Exercise Optional 1 – Lécuyer’s pseudo-haptics

Jaume Benseny

Preconditions

- The exercise was developed with Python 2.7.3 running on Ubuntu 12.04 LTS
- Mouse functionality was implemented using xdotool from Jordan Sissel
  - http://www.semicomplete.com/projects/xdotool/
- In order to execute it, you should install xdotool library on linux, for ubuntu/debian:
  - apt-get install xdotool
  - other dependencies : xlib , xtst (automatically installed with apt-get)
- Commands executed form the xdotool library:
  - “xdotool getmouselocation”
  - “xdotool mousemove x y”
  - "xdotool search --name bump_final.png" (get window ID based on name)
  - “xdotool mousemove --window 72737374 1 41" (move mouse to initial position relative to image position)

Version 1 – horizontal stairway

- In order to validate the correct function of xdotool library as well as its feedback delay a first script *horitzontal_stairway.py* was developed.
- This script simulates a stairway with 8 levels. Them pointer climbs up from left to right. This way the difficulty to move the pointer increases when moving between levels. For simplicity, in this first exercise no image was created nevertheless the pointer can only be moved within a window of 800px x 600px. Initial position is 0px , 300px.
- In order to model the new behaviour of the pointer, we took into account that we deal with a trajectory based task in the context of a continuous control exercise. If we analyse the exercise from the point of view of the Steering Law, we can say that the pointer travels from point A to point B where the index of difficulty (ID) depends on the amplitude of the movement B-A and their relative height.
- For this exercise a matrix of 800 x 600 was created where value of each column equals de previous plus 1 starting form 0 in column 0.
- The script works as follows: Before the script accepts the present position of the pointer as the final one, distance between present position and last position has to be greater than cost. Cost is calculated as MATRIX(B)//100 which automatically creates 8 levels. If the difference between B-A is smaller than the cost to get from A to B the mouse pointer is sent back to its last position.
- SUMMARY: we can say that this script allows the pointer to travel to higher places depending on its speed between A and B and the height of B compared to A.
- LIMITATION: Speed is calculated as distance/sampling where sampling time is the time between mouse positions the xdotool allow us to get.
- LIMITATION: It only works in left to right direction.
- LIMITATION: When the pointer moves too fast no difficulty is perceived.

Version 2 – the inverted bump

If there is any problem with the execution of the script please don't hesitate to contact me and see what can be the cause.

- This script can be found as *bump.py*.
- This script generates a matrix named Z of 800px x 600px where a bump is created with values that increase its value from its borders towards its centre as radius_number power 2. (radius_number starts as 1 and increments +1)
- Image of the bump was plotted with the same algorithm and saved as png in the file system. In order to be able to surf over the image, it's automatically opened by the programme.
- Synchronization between initial position in the matrix Z and position of the pointer on the image is achieved by the command:
  - "xdotool mousemove --window 72737374 1 41"
  - Where "1 41" are the pixels between the upper left side of the window that contains the image and the upper left side of the image itself
  - **it runs automatically! But you MUST wait until the mouse is automatically located on the upper left side of the image. (alert messages are shown on the python console)**
- In order to make the user feel movement difficulty proportional to the difference of height between point A and B, the script calculates the difference between heights as a multiple of the Z(B) – Z(A). Therefore COST = P * (Z(B) – Z(A)).
- Before the script accepts the present position of the pointer as the final one, the user will need to accumulate a number of pixels enough to compensate the cost between points.
- The factor that determines if the cursor gets increased difficulty to climb up the bump or gets extra momentum because it gets down the bump is the sign of the COST.
- When climbing: For the present position to be accepted, both x and y dimensions have to be evaluated. The cost condition can be satisfied by x and y dimensions independently. Therefore it's possible that the present x position of the cursor is accepted but not the y position, in this case the pointer is moved ONLY to the present position of those dimensions which comply with the condition.
- When getting down the bump: The new position gets extra pixels.
- SUMMARY: The user feels that more effort is required to get over the inverted bump because pixels are accumulated in the buffer until there are enough to climb to the next level.
- LIMITATION: Getting down from the inverted bump doesn't give momentum proportional to the altitude lost.

Comparison with Lécuyer's pseudo-haptics

I tried to implement Lécuyer's algorithm and try to control the pointer bit-by-bit, however it has been not possible due to the poor sampling frequency achieved with the xdotool configuration. Instead I used the option of comparing present position with last position that better addressed the circumstances of my experiment.