trials

            def present_stimuli(self, color, text, position, stim):
                _stimulus = stim
                color = color
                position = position
                if settings['Language'] == "Swedish":
                    text = swedish_task(text)
                elif settings['Language'] == "Spanish":
                    text = spanish_task(text)
                else:
                    text = text
                #print(position)
                _stimulus.pos = [position[1],-position[0]]
                _stimulus.setColor(color)
                _stimulus.setText(text)
                return _stimulus

            def running_experiment(self, trials, testtype,mode="stroop"):
                _trials = trials
                testtype = testtype

                timer = core.Clock()
                #stimuli = [self.create_text_stimuli(window) for _ in range(4)]
                stimuli = [self.create_math_stimuli(window) for _ in range(4)]

                for trial in _trials:
                    # Fixation cross
                    fixation = self.present_stimuli(self.txt_color, '+', self.stimuli_positions[
                                                    stimuli[3])
                    fixation.draw()
                    window.flip()
                    core.wait(.6)
                    timer.reset()
                    print(trial)
                    # Target word
                    if mode == "stroop":
                        target = self.present_stimuli(trial['colour'], trial['stimulus'],
                                                  self.stimuli_positions[2], stimuli[2])
                    elif mode == "math":
                        target = self.present_stimuli(self.txt_color, trial['stimulus'],
                                                  self.stimuli_positions[2], stimuli[2])
                    target.draw()
                    # alt1
                    alt1 = self.present_stimuli(self.txt_color, trial['alt1'],
                                                self.stimuli_positions[0], stimuli[0])
                    alt1.draw()
                    # alt2
                    alt2 = self.present_stimuli(self.txt_color, trial['alt2'],
                                                self.stimuli_positions[1], stimuli[1])
                    alt2.draw()
                    window.flip()
        #           mouse1.clickReset()
                    keys = []
        #             while True:
        #                 keys = event.getKeys(keyList=['q'])##keyList=['x', 'm', 'q']
        #                 if len(keys) > 0:
        #                     print(keys)
        #                     print("we")
        #                     break
        #                 buttons, times = mouse1.getPressed(True)
        #                 if buttons[0] == 1:
        #                     if mouse1.getPos()[0] < 0:
        #                         keys.append('x');
        #                         print(keys)
        #                         break
        #                     elif mouse1.getPos()[0] > 0:
        #                         keys.append('m');
        #                         print(keys)
        #                         break
                    # keys = []
                    # while True:
                    #     with pynputMouse.Events() as events:
                    #         # Block at most one second
                    #         keys = event.getKeys(keyList=['x','m','q'])##keyList=['x', 'm', 'q
                    #         if 'q' in keys:
                    #             #print(keys)
                    #             #print("we")
                    #             break
                    #         #print(events)
                    #         #print(type(events))
                    #         #print(type(events[0]))
                    #         event1 = events.get(1.0)

                    #         if event1 != None:
                    #            #print(event1)
                    #            #print(type(event1))
                    #            # print(event.x)
                    #             #print("----------")
                    #             if isinstance( event1, pynputMouse.Events.Click):
                    #                 print("jejjeej")
                    #                 print(event1.button)
                    #                 print(type(event1.button))
                    #                 if event1.button is pynputMouse.Button.left:
                    #                     print("jejjeej")
                    #                     print(event1.x)
                    #                     if event1.x  < 3500 and event1.x >2580:
                    #                         keys.append('x');
                    #                         print(keys)
                    #                     elif event1.x  > 3500:
                    #                         keys.append('m');
                    #                         print(keys)
                    #                     break

                    keys = event.waitKeys(keyList=['x', 'm', 'q'])

                    resp_time = timer.getTime()
                    if testtype == 'practice':
                        if keys[0] != trial['correctresponse']:
                            instruction_stimuli['incorrect'].draw()

                        else:
                            instruction_stimuli['right'].draw()

                        window.flip()
                        core.wait(2)

                    if testtype == 'test':
                        if keys[0] == trial['correctresponse']:
                            trial['Accuracy'] = 1
                        else:
                            trial['Accuracy'] = 0

                        trial['RT'] = resp_time
                        trial['Response'] = keys[0]
                        trial['Sub_id'] = settings['Subid']
                        trial['Sex'] = settings['Sex']
                        trial['Time_stamp'] = datetime.now()
                        write_csv(settings[u'DataFile'], trial)
                        write_csv(settings[u'DataFile2'], trial)
                    event.clearEvents()
                    print(f"keys: {keys}")
                    if 'q' in keys:
                        print(f"breaking because keys: {keys}")
                        break


        def create_instructions_dict(instr):
            start_n_end = [w for w in instr.split() if w.endswith('START') or w.endswith('END')]
            keys = {}

            for word in start_n_end:
                key = re.split("[END, START]", word)[0]

                if key not in keys.keys():
                    keys[key] = []

                if word.startswith(key):
                    keys[key].append(word)
            return keys


        def create_instructions(input, START, END, color="Black"):
            instruction_text = parse_instructions(input, START, END)
            print(instruction_text)
            text_stimuli = visual.TextStim(window, text=instruction_text, wrapWidth=1.2,
                                           alignHoriz='center', color=color,
                                           alignVert='center', height=0.06)

            return text_stimuli


        def display_instructions(start_instruction=''):
            # Display instructions

            if start_instruction == 'Practice':
                instruction_stimuli['instructions'].pos = (0.0, 0.5)
                instruction_stimuli['instructions'].draw()

                positions = [[0,.1], [0,-.3], [0, -.1]]#-.2, 0], [.2, 0], [0, 0]]
                examples = [experiment.create_text_stimuli() for pos in positions]
                example_words = ['green', 'blue', 'red']
                if settings['Language'] == 'Swedish':
                    example_words = [swedish_task(word) for word in example_words]

                if settings['Language'] == 'Spanish':
                    example_words = [spanish_task(word) for word in example_words]

                for i, pos in enumerate(positions):
                    examples[i].pos = pos
                    if i == 0:
                        examples[0].setText(example_words[i])
                    elif i == 1:
                        examples[1].setText(example_words[i])
                    elif i == 2:
                        examples[2].setColor('Green')
                        examples[2].setText(example_words[i])

                [example.draw() for example in examples]

                instruction_stimuli['practice'].pos = (0.0, -0.5)
                instruction_stimuli['practice'].draw()

            elif start_instruction == 'PracticeMath':
                instruction_stimuli['instructions_math'].pos = (0.0, 0.5)
                instruction_stimuli['instructions_math'].draw()
                positions = [[0,.1], [0,-.3], [0, -.1]]#-.2, 0], [.2, 0], [0, 0]]
                examples = [experiment.create_text_stimuli() for pos in positions]
                example_words = ['2', '3', '1 + 1']
                if settings['Language'] == 'Swedish':
                    example_words = [swedish_task(word) for word in example_words]

                if settings['Language'] == 'Spanish':
                    example_words = [spanish_task(word) for word in example_words]

                for i, pos in enumerate(positions):
                    examples[i].pos = pos
                    if i == 0:
                        examples[0].setText(example_words[i])
                    elif i == 1:
                        examples[1].setText(example_words[i])
                    elif i == 2:
                        examples[2].setColor('White')
                        examples[2].setText(example_words[i])

                [example.draw() for example in examples]

                instruction_stimuli['practice'].pos = (0.0, -0.5)
                instruction_stimuli['practice'].draw()

            elif start_instruction == 'Wait':
                instruction_stimuli['wait'].draw()

            elif start_instruction == 'Test':
                instruction_stimuli['test'].draw()

            elif start_instruction == 'Test2':
                instruction_stimuli['test_a'].draw()

            elif start_instruction == 'Test3':
                instruction_stimuli['test_b'].draw()

            elif start_instruction == 'End':
                instruction_stimuli['done'].draw()

            window.flip()
            event.waitKeys(keyList=['space','w'])
            event.clearEvents()


        def swedish_task(word):
            swedish = '+'
            if word == "blue":
                swedish = u"blå"
            elif word == "red":
                swedish = u"röd"
            elif word == "green":
                swedish = u"grön"
            elif word == "yellow":
                swedish = "gul"
            return swedish

        def spanish_task(word):
            swedish = '+'
            if word == "blue":
                swedish = "azul"
            elif word == "red":
                swedish = "rojo"
            elif word == "green":
                swedish = "verde"
            elif word == "yellow":
                swedish = "amarillo"
            return swedish

        if __name__ == "__main__":
            background = "Black"
            back_color = (0, 0, 0)
            textColor = "White"
            # text_color = (1, 1, 1)
            experiment = Experiment(win_color=background , txt_color=textColor)
            settings = experiment.settings()
            language = "English"#settings['Language']
            instructions = read_instructions_file("INSTRUCTIONS", language, language + "End")
            instructions_dict = create_instructions_dict(instructions)
            instruction_stimuli = {}

            window = experiment.create_window(color=back_color)

            for inst in instructions_dict.keys():
                instruction, START, END = inst, instructions_dict[inst][0], instructions_dict[in
                instruction_stimuli[instruction] = create_instructions(instructions, START, END,

            # We don't want the mouse to show:
            mouse1 = event.Mouse(visible=True)

            ################################First Condition################################
            #wait for the no secondary task trial to finish
            display_instructions(start_instruction='Wait')
            display_instructions(start_instruction='Wait')
            # Stroop Practice Trials
            display_instructions(start_instruction='Practice')
            practice = experiment.create_trials('practice_list.csv')
            experiment.running_experiment(practice, testtype='practice',mode="stroop")

            #Stroop Test trials
            settings["Subid"] = settings["Subid"]+"_1_2"
            display_instructions(start_instruction='Test')
            trials = experiment.create_trials('stimuli_list.csv')
            experiment.running_experiment(trials, testtype='test',mode="stroop")
            display_instructions(start_instruction='End')

            #Math Practice trials
            display_instructions(start_instruction='PracticeMath')
            practice = experiment.create_trials('practice_list2.csv')
            experiment.running_experiment(practice, testtype='practice',mode="math")

            #Math Test trials
            settings["Subid"] = settings["Subid"][:-3]+"1_3"
            display_instructions(start_instruction='Test')
            trials = experiment.create_trials('stimuli_list2.csv')
            experiment.running_experiment(trials, testtype='test',mode="math")
            display_instructions(start_instruction='End')

            ################################Second Condition################################
            #wait for the no secondary task trial to finish
            display_instructions(start_instruction='Wait')
            display_instructions(start_instruction='Wait')

            # Stroop Practice Trials
            #display_instructions(start_instruction='Practice')
            #practice = experiment.create_trials('practice_list.csv')
            #experiment.running_experiment(practice, testtype='practice')

            #Stroop Test trials
            settings["Subid"] = settings["Subid"][:-3]+"2_2"
            display_instructions(start_instruction='Test2')
            trials = experiment.create_trials('stimuli_list.csv')
            experiment.running_experiment(trials, testtype='test',mode="stroop")
            display_instructions(start_instruction='End')

            #display_instructions(start_instruction='Wait')

            #Math Practice trials
            #display_instructions(start_instruction='PracticeMath')
            #practice = experiment.create_trials('practice_list2.csv')
            #experiment.running_experiment(practice, testtype='practice')

            #Math Test trials
            settings["Subid"] = settings["Subid"][:-3]+"2_3"
            display_instructions(start_instruction='Test3')
            trials = experiment.create_trials('stimuli_list2.csv')
            experiment.running_experiment(trials, testtype='test',mode="math")
            display_instructions(start_instruction='End')


            ################################Third Condition################################
            #wait for the no secondary task trial to finish
            display_instructions(start_instruction='Wait')
            display_instructions(start_instruction='Wait')

            # Stroop Practice Trials
            #display_instructions(start_instruction='Practice')
            #practice = experiment.create_trials('practice_list.csv')
            #experiment.running_experiment(practice, testtype='practice')

            #Stroop Test trials
            settings["Subid"] = settings["Subid"][:-3]+"3_2"
            display_instructions(start_instruction='Test2')
            trials = experiment.create_trials('stimuli_list.csv')
            experiment.running_experiment(trials, testtype='test',mode="stroop")
            display_instructions(start_instruction='End')

            #Math Practice trials
            #display_instructions(start_instruction='PracticeMath')
            #practice = experiment.create_trials('practice_list2.csv')
            #experiment.running_experiment(practice, testtype='practice')

            #display_instructions(start_instruction='Wait')

            #Math Test trials
            settings["Subid"] = settings["Subid"][:-3]+"3_3"
            display_instructions(start_instruction='Test3')
            trials = experiment.create_trials('stimuli_list2.csv')
            experiment.running_experiment(trials, testtype='test',mode="math")
            display_instructions(start_instruction='End')


            # End experiment but first we display some instructions
            #display_instructions(start_instruction='End')
            display_instructions(start_instruction='Wait')
            window.close()
```