Development of a Steering Law Experiment platform with haptic device Phantom Omni

by Alejandro Tatay Pascual

May 12th, 2010

supervised by :

Professor Xiangshi Ren
Kochi University of Technology, Japan

Dr. M. Carmen Juan Lizandra
Universidad Politécnica de Valencia, Spain

Table of Contents

Chapter 1 - Introduction

1.1. Overview of the project

This project is related to the field of Human-Computer Interaction (HCI) which seeks, as the name says, the study of interaction between users and computer devices. HCI is involved with several fields of study such as computer science, behavioural science, design or usability, just to mention a few.

This document describes the development of Steery, a software platform made to set up and perform trajectory-based experiments. This type of experiments can be used to enhance the design and development of new devices and graphical interfaces.

Particularly, the work described was focused on the haptic device Phantom Omni, an articulated stylus with force-feedback from Sensable Technologies [1]. The system has a configurable set of parameters that the experiment designer introduces through the interface and then launch the experiment.

The system has been developed in C/C++ language using OpenGL Utility Toolkit (GLUT) libraries to display the Graphical User Interface (GUI) and OpenHaptics API toolkit, which allows the control and development of the Phantom device features.

Finally, an investigation has been done with a variety of experiments using the developed software to study trajectory-based tasks in different environments. These experiments were carried out by Hironori Ishiyama [2] and Onishi Yoshinobu [3] for their respective Bachelor's Thesis. These experiments aimed the verification of Steering Law formula in three-dimensional tunnels such as a torus and a cylinder, respectively. Subsequently, another experiment nicknamed *boxwalls* was performed to study two-dimensional steering task in a three-dimensional environment.

Therefore, the main goal of this project has been the development of an application that allows HCI researchers to design, set and launch trajectory-based experiments fast and easily, lightening the process by removing the programming stage.

1.2. Structure of the work

The work has been conducted in 4 different stages:

1. System requirements analysis and design of the use cases.

2. Construction of the GUI layer using the GLUT/GLUI prototypes.

3. Integration of the OpenHaptics toolkit and device control. Development of the different types of experiments.

4. Usability tests and design of several experiments by different members of Ren Laboratory. Output data and participants' subjective evaluation was gathered to interpret and understand the utility and improvement of trajectory-based tasks in graphical interfaces.

1.3. Disposition of the document

The first part is an introduction to this project: it presents briefly the project carried out, its structure, objectives and disposition of this document.

The second part of this document is a Literature Review about the use and study of Steering Law in the field of HCI: initially, it introduces the reader to the field of HCI and literature involved. After describing some technical definitions, it focus on Steering Law related work, derived from Accot and Zhai investigation. Finally, several studies combining Haptics research and trajectory tasks are explained briefly.

The third chapter presents in detail the OpenHaptics toolkit which was used to develop the project. In this part this haptic library and its approach to program a haptic device is described.

The fourth chapter is a presentation of the implemented project. Starting from the GUI layer made with GLUT/GLUI prototypes, it continues relating the most important Steery haptic parameters and their development. Afterwards, each type of experiment, implementation issues and other relevant features are described.

The fifth chapter deals with the conducted experiments in three-dimensional environments, resuming the results and discussion according to the data obtained.

Then, conclusions and future work are explained in the fifth chapter.

Next, the references of this thesis are reported.

Finally, a variety of appendices are attached to the document. At page 35 starts the Steery documentation generated automatically from the source code comments -appropriately formatted- with Doxygen, a documentation generator under GNU General Public License. Next, at page 63, the Steery User's Guide is enclosed. This guide is expected to be used by future researchers as a manual to utilize Steery application. At last but not least is enclosed the *boxwalls* report, which presents an experiment conducted with Steery in a three-dimensional scenario.

Chapter 2 - Literature review of Steering Law in HCI

2.1. Background

In the field of computer technologies, devices and graphical user interfaces have obtained more relevancy year by year. Everyday we interact with objects, systems and machines which are a fundamental and daily element of our lives: personal computers, cellphones, multimedia players, cameras, e-book readers or video game consoles are some of them (Illustration 1). As a consequence, this interaction has become a priority when it comes the time to develop a system, when concepts such as usability, design, interactivity or visualization take over in detriment of other system functionalities. This type of research is often studied in HCI.

Illustration 1: computer systems used in daily life

Academic investigation in HCI combines the experimental methods and intellectual framework of cognitive psychology with powerful tools from computer science. HCI benefits from related fields such as education where computers are increasingly used in programs ranging from elementary school through professional skills development. The theory and measurement techniques of educational psychology are applicable to study the learning process in novice computer users. Business system design and management decision making are endeavours which are being increasingly shaped by the nature of the computer facilities. Library and information services are also dramatically influenced by the availability of computer-based systems.

Therefore, investigation and design of improved and more usable methods of interaction between human users and computer devices has become one of the main goals in the field of HCI. Theories, paradigms and taxonomies compete to ameliorate it: reducing learning times, faster performance on tasks, lower rate errors, higher subjective satisfaction, better human retention over time, etc.

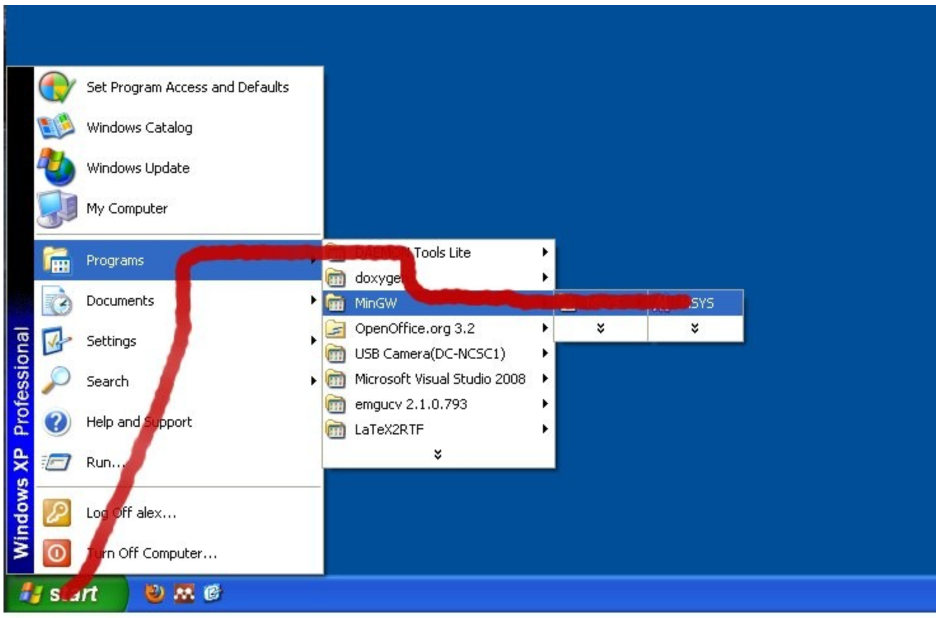


Illustration 2: Traversing nested menus involves multiple segments of steering tasks

Particularly, trajectory-based interactions, such as navigating through nested-menus, selecting items in menus, tracing patterns or steering through straight “tunnels” is a common tasks while interacting with computer systems (Illustration 2). The current project has been focused on this field of investigation, which leads to the next point, Steering Law.

2.2. Steering Law

Conversely, while HCI investigation takes place, it becomes difficult to find a thorough technique to evaluate and compare interfaces. The advancement of HCI lies in “hardening” the field with quantitative, engineering-like models. Extending movement theory to human perceptual-motor system, Paul Fitts [5] found a formal relationship that models the speed versus accuracy trade-off in aimed movements. It predicts that the time T need to cover a target of width W and height A is logarithmically related to the inverse of spatial relative error $\frac{A}{W}$, which is called Index of Difficulty (ID).

$$T = a + b\log_2\left(\frac{A}{W} + c\right)$$

Equation 3: Fitts' Law

Thus, the Index of Difficulty of the target behaves according to the empirically obtained constants *a, b*, and *c*=[0,1]. Due to its accuracy and robustness, Fitts' Law has been a popular research topic, hence numerous studies rely on Fitts' Law to compare device performance results. This formula revealed a rigorous an intuitive trade-off in HCI: the faster we move, the less precise our movements are, or vice versa.

Using integral calculus, Johnny Accot and Shumin Zhai extended Fitts' Law to develop a most general mathematical statement of it [6], also known as Steering Law. By this derivation the formula gets simplified under the circumstance of a straight tunnel as

$$T = a + b\frac{A}{W}$$

Equation 4: Steering Law

In this context, the steering law is a predictive model of human movement, concerning the time with which a user may steer a pointing device through a two-dimensional tunnel presented on a screen. Many researchers find it surprising that the steering law model predicts performance as well as it does, confirming the robustness of Fitts' law. Since this point, this new paradigm becomes a useful tool to develop, design, compare and analyse new GUIs and pointing devices, where navigation through menus may be represented as steering tasks (trajectory-based tasks).

In addition, Accot and Zhai experimented with different types of tunnel, increasing the complexity of the tunnel. From those experiments, theoretical models are derived to quantify the difficulty of path steering tasks.

Other authors, as Mackenzie *et al.* [7] and Kattinakere [8] adapted Fitts' Law for three-dimensional tunnels, selecting two-dimensional targets, as represented in Equation 5. The target was a rectangle of width *W* and height *H*. They found that Fitts' Law can be adapted by replacing the *W* in Equation 3 by the smaller of two sides of the target.

$$T=a+b\log_2(\sqrt{(\frac{A}{W})^2+\eta(\frac{A}{W})^2}+1)$$

Equation 5: Kattinakere's derivation

Accot and Zhai demonstrated [9] that this equation 5 provides a more precise prediction as it considers the effects of both width and height of the target.

2.3. Haptics and Steering Law

In recent years there has been much discussion about the "look and feel" of user interfaces. The part of "look" has been constantly researching and improving with better GUI designs, more usable, with reduced user learning times and faster performance. However, the "feel" word brings us to a still unexploited field: *haptics* technology. This term comes from the Greek word ἁπτικός (*haptikos*), meaning "I fasten onto, I touch".

Haptic interaction with computers has primarily to do with input. While we have physical contact with transducers such as mice, their design does not afford us to feel the edges of things that we are pointing at. That is we can not feel the object that the mouse is touching. In contrast, there are haptic output devices, usually called *force-feedback* devices. These devices can provide some output by tactile either kinaesthetic feedback based on servo control systems. In practice, the quality and appropriateness of this "feel" is crucial to determine a device's effectiveness and acceptance in a particular context. These devices are commonly controlled by the hand, and there is a wide variety of types in the market: data gloves (Illustration 6) based on sensor technologies that recognize the pose of the hand; high-precision pen-shaped grasping devices (Illustration 8) with several degrees of freedom and end-effector force-feedback; or hand exoskeletons(Illustration 7) which adds resistive force to each finger, these are just some examples.

Illustration 6: 5DT Data Glove. Fifth Dimension Technologies©

Illustration 8: Omega 6 haptic inferface. Force Dimension©

Illustration 7: CyberForce. CyberGlove Systems©

Haptic technology is widely used in a variety of applications. For example, virtual reality worlds in recreational applications as computer video games. In particular, players can sense virtual world properties by the utilization of joysticks and steering wheels (Illustration 9). However, more academical and professional fields can enhance with the use of haptics technology. Namely, remote controlled robotic tools -also called teleoperators- incorporate this feature, hence the operator can sense the force-feedback from the remote location where the robot is working. Medical scenarios as invasive procedures with remote surgery using teleoperators is an example of this application. The surgeon palpates and operates the patient remotely. This way, he can perform many operations of a similar type with less fatigue and better patient outcome statistics. Besides, medical teleoperators can also be used as a simulation training for students practice(Illustration 10).

Illustration 10: LaparoscopyVR Surgical Simulator. CAE HEalthcare©

Illustration 9: force-feedback devices. Logitech©

Likewise, haptics is used in other applications such as mobile consumer technologies (where the user receives haptic feedback from the cellphone vibrator), graphic design, animation or pressure in Robotics. Basically, haptic technology consists of actuators (polymers, piezoelectric or electrostatic) that apply forces in response to an electrical stimulus.

Haptics is also used in investigation to simulate a surface being touched. Even it does not provide a realistic feel, it does provide useful feedback, allowing the user to discriminate and recognize among various emulated shapes, textures and resiliencies.

Similarly, several studies such as [10] [11] [12] and [13] have shown that the performance of steering tasks can be improved by providing users with tactile feedback during the movement along the tunnel. Xing-Dong *et al.* [14] proposes a model for force-enhanced goal crossing task,

$$T = a + b\left(\frac{A}{W + \eta * S}\right)$$

Equation 11: Xing-Dong's derivation

where the difficulty of the task is inversely proportional to the spring stiffness *S*. This value represents the intensity of the guiding force, and the empirical constant *η* is determined by the contributions of *W* and *T*. In Xing-Dong's study, force-feedback was provided by a Phantom Omni device, the same device that this Steery project has been done for, as well (Illustration 12).

Illustration 12: Phantom Omni. Sensable Technologies©

We have introduced some recent approaches that study the difficulty of steering tasks for different environments. The purpose of Steery is to provide a software platform to design and experiment steering tasks onto the Phantom Omni, exploiting the resources of this haptic device.

Chapter 3 - OpenHaptics

3.1. Introduction

OpenHaptics is a software toolkit developed to make programming simpler by encapsulating the basic steps common to all haptics/graphics applications. This encapsulation is implemented in the C++ classes of the QuickHaptics micro API. By anticipating typical use scenarios, a wide range of default parameter settings is put into place that allow the user to code haptically enabled applications very efficiently This SDK constructed by Sensable Technologies [1] constituted the backbone of this project, allowing the programming and control of the device Phantom Omni. The OpenHaptics toolkit includes the QuickHaptics micro API, Haptic Device API (HDAPI) and Haptic Library API (HLAPI), as we can see on Illustration 13.

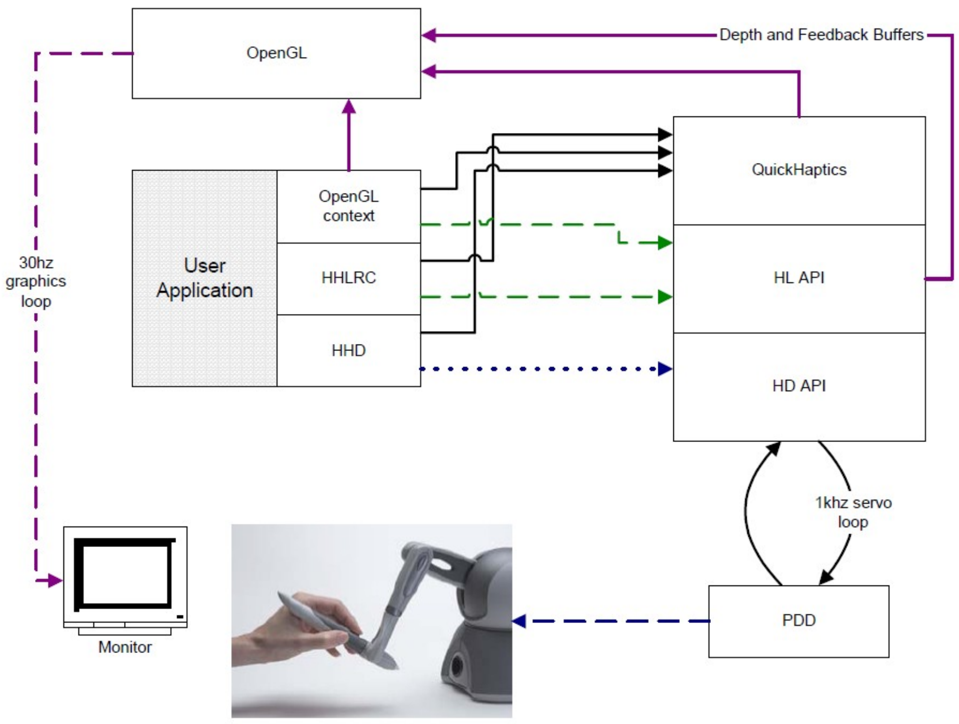


Illustration 13: OpenHaptics overview

3.2. QuickHaptics

QuickHaptics is a micro API that makes it fast and easy to write new haptic applications or to add haptics to existing applications. Built-in geometry parsers and intelligent default parameters make it possible to set up haptics/graphics scenes with a minimal amount of code.

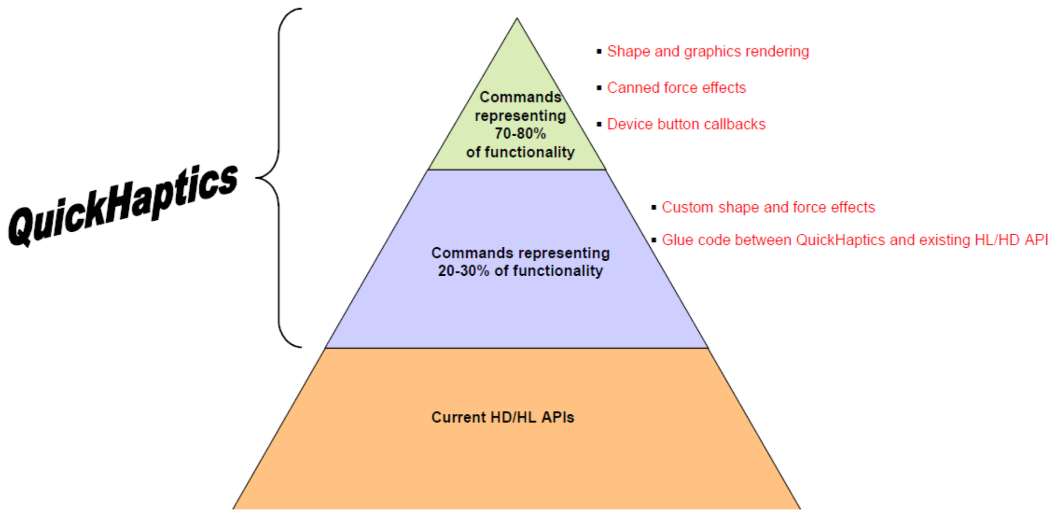

QuickHaptics
Commands representing 70-80% of functionality
Commands representing 20-30% of functionality
Current HD/HL APIs
Shape and graphics rendering
Canned force effects
Device button callbacks
Custom shape and force effects
Glue code between QuickHaptics and existing HL/HD API

Illustration 14: OpenHaptics vs. QuickHaptics micro API Functionality

Let's see it on Illustration 14.

QuickHaptics is implemented in C++ and it defines four primary functional classes:

- DeviceSpace class: it defines the force properties and user interaction through the haptic workspace for a particular PHANTOM device. This class' methods manage force effects (like friction or damping) and User Callbacks, or function calls that occur as a response to an event.

- QHRenderer class: base class for on-screen window graphics. It renders from a camera viewpoint and lets the user feel those shapes with a haptic device. From this class two other classes are inherited: QHWin32 and QHGLUT. This classes are created specifically for use with Microsoft Win32 API or the OpenGL Utility Toolkit (GLUT), respectively. QHRenderer defines a simple display list for haptics and graphics, a world space to PHANTOM device space transformation and a simple camera and lighting models.

- Shape class: it defines the base class for all geometry primitives, such as Sphere, cylinder, Cone, Box, Text, TriMesh... Each Shape has haptics and graphics properties, including textures, colour, spin, position, friction, damping, and so on. This is information is saved in the QHRenderer display list. Each shape defined has an implied bounding box that encompasses the shape, and all bounding boxes combined comprise a single global bounding box that contains all of the objects inside it. The device space is mapped into this world space and clipped by front and far planes as this diagram (Illustration 15) shows:

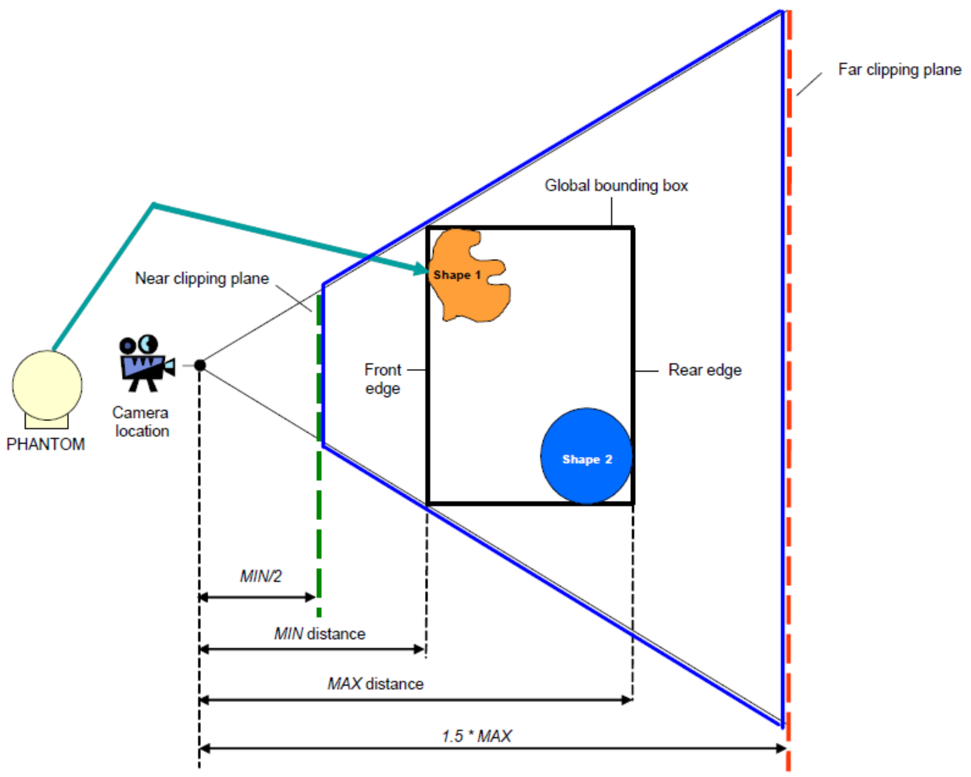


Illustration 15: default clipping planes for world space

- Cursor class: graphical representation of the end point of the second link on the PHANTOM device. This end point is sometimes called the haptic interface point (HIP). The HIP is in the same corresponding location for other PHANTOM devices. This class pulls together information from all of the three aforementioned classes because calculating the final Cursor location requires interaction with all of the components in a scene.

3.3. HDAPI and HLAPI overview

The HDAPI (Haptic Device API) provides low-level access to the haptic device, enables haptics programmers to render forces directly, offers control over configuring the runtime behaviour of the drivers, and provides convenient utility features and debugging aids.

The HLAPI (Haptic Library API) provides high-level haptic rendering and is designed to be familiar to OpenGL API programmers. It allows significant reuse of existing OpenGL code and greatly simplifies synchronization of the haptics and graphics threads.

The HLAPI is built on top of HDAPI and provides a higher level control of haptics than HDAPI, at the expense of flexibility comparison to HDAPI. The HLAPI is primarily designed for ease of use to those well versed in OpenGL programming. HLAPI follows the traditional graphics techniques as found in the OpenGL API, hence adding haptics to an object is a fairly trivial process that resembles the model used to represent the object graphically. The HLAPI also provides event handling for ease of integration into applications. Besides, it allows the developer to command the haptic rendering pipeline from the graphics rendering loop, which makes it significantly more approachable for a developer to introduce haptic rendering to an existing graphics loop driven application.

On the other hand, HDAPI requires the developer to manage direct force rendering for the haptic device, programming rendering/collision detection algorithms and data structures. This is due to the high frequency of force refreshes required for stable closed-loop control of the haptic device.

HDAPI and HLAPI can be used together. Developers can leverage pieces of functionality from HDAPI to augment an HLAPI program. HDAPI must be used to initialize and configure the haptic device handle (HHD). The HHD from HDAPI is used by the HL haptic rendering context (HHLRC) to interface with the haptic device. This allows the developer to control behaviours for the haptic device that will be realized by the haptic rendering library.

3.4. Creating Haptic Environments

In order to render stable haptic feedback, the HD servo loop runs at approximately 1KHz, 30 times faster than the graphics loop. The servo loop refers to the tight control loop used to calculate forces to send to the haptic device. In order to maintain such a high update rate, the servo loop is generally executed in a separate, high-priority thread. This thread is referred to as the servo loop thread.

OpenHaptics compute the force interaction form considering the position of the device and *end-effector* (the end of the kinematic chain of the device you hold in your hand) and its relationship to objects in a virtual environment. When zero force is being rendered, the motion of the device end-effector should feel relatively free and weightless, so the device generates a force to counteract against its own weight. As the user moves the device's end-effector around the virtual environment, OpenHaptics commands forces into the servo loop thread. This allows the user to effectively feel the shape of objects in a virtual environment. Nevertheless, this forces can vary to produce different effects, so the forces can make an object surface feel hard, soft, rough, slick, sticky, etc. Furthermore, the forces generated by the haptics rendering can be used to produce an ambient effect. For instance, inertia and viscosity are common ways to modify the otherwise free space motion of the user in the environment. Another common use of forces in a virtual environment is to provide guidance by constraining the user's motion while the user is selecting an object or performing a manipulation.

3.4.1. Force rendering

The force vector is the unit of output for a haptic device. There are three main classes of forces that OpenHaptics can simulated: motion dependent, time dependent, and a combination of both.

A motion dependent force means that is computed based on the motion of the haptic device. HLAPI includes functions to generate types of motion dependent force as:

- **Spring**: it is the most common, versatile and simple to use force. A spring force can be computed by applying the *Hooke's Law* ($\vec{F}=k\vec{x}$, where k is a stiffness constant and x is a displacement vector)

- **Damper**: is a common metaphor in haptics rendering. It can be defined as a force that reduces vibration since it opposes motion. In general, the strength of a damper is proportional to end-effector velocity. The standard equation is $\vec{F}=-b\vec{v}$, where b is the damping constant and v is the velocity end-effector. The force is always pointing in the opposite direction of motion.

- **Friction**: a number for forms of friction can be simulated with the haptic device

    - *Coulombic friction*: simply opposes the direction of motion with a constant magnitude friction force, according to the formula $\vec{F} = -c\|\vec{v}\|$, where C is the friction constant and v is the velocity of the end-effector. It is implemented using a damping expression with a high damping constant and a small constant force clamp. Coulombic friction helps to create a smooth transition when changing directions, since friction will be proportional to velocity for slow movement.

    - *Viscous friction*: similar to Columbus friction, using a low damping constant and a high clamp value.

    - *Static and dynamic friction*: also referred as stick-slip friction, this friction model switches between no relative motion and resisted relative motion. The friction force is always opposing lateral motion along a surface, and the magnitude of the friction force is always proportional to the perpendicular (normal) force of contact

- **Inertia**: force associated to a moving mass and a acceleration, not necessary related to the device movement. It is easily calculated using Newton's Law $\vec{F} = m\vec{a}$.

On the other hand, a time dependent force means that it is computed as a function of time. OpenHaptics can generate time dependent forces such as:

- **Constant**: a force with a fixed magnitude and direction. For instance, it is commonly used for gravity compensation such as to make the end-effector feel weightless.

- **Periodic**: it applies a pattern that repeats over time. Patterns include saw tooth, square or sinusoid. A period force is described by a time constant (period), an amplitude to determine the peak of the cycle, and a force direction.

- **Impulses**: a force vector that is instantaneously applied. In practice, an impulse with a haptic device is best applied over a small duration of time in the servo loop.

3.4.2. Contact and Constraints

Simulating contact with a virtual object amounts to computing forces that resist the device end-effector from penetrating the virtual object's surface. One approach to simulate this interaction is through the concept of a *proxy* that follows the transform of the device end-effector in the virtual environment.

The geometry for the proxy is typically a point, sphere or collection of points. If it is a point, it is sometimes referred to as the *SCP*, Surface Contact point. SCP attempts to follow the end-effector position but is constrained to be on the surface of the object. In free space the SCP is at the end-effector position as shown in Illustration 16. When touching an object the SCP can be calculated by moving the last SCP towards the end-effector position without violating the surface. The force is calculated by simulating a spring stretched from the end-effector position to the SCP. t_1 shows penetration into the object. t_2 shows further penetration. The spring is stretched longer and hence the user will feel greater resistance.

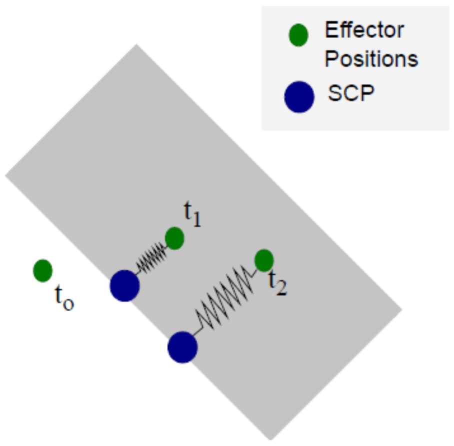


Illustration 16 : proxy concept to simulate surface contacts

In addition, the proxy should respect spatial coherence of contact. As a result of computing a constrained proxy transform, forces can be determined that will impede the motion of the haptic device end-effector from further penetrating the contacted surface. This technique of maintaining a constrained proxy can be applied to feeling all kinds of geometry, such as implicit surfaces, polygonal meshes, and voxel volumes. It can also be applied to feeling geometric constraints, such as points, lines or combinations of both.

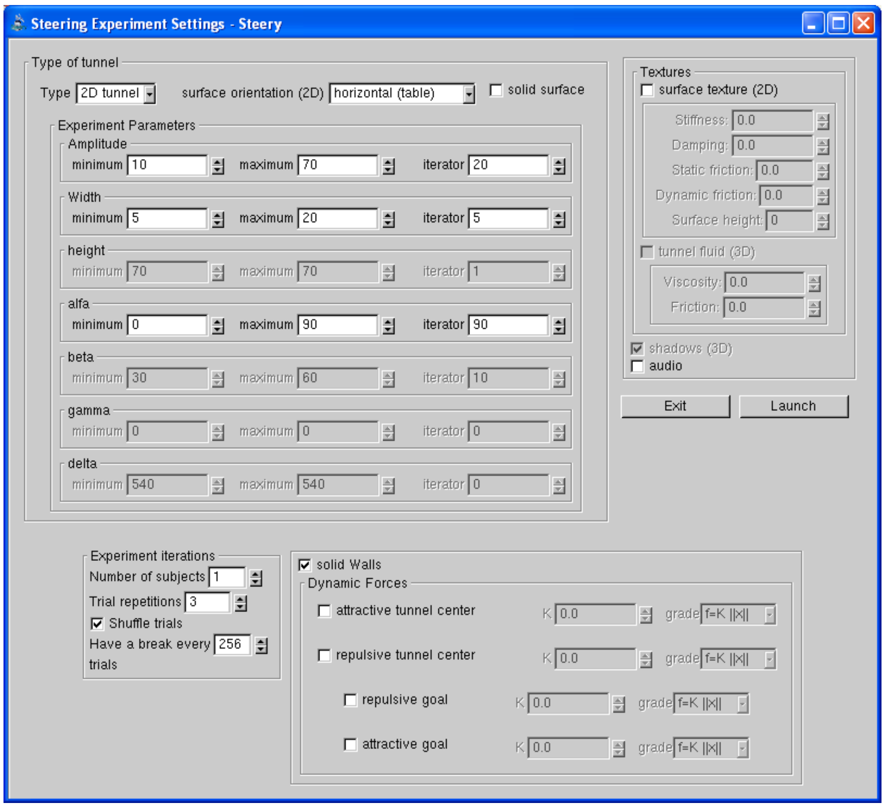
Chapter 4 - Steery project development

In this chapter, the steps to develop the system will be presented. First the graphic layer will be described and then the software architecture with its classes, haptic features and functionalities will be presented.

To explain Steery development, we have to understand first the basic idea of a Steering Law experiment. In this kind of experiment, the designer sets up different parameters such as tunnel width, height, inclination and so on. Each of those parameters have been assigned with different values, generating a number of trials in a factorial design that the experiment subjects will have to perform. By factorial design we define the generation of all the possible combinations of the different levels (values) of each factor (width, height, etc.). Then, every subject will perform all the trials, and the designer will recollect the trials' data and analyse it.

4.1. The interface development

OpenHaptics toolkit has been created for use with Microsoft Win32 API and GLUT. On one side, MS Win32 API has the advantage to provide shorter source code and simpler GUI prototypes. However, OpenGL is a standard specification defining a cross-language cross-platform API. Seeing that, OpenGL API was chosen in this project to develop the graphic interface layer (see Illustration 17).


Steering Experiment Settings - Steery

Type of tunnel
Type: 2D tunnel — surface orientation (2D): horizontal (table) — ☐ solid surface

Experiment Parameters

	minimum	maximum	iterator
Amplitude	10	70	20
Width	5	20	5
height	70	70	1
alfa	0	90	90
beta	30	60	10
gamma	0	0	0
delta	540	540	0

Textures
☐ surface texture (2D)
Stiffness: 0.0
Damping: 0.0
Static friction: 0.0
Dynamic friction: 0.0
Surface height: 0
☐ tunnel fluid (3D)
Viscosity: 0.0
Friction: 0.0
☑ shadows (3D)
☐ audio

Experiment iterations
Number of subjects: 1
Trial repetitions: 3
☑ Shuffle trials
Have a break every 256 trials

☑ solid Walls
Dynamic Forces

	K	grade
☐ attractive tunnel center	0.0	f=K ||x||
☐ repulsive tunnel center	0.0	f=K ||x||
☐ repulsive goal	0.0	f=K ||x||
☐ attractive goal	0.0	f=K ||x||

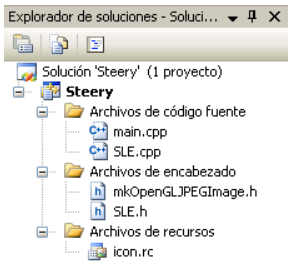
Illustration 17: Steery GLUT main window

OpenGL contains rendering commands but is designed to be independent of any window system or operating system. Unfortunately, it is impossible to write a complete graphics program without at least opening a window, using user input or other services from the operating system. Therefore, Steery uses GLUT and GLUI libraries to run the graphic user interface layer. GLUI is a C++ used interface library based on GLUT to augment its functionality. It provides controls such as buttons, checkboxes, radio buttons, spinners and so on in OpenGL applications. It is window-system independent, relying on GLUT to handle all system-dependent issues, such as window and mouse management.

The system initially shows a main window where the user configures every experiment parameter. Every time that a GUI element is changed, the GLUT loop calls a control callback function to perform the appropriate operation. For a deeper explanation of the available parameters, listboxes, and other functionalities in the graphic layer, see the enclosed Appendix II. Steery User's guide.

4.2. The SLE class

Every GUI element is linked with the Steering Law Experiment class. This class, defined and implemented in SLE.h and SLE.cpp files respectively (Illustration 18), manages all the information and operations of the ongoing experiment. In the main file, a global instance of the SLE class is declared at the beginning. A more detailed overview of the class elements and methods is explained in the Appendix I. Steery generated documentation.


```
Explorador de soluciones - Soluci...
Solución 'Steery' (1 proyecto)
  Steery
    Archivos de código fuente
      main.cpp
      SLE.cpp
    Archivos de encabezado
      mkOpenGLJPEGImage.h
      SLE.h
    Archivos de recursos
      icon.rc
```
Illustration 18: Steery file directory

As the user launches the experiment, the *SLE.init_trials()* method is called to create all the experiment trials in a factorial design. If selected, the trials of each subject are shuffled.

During the experiment, the current trial information, saved on SLE class, allows the system to display it on the screen. Trial data such as user trajectory, standard deviation or time lapse is recorded on this class.

To conclude the experiment, a settings selection window pops up to save the experiment data into a file. These GUI elements are linked to *SLE.output* structure, which are checked in the *SaveDataToFile()* function.

4.3. OpenHaptics features

Steery takes advantage of some of the features explained on Chapter 3 - OpenHaptics to simulate haptic feedback in the PHANTOM Omni device.

Steery haptics are mainly scheduled in an asynchronous callback (*touchScene*) executed in the servo loop, so any submitted callback is run immediately regardless of priority. This callback executes a HD Frame, that is, a haptic rendering pass, a block of code guaranteed to be consistent. In this function, for the selected type of experiment and settings, different forces are added to a force vector (hduVector3Dd), which is finally applied to the device.

Device button events are handled by a callback which, according to the state of the trial, it may switch during the different trial stages.

For some special features such as surface, viscosity or damping, a HL frame is used. It updates the current state of the haptic device and clears the current set of haptics primitives (shapes and effects) being rendered in preparation for rendering a new or updated set of primitives.

Every time the scene is drawn, a synchronous scheduler (*copyHapticDisplayState*) is called to obtain the device display state and calculate its position and other useful values for the current type of experiment.

Finally, the *updateworkspace* function is called either when a new trial starts or when the camera has changed. This function defines the new camera projection to fit, scale and map it into the haptics workspace, as well as the cursor. It computes the transform for going from device coordinates to world coordinates, based on the current viewing transforms.

Next, the implemented haptics features in Steery are explained.

4.3.1. Dynamic forces

Dynamic force refers to the aforementioned spring force, based on Hooke's Law $\vec{F}=k\vec{x}$. Steery has implemented four types of dynamic forces, belonging to the HD frame:

- Attractive tunnel centre: by tunnel centre we define the path that goes along the tunnel, which is at equidistant from the tunnel walls or boundaries. If selected, a force perpendicular to the tunnel is applied to the device in the asynchronous callback, pushing it to the tunnel centre.

- Repulsive tunnel centre: similarly to the previous one, it exerts a tunnel centre perpendicular force, but opposite to the tunnel centre.

- Attractive goal: each trial has a starting and an end position. If selected, a force is applied in the same tunnel direction, pushing the device towards the end position.

- Repulsive goal: if selected a force is applied pushing the device towards the starting position.

The implementation of these forces is not always trivial. The calculation of the force calculation may depend on the type of tunnel, position of the device or tunnel rotations. It differs widely to calculate the attractive goal force for a straight two-dimensional tunnel from an attractive goal curved three-dimensional torus, as it can be seen in the next Fig. 19 and 20.

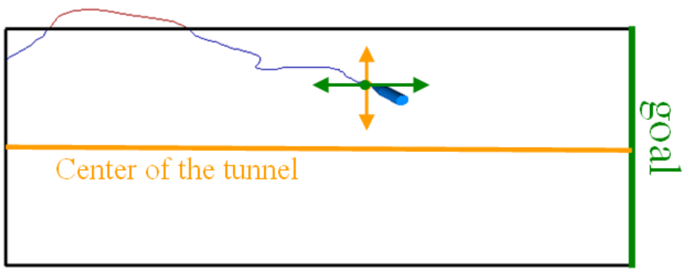


Illustration 20: basic tunnel dynamic forces

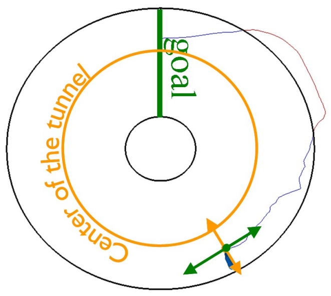


Illustration 19: curve 2D tunnel dynamic forces

To augment the possibilities of future dynamic force experiments, another functionality was added. The user can select the grade of the force function. Thus, the force is not only restricted to linear forces: constant forces $\vec{F}=k\|\vec{x}\|$, linear $\vec{F}=k\vec{x}$, quadratic $\vec{F}=k\vec{x}^2$ inverse $\vec{F}=\frac{k}{\vec{x}}$ and logarithmic $\vec{F}=k\log(\vec{x})$. Before applying the force, the callback function calls to *SLE.GradeEquation* (or *SLE.GradeEquationInCurve* for curved tunnels)

4.3.2. Solid Walls

This force behaves similarly to a dynamic force. It emulates solid walls at the sides of the tunnel by applying a force when the device cursor is farer than the width tunnel from the centre path. This force is based on the distance to the tunnel side, growing exponentially to the distance, hence emulating a soft -spongy- wall. It is also applied in the HD frame. In 3D tunnels, the cursor is locked up inside the tunnel space.

4.3.3. Surface: orientation and stiffness

For two-dimensional tunnels, the experiment can be executed in horizontal (as a table) or vertical (as a blackboard). The default position is horizontal, but if the user sets the experiment into vertical, all the system workspace is rotated 90 degrees. Besides, several haptic global variables are modified in order to behave in accordance with the new coordinates.

It is also common to perform this experiments in a horizontal surface. The user can activate a simple surface emulation, solid walls alike. In this case, the cursor is forced to stay in the Y-axis (y=0) of the device coordinate system. If vertical orientation is selected, the coordinate system is rotated as well, and the same force is applied. This force, implemented in the HD frame, grows exponentially with the Y value. It is illustrated in a picture at the Appendix II. Steery User's guide, page 72. In short, this force emulates a cursor stuck to a soft surface. For a more realistic surface emulation, the next section has been developed.

4.3.4. Surface textures

These features take advantage of the HLAPI high-level functionality. Creating a HL frame, it emulates a surface with graphic and haptics properties. These haptic properties are set according to the user selected material values of damping, stiffness, damping, static friction and dynamic friction, oscillating in range [0,1]. Besides, the surface is created in the user selected height of the device workspace.

4.3.5. Tunnel fluid

For 3D tunnels, another HL frame is created. To evaluate haptic properties in a 3D tunnel experiment, we face the problem of the uselessness of a surface in a three-dimensional environment. For that reason, in 3D tunnels Steery can only provide the subject with free space force effect feedback. This effect is sent to the device while the cursor is inside the tunnel. Two types of effects can be selected:

- Viscosity: this force is based on the current velocity of the haptic device, and it is calculated to resist the motion of the haptic device. It is calculated using the expression $\vec{F} = -k\vec{v}$, where F is the spring force, k is the user selected value (gain) and v is the velocity.

- Friction: as commented before, this force is applied according to the formula $\vec{F} = -c\|\vec{v}\|$, where the clamping value c is set by the user.

4.4. Types of tunnel

The main feature of a experiment in Steery is the type of tunnel. It defines the scenario and environment in which the subjects will realize the trials.

Mainly, tunnels can be separated in two types: two-dimensional and three-dimensional. In the first type, OpenGL generates an orthographic projection, whereas a perspective projection is generated in the second type. In an orthographic projection, the viewing volume is a rectangular parallelepiped (a box). Thus, the size of the viewing volume doesn't change from end to the other, so distance from camera doesn't affect how large an object appears. By using this projection, the tunnel maintain its actual size no matter how it is placed. However, in a 3D experiment the viewing volume is a frustum of a pyramid. Objects that fall within the viewing volume are projected toward the apex of the pyramid, where the camera is. Closer objects appear larger because they occupy a proportionally larger amount of the viewing volume than those that are farther away, in the larger part of the frustum.

Tunnels are mostly drawn with a OpenGL display lists (except 2D tunnel, due to its simplicity). A display list is a group of OpenGL commands that have been stored for later execution, improving the display loop performance. Because of tunnel physiognomy, it becomes useful to take advantage of display list, redrawing the same geometry during the whole trial. Thus, the display list is stored before the trial starts.

To control the sequence of the experiment, every trial behaves as a state machine. In the OpenGL display loop the trial condition is checked to jump between states. In the first states of the trial, the starting position is drawn, providing feedback to the user about where to start the experiment. During the trial, the device position is saved in a vector to display the whole trajectory on the screen, as well as the cursor. Once the subject reaches the end position, the system process the trial trajectory to calculate the standard deviation, OPM and MT into the SLE class.

A more concise description of every type of tunnel is described in the next point.

4.4.1. Basic tunnel

This two-dimensional straight tunnel is the most well-known tunnel in steering law experiments. It is defined by a width and height. Besides, we added a third parameter called alpha (α), in degrees, which rotates the tunnel clockwise. The size of the tunnel and camera has been defined to occupy the same number of pixels in the screen. The starting position is by default placed at the left side, so the subject has to steer the tunnel from left to right.

To calculate the standard deviation, it is needed to obtain the distance from each point of the user trajectory to the tunnel centre. This value is defined and deducted as the distance from the cursor position to the tunnel centre line, which starts in the starting position and finish in the tunnel end position (Illustration 21). To calculate the distance from a point p to a line $\bar{ab}$, we calculate the orthogonal projection of the vector $\bar{ap}$ onto the line $\bar{ab}$, generating the vector $\bar{ac}$. The euclidean distance $\|\bar{pa}\|$ is the desired value.

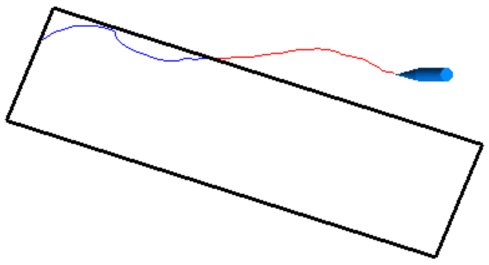


Illustration 21: basic tunnel distance

4.4.2. Circular tunnel

The investigation of curved trajectory-based tasks in graphic interfaces is becoming an incipient field. It was decided to curve the tunnel into a full circumference, becoming an annulus. The amplitude was the perimeter of the tunnel centre, and the width the thickness of the annulus. From the width (*W*) an amplitude (*A*), both annulus circumferences can be deducted as $r_1 = r - \frac{W}{2} = \frac{A}{2\pi} - \frac{W}{2}$ and $r_2 = \frac{A}{2\pi} + \frac{W}{2}$. Alpha value has been also implemented, rotating the tunnel clockwise. The starting position is placed on the right side by default. This tunnel has the limitation of clockwise steering task only.

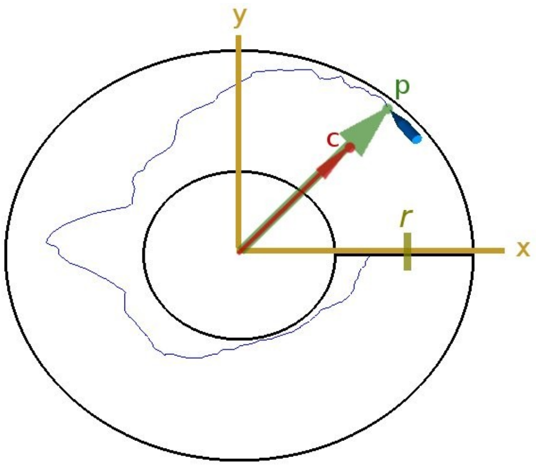


Illustration 22: torus bias

Due to its complexity, obtaining the standard deviation (or bias from the path centre) and OPM it is not as trivial as the previous tunnel and requires more calculus. The system calculates the closest point of the tunnel centre to the cursor by norming the vector to unit vector and multiplying by *r*. Given the cursor position $\vec{p}$, the tunnel centre closest point $\vec{c}$ will be $\vec{c} = \frac{\vec{p}}{\|\vec{p}\|} r$. Then, the deviation from each point of the cursor trajectory is deducted as the euclidean distance between $\vec{p}$ and $\vec{c}$ as the Illustration 22 shows.

4.4.3. Curve

This feature has been prorogued for further development. Its purpose seeks to generate free curved tunnel sketched by the user with the device cursor, and then perform the steering task. Due to the instability of the user hand drawn pulse and limitations of the task, this type of tunnel was postponed after more discussion.

4.4.4. tunnel 3D

This tunnel expand the basic tunnel to the third dimension, generating a cylinder the user can steer through. In consequence, the three degrees of freedom of the haptic device can be exploited to study its steering performance. In this type, the amplitude defines the height of the tunnel, and the width is the diameter of the cylinder base. Besides, the tunnel can be rotated to any position by the combination of two angles alpha α and beta β. Alpha rotates the tunnel in the Y-axis, and beta in the Z-axis. The centre of rotation stays in the middle of the tunnel, preventing it for moving out of the camera.

This combination of rotations can be considered as a Euler transformation, applying the Euler angles. For the implementation, custom rotation functions where used to dispense from OpenGL functions, hence having more flexibility and control over the tunnel parameters, matrices and positions. To apply a rotation in any of the three dimensions, the basic 3D rotation matrices are:

$$R_x(\gamma)\begin{bmatrix}1 & 0 & 0\\ 0 & -\cos\gamma & \sin\gamma\\ 0 & \sin\gamma & \cos\gamma\end{bmatrix} \quad R_y(\alpha)\begin{bmatrix}\cos\alpha & 0 & \sin\alpha\\ 0 & 1 & 0\\ -\sin\alpha & 0 & \cos\alpha\end{bmatrix} \quad R_z(\beta)\begin{bmatrix}\cos\beta & -\sin\beta & 0\\ \sin\beta & \cos\beta & 0\\ 0 & 0 & 1\end{bmatrix}$$

The calculus of the deviation during the trajectory is done exactly the same as the basic tunnel: distance from a point (cursor position) to a line (starting-end points). The only different is that the third dimension is included in the calculation.

4.4.5. torus 3D

Similar to the torus 2D, the toroidal shape is defined by a circle of diameter W which revolves towards a coplanar axis, generating a surface of revolution. The length of this revolving corresponds to the amplitude value A. Thus, the revolving radius is $r=\frac{A}{2\pi}$. The torus can be rotated to any position by the combination of three rotations alpha (α), beta (β) and gamma (γ). Alpha rotates the tunnel towards the Y-axis, beta does it towards Z-axis, and gamma rotates the tunnel in the X-axis. In the default position, the starting region is in the left part of the screen (negative X-axis) and the user steers the tunnel in the revolving direction of positive Y-axis, according to the right-hand rule (See Illustration 23).

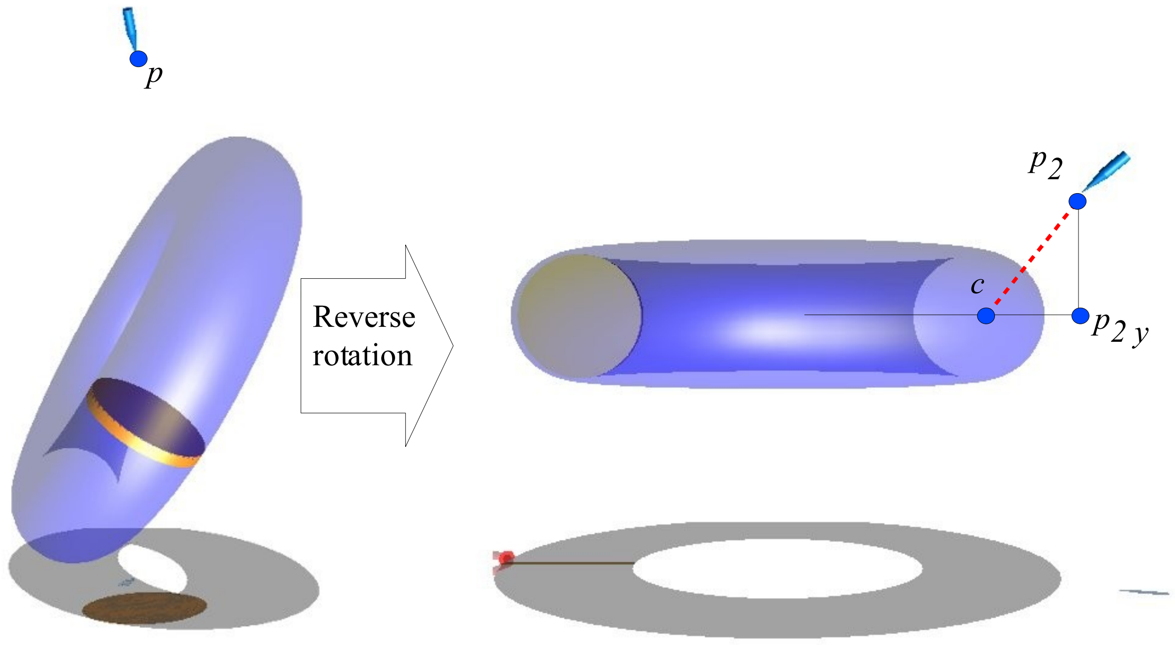


Illustration 23: torus 3D bias

As a result of this rotation, the bias calculation becomes more complex, as the Illustration 23 shows. In the cursor position callback, the position *p* is rotated backwards in the three angles in inverse order, obtaining a second position p_2 where the cursor would be in a default torus (without rotating). Then, the orthogonal projection of p_2 in the Y plane is calculated p_{2y}. Finally, we proceed as section 4.4.2. Circular tunnel to calculate the point *c*, obtaining the euclidean distance $c\overline{p}_2$, or bias.

After the first alpha tests we realized that the steering direction of the trial can be misinterpret by the experiment subject. Consequently, a new feature was added to provide the user with new feedback that would inform him about the direction of the task. This feature consisted in a small red torus situated at the left bottom corner of the screen, provided with arrows which indicate the task direction. The same rotation matrix was applied to this torus, thus being drawn in the same orientation than the tunnel (Illustration 24).

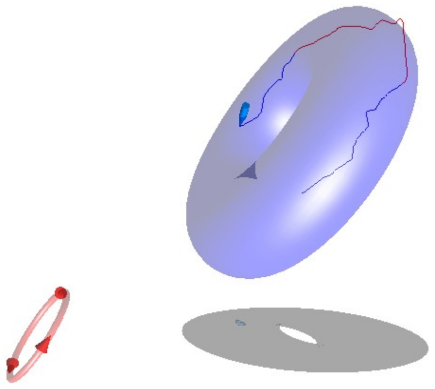


Illustration 24: torus3D direction feedback

4.4.6. Helix 3D

A helix is a space curve, defined by a simple function $(a\cos t, a\sin t, bt)$. But a volumetric helix requires more parameters to be constructed. In Steery, a 3D helix is a surface of revolution generated by a revolving circle towards a coplanar axis (torus), but for each "slice" of the torus, the circle is displaced one position into the positive direction of the coplanar axis. Besides, the circle is revolved δ degrees, where δ=[1, 1080]. The revolving radius, similar to the torus 3D, is $r=\frac{A}{2\pi}$, and *W* is the diameter of the revolved circle. The height of the helix corresponds to the parameter *h* and, likewise the torus 3D, it can be rotated to any position using the Euler angles α, β and γ. To put it briefly, in Steery a helix is defined by the parameters *A*, *W*, *h*, α, β, γ and δ. Thus, the helix is defined by

$$S(\theta)=(r\cos\theta, h\frac{\theta}{\delta}, r\sin\theta)=(S_x, S_y, S_z)$$

4.4.6.1 Distance to helix solution

The calculation of the closest distance of a point to a helix is nothing but trivial. In fact, finding a solution resulted a mathematical and programming challenge.

Given a curve defined in an analytic representation $S(\theta)$ and a point *P*, the distance *D* from the point to the curve is defined as the euclidean distance from the point to the curve, $D=\sqrt{(P_x-S_x)^2+(P_y-S_y)^2+(P_z-S_z)^2}$. We derive the equation to find the solution which satisfy $\frac{\partial D}{\partial\theta}=0$, obtaining the minimum distance from the curve to the point. However, the polar nature of the helix definition prevents any algebraic solution for the derivative.

$$\frac{\partial D}{\partial\theta}=\frac{\frac{-2\mathrm{h}(P_y-\frac{h\theta}{\delta})}{\delta}-r\cos\theta+۲\mathrm{r}(P_x-r\cos\theta)\sin\theta}{۲\sqrt{P_z+(P_y-\frac{h\theta}{d})^2+(P_x-r\cos\theta)^2-r\sin\theta}}=0 \quad \Longrightarrow \quad \text{Non-algebraic solution}$$

Therefore, a programmatic approach was implemented. Given a cylinder of radius *r* which contains the helix, we calculate the point *B*, the closest point of the enclosing cylinder to the current position *P*. Both points have the same height, $B_y=P_y$. Then, we calculate the vertical projection of *B* in the helix, called *C*, and the horizontal projection of *B* in the helix, *D*. From C to D a helix segment is obtained, which is covered in N segments (points) looking for the closest point to *P*. This method is shown in the Illustrations 25 and 26.

$$\varphi = \arctan\left(\frac{P_z}{P_x}\right)$$
$$B = (r\cos\varphi, P_y, -r\sin\varphi)$$
$$C = (B_x, \varphi\frac{h}{\delta}, B_z)$$
$$\omega = \delta\frac{B_y}{h}$$
$$D = (r\cos\omega, B_y, -r\sin\omega)$$

Illustration 25: distance to helix approach pseudo-code

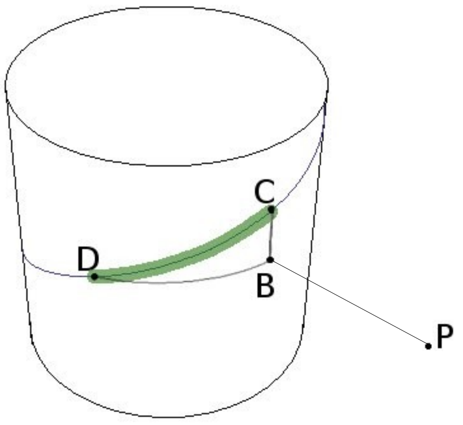


Illustration 26: distance to helix approach graphic

4.5. Audio feedback

The experiment designer can activate the audio feedback checkbox in the main window. This feature has been implemented with the *Playsound* function from *windows* library. If selected, the system plays an asynchronous error sound loop during the trial when the cursor gets out of the tunnel.

4.6. Lights and shadows

For every 3D tunnel, the surface of the tunnel object (cylinder, tunnel, helix, …) is displayed with an alpha channel of 0.5, hence the cursor can be seen even inside the tunnel. Tunnels are displayed in a light model with a diffuse and a directional light to give a more realistic three-dimensional feedback. Then, the tunnel material is set with coefficients of specular and shininess properties. To combine properly the alpha channel, illumination model and material properties, a blending function is used. These parameters are set in the *initGL* function.

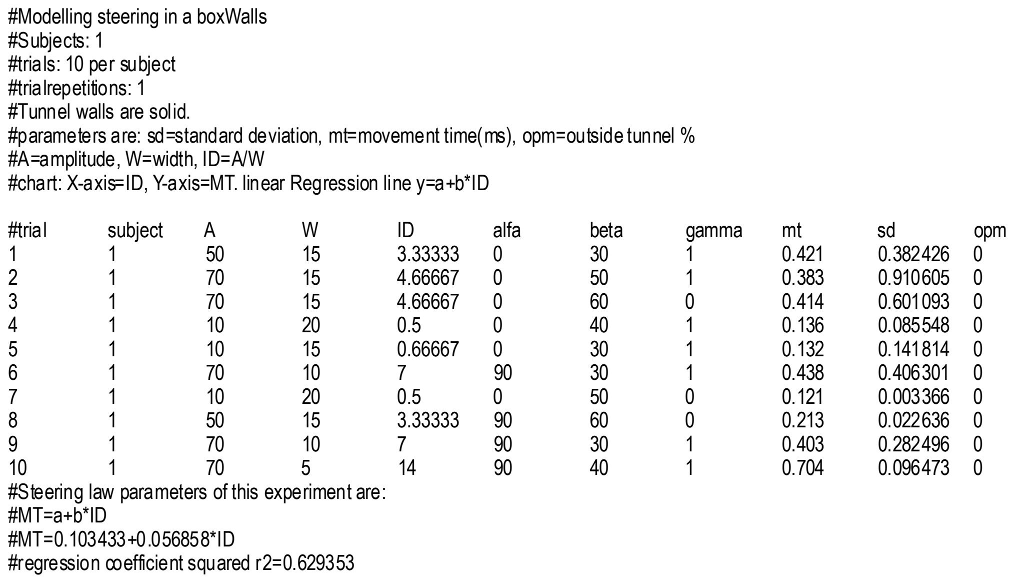
In three-dimensional tunnels, the tunnel, start position and cursor shadows are drawn to provide the user with more feedback to his depth perception. OpenGL does not support shadows directly, so the Phaetos [21] projection algorithm is used to emulate this shadows. The current matrix stack is multiplied by a projection matrix, and the object is printed a second time. The projection matrix depends on the position of the light source $\vec{L}=(L_x, L_y, L_z)$, the normal vector of the plane $\vec{n}=(n_x, n_y, n_z)$ in which the shadow is projected and a point of the plane $\vec{E}=(E_x, E_y, E_z)$. From this three parameters we derive the values *d* and *c*, and obtain the shadow projection matrix *M* as follows:

$$d=\vec{L}\vec{n} \qquad d=\vec{E}\vec{n}-d \qquad M=\begin{bmatrix} L_x n_x + c & L_x n_y & L_x n_z & -cL_x - dL_x \\ L_y n_x & L_y n_y + c & L_y n_z & -cL_y - dL_y \\ L_z n_x & L_z n_y & L_z n_z + c & -cL_z - dL_z \\ n_x & n_y & n_z & -d \end{bmatrix}$$

The light source is positioned in an “infinite” elevation (0, 1000,0) far from the floor, of which normal vector is (0,1,0).

4.7. Output data

The results of the experiment can be saved into a file. Using C++ fstreams, Steery saves the data into a plain text file (TXT) or Comma-separated Value file (CSV), according to the user selection. In case of plain text, lines of comments start with a pound sign #, for a more comfortable data processing (Illustration 27).



```
#Modelling steering in a boxWalls
#Subjects: 1
#trials: 10 per subject
#trialrepetitions: 1
#Tunnel walls are solid.
#parameters are: sd=standard deviation, mt=movement time(ms), opm=outside tunnel %
#A=amplitude, W=width, ID=A/W
#chart: X-axis=ID, Y-axis=MT. linear Regression line y=a+b*ID

#trial  subject  A    W    ID       alfa  beta  gamma  mt     sd        opm
1       1        50   15   3.33333  0     30    1      0.421  0.382426  0
2       1        70   15   4.66667  0     50    1      0.383  0.910605  0
3       1        70   15   4.66667  0     60    0      0.414  0.601093  0
4       1        10   20   0.5      0     40    1      0.136  0.085548  0
5       1        10   15   0.66667  0     30    1      0.132  0.141814  0
6       1        70   10   7        90    30    1      0.438  0.406301  0
7       1        10   20   0.5      0     50    0      0.121  0.003366  0
8       1        50   15   3.33333  90    60    0      0.213  0.022636  0
9       1        70   10   7        90    30    1      0.403  0.282496  0
10      1        70   5    14       90    40    1      0.704  0.096473  0
#Steering law parameters of this experiment are:
#MT=a+b*ID
#MT=0.103433+0.056858*ID
#regression coefficient squared r2=0.629353
```

Illustration 27: output TXT file sample

It was also implemented a feature to show the results in a graphic chart where X-axis corresponds to Index of Difficulty (A/W) and Y-axis is the Movement Time. This chart is displayed with an orthographic projection in OpenGL. Besides, the user can save the chart into a JPEG file using the mkOpenGLJPEGImage [22] library.

Chapter 5 - Experiments, results and discussion

Steery has been built to facilitate researchers the investigation of steering law experiments. Accordingly, several experiments were performed by different laboratory members for their own investigation.

5.1. Verification of linear steering in 3D space

Realized by Hironori Ishiyama [2] for his Bachelor thesis, this experiment sought to study Steering task user range for stroke, orientation and tilt, hence examining it and contribute to create a future interface.

The experiment used 3D tunnel type, based on a cylinder. The parameter settings where:

- Width W: 5, 15, 25, and 35 pixels. x4
- Amplitude A: 50, 75 and 100 pixels. x3
- Azimuth (α) and tilt (β): from 0 to 315º incrementing by 45º. x8 x8
- 12 subjects.

Combined in factorial design, each subject had to perform 768 trials.

Analysing each parameter separately, error rate decreases as the width increases. In 5px width, error rate is 95% but in 25 and 35px it decreases until 20% or less. Referring to angles α and β, error rate is lower than the direction. Results suggested that, on a diameter of three-dimensional space operations, perpendicular tilts (β) of 0, 90,180 and 270, and an azimuth of 180 and 270 were valid for the experiment. These results are expected to help and encourage the development of a new pen interface.

5.2. Verification of curved steering in 3D space

This experiment was performed by Onishi Yoshinobu [3] for his Bachelor thesis, as well. Similarly to the previous experiment, this one made use of the torus 3D tunnel type with 12 subjects to study curved trajectories in three dimensions. The experiment settings where:

- Width W: 20, 30 and 40px. x3
- Amplitude A : 160, 200 and 240px. x3

- Azimuth (α) and tilt (β) and γ from 0 to 135º incrementing by 45º. x4 x4 x4

Combined, each subject had to perform 576 trials.

The torus tasks were found more difficult to perform in the cases where it was steered vertically rather than horizontally. Besides, it was difficult to adjust the vertical position of the pointer to look sideways. The results obtained are expected to be useful in future operations to improve the usability of interfaces and modelling.

5.3. Boxwalls experiment

To test the extensibility of Steery, a new type of tunnel (therefore, a experiment) was implemented. It consist of a virtual room with floor, lateral and front walls, thus nicknamed as *boxwalls*. A straight 2D tunnel is displayed in a wall, and the subject has to steer through it. Every trial is perform two times: first, in the front wall, and next in the “floor” wall. Walls were provided by haptic feedback, implemented in a HL frame. The available parameters where: amplitude A of the tunnel, width W, rotation α, and camera angle β.

In this scenario, we studied the influence of the camera to compare the front wall and floor wall tasks. This experiment and its results are thorough and widely explained in the report enclosed to this document, Appendix III. Boxwalls experiment report. The results of the study showed that the rejection of wrong trials produced high error rates. Additionally, the steering law correlation was lower than expected in this type of experiment, which commonly tends to be $R^2>0.8$. The user's lack of perception presented to be the main disadvantage of the experiment, poorly balanced with graphic feedback. As a consequence, Steering Law had to be refrained to be used in this scenario.

Chapter 6 - Conclusions and future work

We wanted to realize a software platform where the HCI researchers could deploy and perform they experiments in an easy and quick methodology.

According to the experiments and results explained in the previous chapter, the application succeed to reach its goals. Three different researchers could carry out their investigations with Steery in a swift way, disposing of any need of application re-programming.

Another successful feature lies in the potential, flexibility and high degree of configuration of the application: the variety of settings permits the designer to construct a wide an heterogeneous collection of experiments.

Steery has been proved to be a good solution to accelerate a Steering experiment design. Future works may be done by students and researchers augmenting the functionality and configuration of the application. Specially in three-dimensional experiments, where a proper and more thorough model hasn't been demonstrated yet.

Besides, it would be interesting to study, experiment and compare the haptics texture emulation with real world textures such as wood, concrete, metal, rubber, and so on. This would allow future application developments with more realistic objects in the virtual world, where the user can feel, distinguish and discriminate between different textures.

Bibliography

[1] SensAble Technologies, Inc. 15 Constitution Way, Woburn, MA 01801 www.sensable.com

[2] Hironori Ishiyama. Verification of linear steering with three-dimensional space (三次元空間上での直線のステアリング操作の検証) 4rd Ren Laboratory Symposium. Vol.10, 2009. pp.1

[3] Onishi Yoshinobu. Verification of steering round on three-dimensional space (三次元空間上での円形のステアリング操作の検証) 4rd Ren Laboratory Symposium. Vol 10, 2009. pp.1

[4] Doxygen documentation generator. Dimitri van Heesch. http://www.doxygen.or

[5] Fitts, P.M. The information capacity of the human motor system in controlling the amplitude of movement. Journal of Experimental Psychology , 47, 381-391(1954)

[6] Accot, J. and S. Zhai. Beyond Fitts' Law: Models for trajectory-based HCI tasks. ACM CHI. P 295-302.(1997)

[7] MacKenzie, I., Buxton, W.: Extending Fitts' law to two-dimensional tasks. In: ACM CHI, pp. 219- 26 (1992)

[8] Kattinakere, R., Grossman, T., Subramanian, S.: Modelling steering within above-the-surface inter- action layers. In: ACM CHI, pp. 317-26 (2006)

[9] Accot, J., Zhai, S.: Refining Fitts Law models for bivariate pointing. In ACM CHI, pp. 193-200 (2003)

[10]Ahlström, D.: Modelling and improving selection in cascading pull-down menus using Fitts' law, the steering law and force fields. In: ACM CHI, pp. 61-70 (2005)

[11]Campbell, C., Zhai, S., May, K., Maglio, P.: What You Feel Must Be What You See: Adding Tactile Feedback to the Trackpoint. In: INTERACT, pp. 383-390 (1999)

[12]Dennerlein, J., Martin, D., Hasser, C.: Force- feedback improves performance for steering and combined steering-targeting tasks. In: ACM CHI, pp. 423-429 (2000)

[13]Forsyth, B., MacLean, K.: Predictive Haptic Guidance: Intelligent User Assistance for the Control of Dynamic Tasks. IEEE Transactions on Visualization and Computer Graphics 12(1), 102-113 (2006)

[14]XingDong Y., Pourang I., Boulanger P., Bischof W.: A Model for Steering with Haptic-Force Guidance. Proceedings of the 12th IFIP TC 13 International Conference on HCI: Part II

[15]Shneiderman, Ben (Ed.). Sparks of Innovation in human-computer interaction. Ablex Publishing Corporation. Norwood (1993) pp. 13-17

[16]Booth, Paul. An Introduction to Human-Computer Interaction. Hove, UK Lawerence Erlbaum Associates. (1989) pp. 44-49

[17]Buxton, Bill (2003). Human Input to Computer Systems: Theories, Techniques and Technology. Draft in the internet http://www.billbuxton.com/inputManuscript.html

[18] OpenHaptics toolkit 3.0 Programmer's Guide. Sensable Technologies, Inc. Jan 2009

[19] OpenHaptics toolkit 3.0 API Reference Manual. Sensable Technologies, Inc. Dec 2008

[20] OpenGL Programming Guide, 7th edition. Chapter 7 (1996)

[21]Shadow projection in OpenGL. A mathematical explanation of implementing shadow projection. Phaetos. 2003. http://www.devmaster.net/articles/shadowprojection

[22]MkOpenGLJPEGImage library. Michael Kennedy, 2000. http://mkennedy.101main.com

Appendix I. Steery generated documentation

Steery
release candidate 1

Generated by Doxygen 1.6.3

Tue Apr 13 10:33:27 2010

Contents

1 Steery platform

Author
Alejandro Tatay Pascual

Date
April-2010

1.1 Introduction

Steery is a Steering Law Experiment platform. Steery is suitable for research on trajectory-based interactions in a variety of environments. Steery is designed to work with force-feedback haptic devices. It works with any haptic device of Sensable Technologies1 PHANTOM product line. Besides, Steery provides a different type of environment by using the haptic properties of these devices. Steery is extensible. It is designed to be augmented by developing new types of experiments and functionality. Steery is an academic software developed for Ren Laboratory and its members. Any external use should be reported to Kochi University of Technology

2 Class Documentation

2.1 _outputParameters Struct Reference

Output settings of the experiment.
#include <SLE.h>
Public Attributes

- char file [50]
- int fileType
- int amplitude
- int width
- int id
- int sd
- int opm
- int mt
- int alfa
- int beta
- int gamma
- int delta
- int height

- oat minID
- int maxID
- DWORD maxMT
- DWORD minMT

2.1.1 Detailed Description

Output settings of the experiment. Structure to control the output file and output chart.

2.1.2 Member Data Documentation

2.1.2.1 int _outputParameters::alfa

If enabled, SLE.trial[].parameter[3] will be saved for all trials

2.1.2.2 int _outputParameters::amplitude

If enabled, SLE.trial[].parameter[0] will be saved for all trials

2.1.2.3 int _outputParameters::beta

If enabled, SLE.trial[].parameter[4] will be saved for all trials

2.1.2.4 int _outputParameters::delta

If enabled, SLE.trial[].parameter[6] will be saved for all trials

2.1.2.5 char _outputParameters::file[50]

Full path output file name

2.1.2.6 int _outputParameters::fileType

File type. If 0, output file will be a plain text file (txt). If 1, output will be saved as a comma-separated values file (csv).

2.1.2.7 int _outputParameters::gamma

If enabled, SLE.trial[].parameter[5] will be saved for all trials

2.1.2.8 int _outputParameters::height

If enabled, SLE.trial[].parameter[2] will be saved for all trials

2.1.2.9 int _outputParameters::id

If enabled, Index of difficulty will be saved for all trials

2.1.2.10 int _outputParameters::maxID

maximum Index of Difficulty value. Useful for drawing the X-axis higher boundary of the chart

2.1.2.11 DWORD _outputParameters::maxMT

maximum Movement Time value. Useful for drawing the Y-axis higher boundary of the chart

2.1.2.12 oat _outputParameters::minID

minimum Index of Difficulty value. Useful for drawing the X-axis lower boundary of the chart

2.1.2.13 DWORD _outputParameters::minMT

minimum Movement Time value. Useful for drawing the Y-axis lower boundary of the chart

2.1.2.14 int _outputParameters::mt

If enabled, Movement Time will be saved for all trials

2.1.2.15 int _outputParameters::opm

If enabled, OPM will be saved for all trials

2.1.2.16 int _outputParameters::sd

If enabled, Standard Deviation will be saved for all trials

2.1.2.17 int _outputParameters::width

If enabled, SLE.trial[].parameter[1] will be saved for all trials
The documentation for this struct was generated from the following file:

• Visual Studio 2005/Projects/Steery/Steery/SLE.h

2.2 _SLE Class Reference

Steering Law Experiment class.
#include <SLE.h>

Public Member Functions

- _SLE (void)
class constructor
- ~_SLE (void)
class destructor
- void init_trials ()
Initialize trials.
- int trialsNumber ()
Number of trials.
- void ResetTrialsNumber ()
set trialsN to zero
- int genRandomNum (int size)
Generate random number.
- void shuf eTrials ()
Shuf e experiment trials.
- double distPoint2Point (hduVector3Dd p1, hduVector3Dd p2)
Distance between points.
- bool isCrossed (hduVector3Dd lastProjPos, hduVector3Dd currProjPos, hduVector3Dd p)
point is crossed
- bool isPointOnLine (hduVector3Dd p, hduVector3Dd b0, hduVector3Dd b1)
Point p is on the b0-b1 line segment.
- hduVector3Dd pointProj2Line (GLdouble p[ ], GLdouble a[ ], GLdouble b[ ])
Projected point in line.
- hduVector3Dd pointProj2Circle (GLdouble p[ ], GLdouble a[ ], GLdouble A)
Projected point in circle.
- hduVector3Dd vectorProj (hduVector3Dd a, hduVector3Dd b)
Vector projection.

- hduVector3Dd pointProj2helix (hduVector3Dd pos, int HSlices, hduVector3Dd *torusProjPerp, hduVector3Dd *CC, hduVector3Dd *DD)
Helix projection.
- hduVector3Dd getNormal (double x1, double y1, double z1, double x2, double y2, double z2, hduVector3Dd last)
calculates the normal vector of the plane
- hduVector3Dd GradeEquation (hduVector3Dd x, int grade, double K)
Calculates the equation vector of the dynamic force.
- hduVector3Dd GradeEquationInCurve (hduVector3Dd start, hduVector3Dd end, int grade, double K)
Calculates the final force to be applied in curved tunnels.
- string int2string (int i)
Converts integer to a C string.
- string double2string (double i, int decimals)
Converts double to a C string with exact decimals.
- double getStandardDeviation (std::vector< hduVector3Dd > Traj, hduVector3Dd startPosition, hduVector3Dd endPosition)
calculates standard deviation of a trial
- double getOPM (std::vector< hduVector3Dd > Traj, hduVector3Dd startPosition, hduVector3Dd endPosition)
calculates OPM of a trial
- bool LinRegress (double *a, double *b, double *r)
Linear regression function.
- bool equalD (double x, double y)
compares 2 similar doubles with precision error=0.000001
- void MatrixNxNMult (HDdouble A[3][3], HDdouble B[3][3], HDdouble Res[3][3])
Multiplication of 3x3 Matrix, Res=A*B.
- void MatrixPerPointMult (HDdouble A[3][3], HDdouble *p, HDdouble *Res)
Scalar multiplication of a matrix A and a sacalar p gives a product matrix Res=p*A.
- void RotatePoint3D (GLdouble alfa, GLdouble beta, GLdouble gamma, GLdouble p[3], GLdouble p2[3])
Rotation of a point in 3D space.

- void RotateMatrix (GLdouble alfa, GLdouble beta, GLdouble gamma, GLdouble M[16], GLdouble M2[16])
  Matrix rotation.
- void MatrixMult (GLdouble *A, GLdouble *B, GLdouble *Res, int N)
  Multiplication of NxN matrixes Res=A*B.

Public Attributes

- bool paramEnableType [7][7]
- _trial * trials
- int currentTrial
- SLEtype type
- SLESettings settings
- _outputParameters output

2.2.1 Detailed Description

Steering Law Experiment class. Structure to control the output file and output chart.

2.2.2 Constructor & Destructor Documentation

2.2.2.1 _SLE::_SLE (void)

class constructor
Initialize the paramEnableType array.

2.2.2.2 _SLE::~_SLE (void)

class destructor
It frees trials array memory

2.2.3 Member Function Documentation

2.2.3.1 double _SLE::distPoint2Point (hduVector3Dd p1, hduVector3Dd p2)

Distance between points.
Euclidean distance between 2 points in the space

Parameters
p1 starting point in a 3D space.
p2 endinf point in a 3D space

Returns
distance(p1,p2)

2.2.3.2 string _SLE::double2string (double i, int decimals)

Converts double to a C string with exact decimals.
Using iostream library converts a decimal double value, returning a C standard string with the desired number of decimals

Parameters
i double
decimals number of decimals

Returns
double in casted string

2.2.3.3 int _SLE::genRandomNum (int size)

Generate random number.

Parameters
size maximum random value

Returns
random number

2.2.3.4 hduVector3Dd _SLE::getNormal (double x1, double y1, double z1, double x2, double y2, double z2, hduVector3Dd last)

calculates the normal vector of the plane
Given 3 points 1, 2 and last, calculates the normal vector of the plane defined by these three points.

Parameters
x1 first dimension value of first point
y1 second dimension value of first point
z1 third dimension value of first point
x2 first dimension value of second point
y2 second dimension value of second point
z2 third dimension value of second point
last third point

Returns
plane normal vector

2.2.3.5 double _SLE::getOPM (std::vector< hduVector3Dd > Traj, hduVector3Dd startPosition, hduVector3Dd endPosition)

calculates OPM of a trial
calculates the Out-of-the-Path Movement percentage of a trial trajectory through the tunnel. Percentage is calculated by the proportion of points of the trajectory out of the current trial tunnel, from the total points of the trajectory

Parameters
Traj user trajectory vector of points
startPosition start position of the tunnel trial
endPosition end position of the tunnel trial

Returns
trial OPM

2.2.3.6 double _SLE::getStandardDeviation (std::vector< hduVector3Dd > Traj, hduVector3Dd startPosition, hduVector3Dd endPosition)

calculates standard deviation of a trial
calculates the standard deviation of a trial trajectory through the tunnel. SD is calculated according to the formula

N
2 s = 1 (x - x)
i N
i =1

Parameters
Traj user trajectory vector of points
startPosition start position of the tunnel trial
endPosition end position of the tunnel trial

Returns
trial standard deviation

2.2.3.7 hduVector3Dd _SLE::GradeEquation (hduVector3Dd x, int grade, double K)

Calculates the equation vector of the dynamic force.
According to the Force grade, a proportional force is returned. Multiplies the initial value x per K and applies the corresponding equation.

Parameters
x initial vector of the force
grade Equation grade. If 0, a force K is applied in the direction of the force, constant value

If 1, equation is K*x, linear If 2, equation is quadratic, K*x 2 If 3, equation is K/x If 4, rquation is K*log(x), logarithmic

Parameters
K constant K

Returns
force final vector

2.2.3.8 hduVector3Dd _SLE::GradeEquationInCurve (hduVector3Dd start, hduVector3Dd end, int grade, double K)

Calculates the final force to be applied in curved tunnels.
Depending to the current position angle, the dynamic force to be applied is different. This function calculates this force extrapolating it from the current position angle, start position, end position, grade and type of tunnel. Only works for torus3D, torus2D and helix3D tunnels

Parameters
start start position of the tunnel

end end position of the tunnel
grade equation grade of the force
K constant K

Returns
force final vector

2.2.3.9 void _SLE::init_trials ()

Initialize trials.
This function is called after setting up the the experiment. allocates dinamically trials memory and construct the parameters in a factorial design.
cp=current parameters

2.2.3.10 string _SLE::int2string (int i)

Converts integer to a C string.
Using iostream library converts a decimal integer value, returning a C standard string

Parameters
i integer

Returns
integer in casted string

2.2.3.11 bool _SLE::isCrossed (hduVector3Dd lastProjPos, hduVector3Dd currProjPos, hduVector3Dd p)

point is crossed
Calculates if a point is situated in the space between two other points, by calcujating the orthogonal projection of the point onto the line defined by the other two points.

Parameters
lastProjPos initial point of the line segment
currProjPos ending point of the line segment
p point to be calculated

Returns
true if is crossed. False if not

2.2.3.12 bool _SLE::isPointOnLine (hduVector3Dd p, hduVector3Dd b0, hduVector3Dd b1)

Point p is on the b0-b1 line segment.

Parameters
p point to be checked if it is on the line or not
b0 starting point of the line segment
b1 ending point of the line segment, different from b0

Returns
true if p is in the line segment. False if not

2.2.3.13 bool _SLE::LinRegress (double * a, double * b, double * r)

Linear regression function.
Calculates linear regression from all trials, according to the formula y (x) = a + bx, for n samples. The following assumes the standard deviations are unknown for x and y. X values correspond to trial Index of Difficulty (ID) and Y-values to Movement Time (MT).

Parameters
a y-intercept value of the linear regression function, point where the function intercepts the Y-axis.
b slope of the linear regression function
r correlation coefficient of the linear regression

Returns
true if the calculation was correct. It will return false if trials number is smaller than 3. Or if the slope b is infinite, so y-intercept a value does not exist

2.2.3.14 hduVector3Dd _SLE::pointProj2Circle (GLdouble p[ ], GLdouble a[ ], GLdouble A)

Projected point in circle.
Calculates the closest point of the circumference to the point p.

Parameters

p	point to be projected
a	center of the circumference
A	circumference perimeter

Returns
orthogonal projection point

2.2.3.15 hduVector3Dd _SLE::pointProj2helix (hduVector3Dd pos, int HSlices, hduVector3Dd * torusProjPerp, hduVector3Dd * CC, hduVector3Dd * DD)

Helix projection.
In the helix experiment, returns the closest point of the center of the helix to the current position. Helix equation is {$S_x = r * \cos(angle)$, $S_y = trialHeight_{z\,trialDelta_{angle}}$, $S_z = r * \sin(angle)$}
To find the minimum distance from current position to this equation:

$$Dist(pos, helix) = (pos_x - S_x)^2 + (pos_y - S_y)^2 + (pos_z - S_z)^2$$

we derivate the equation (without v) and find the solution which satisfy the derivate=0 but this derivation from a polar equation is unaffordable, because there is no lineal solution that satisfy $2 * pos_x \frac{h}{d} = 2 * r(\sin(angle) - \cos(angle)) + \frac{2h^2}{d^2} * angle$
Thus, the suggested approach done in this function is:
1. Given the cylinder which contains the helix, we calculate B,the closest point of the enclosing cylinder to current position.
2. Find C, the vertical projection point of B in the helix,
3. Find D, the horizontal projection point of B in the helix.
4. From C to D, cover the curve in N points looking for the closest one to current position. (N=10)

Parameters

pos	current cursor position
HSlices	helix tessellated slices
torusProjPerp	vector perpendicular to final projection, used later to calculate and apply dynamic forces.

CC vertical projection of point B
DD horizontal projection point of B

Returns
closest point from pos to current trial helix function

2.2.3.16 hduVector3Dd _SLE::pointProj2Line (GLdouble p[ ], GLdouble a[ ], GLdouble b[ ])

Projected point in line.
orthogonal projection of the a->p vector onto a->b vector. That is, the closest point of the line a-b to the point p. Useful to project the 3D device position in a plane and situate the cursor in a 2D experiment

Parameters
p point to be projected
a one point of the line
b other point of the line, different from b

Returns
orthogonal projection point

2.2.3.17 void _SLE::RotateMatrix (GLdouble alfa, GLdouble beta, GLdouble gamma, GLdouble M[16], GLdouble M2[16])

Matrix rotation.
alpha, beta&gamma angles given, and a matrix M, calculates the new matrix M2 applying the intrinsec rotations (Euler) to M. A=B*C*D, M2=A*M. B,C and D correspond to the rotation matrixes of alpha, beta and gamma. First, A is calculated by the BCD multiplication. Then, M2 is calculated.

Parameters
alfa alpha angle, in Y-axis
beta beta angle, in Z-axis
gamma gamma angle, in X-axis
M initial matrix 4x4
M2 rotated matrix. M2=A*M, where matrix A=B*C*D.

2.2.3.18 void _SLE::RotatePoint3D (GLdouble alfa, GLdouble beta, GLdouble gamma, GLdouble p[3], GLdouble p2[3])

Rotation of a point in 3D space.
alpha, beta&gamma angles given, and a point p, calculates the new point p2 applying the intrinsec rotations (Euler) to the point. A=B*C*D; p2=A*p.First, p is rotated in X-Z plane about Y-axis by alfa (Matrix D). Then, result is rotated in X-Y plane about Z-axis by beta (Matrix C).Finally, result is rotated Y-Z plane about X-axis by gamma (Matrix B)

Parameters

alfa	alpha angle, in Y-axis
beta	beta angle, in Z-axis
gamma	gamma angle, in X-axis
p	initial point
p2	rotated point. p2=A*p, where matrix A=B*C*D.

2.2.3.19 void _SLE::shuf eTrials ()

Shuf e experiment trials.
For every subject, trials are shuf ed

2.2.3.20 int _SLE::trialsNumber ()

Number of trials.

Returns
number of trials. If SLE.trialsN=0, calculates experiment factorial combinations and updates SLE.trialsN according to GUI settings.

2.2.3.21 hduVector3Dd _SLE::vectorProj (hduVector3Dd a, hduVector3Dd b)

Vector projection.
orthogonal projection of the B vector onto A vector

Parameters

a	vector where projection is calculated on

b vector to be projected

Returns
orthogonal projection vector

2.2.4 Member Data Documentation

2.2.4.1 int _SLE::currentTrial

current trial during the experiment, inside the interval [0, n-1]. Initial value is -1.

2.2.4.2 _outputParameters _SLE::output

output settings of the experiment instance

2.2.4.3 bool _SLE::paramEnableType[7][7]

Parameters enabled for each type of
experiment. Syntax is paramEnableType[t][p], where t correspond to an SLEtype and
p is one of the 7 parameters amplitude, width, height, alfa, beta, gamma or delta

2.2.4.4 SLESettings _SLE::settings

GUI panel linked settings

2.2.4.5 _trial* _SLE::trials

experiment array of trials. Memory is allocated dinamically in init_trials() function

2.2.4.6 SLEtype _SLE::type

enum type of the experiment. Value is updated everytime the listbox is changed.
The documentation for this class was generated from the following files:

- Visual Studio 2005/Projects/Steery/Steery/SLE.h
- Visual Studio 2005/Projects/Steery/Steery/SLE.cpp

2.3 _trial Struct Reference

Trial structure.
#include <SLE.h>

Public Attributes

- int parameter [7]
- oat id
- double sd
- DWORD MT
- oat OPM
- int subject

2.3.1 Detailed Description

Trial structure. Contains the input and output parameters of a experiment trial.

2.3.2 Member Data Documentation

2.3.2.1 oat _trial::id

Index of difficulty. Value derived from Amplitude/width.

2.3.2.2 DWORD _trial::MT

movement time. It last since the start position is crossed until the user reaches the end position

2.3.2.3 oat _trial::OPM

We use
the OPM to measure the out of path movement. OPM is the percentage of trajectory points outside the tunnel boundary. This metric was previously used by Kulikov and it was defined as "OPM (Out of Path Movement, percentage of sample points outside the Constraint lines). For example, if 100 points were sampled and 14 of those points were outside the Constraint lines, then OPM would be 14".

2.3.2.4 int _trial::parameter[7]

Measurements and angles of the trial: amplitude, width, height, alfa, beta, gamma and delta

2.3.2.5 double _trial::sd

standard deviation of trial user trajectory

2.3.2.6 int _trial::subject

Number of the current subject
The documentation for this struct was generated from the following file:

- Visual Studio 2005/Projects/Steery/Steery/SLE.h

2.4 HapticDisplayState Struct Reference

haptic variables,structs&functions

Public Attributes

- hduVector3Dd position
- HDdouble transform [16]
- hduVector3Dd anchor
- HDboolean isAnchorActive
- HDboolean recordUserTraj

2.4.1 Detailed Description

haptic variables,structs&functions

2.4.2 Member Data Documentation

2.4.2.1 hduVector3Dd HapticDisplayState::anchor

Cursor anchor

2.4.2.2 HDboolean HapticDisplayState::isAnchorActive

is anchor active boolean

2.4.2.3 hduVector3Dd HapticDisplayState::position

current position of the cursor

2.4.2.4 HDboolean HapticDisplayState::recordUserTraj

if active, records user trajectory

2.4.2.5 HDdouble HapticDisplayState::transform[16]

 transformation matrix
The documentation for this struct was generated from the following file:

- Visual Studio 2005/Projects/Steery/Steery/main.cpp

2.5 HookeForce Struct Reference

Dynamic Force structure.
#include <SLE.h>

Public Attributes

- int isON
- oat K
- oat x
- int equationGrade

2.5.1 Detailed Description

Dynamic Force structure. Represented according to Hooke's Law F=K*x. Depending on the equationGrade value, this force can be constant, linear, quadratic, logarithmic or inverse

2.5.2 Member Data Documentation

2.5.2.1 int HookeForce::equationGrade

Grade. See _SLE::GradeEquation for further detail.

2.5.2.2 int HookeForce::isON

Force activator. When is equal to zero, force is disabled.

2.5.2.3 oat HookeForce::K

 coefficient of the force.

2.5.2.4 oat HookeForce::x

 distance.
The documentation for this struct was generated from the following file:

- Visual Studio 2005/Projects/Steery/Steery/SLE.h

2.6 SLEParameter Struct Reference

Steering Law Experiment Parameter structure.
#include <SLE.h>

Public Attributes

- int min
- int max
- int iter

2.6.1 Detailed Description

Steering Law Experiment Parameter structure. One of the 7 parameters of a SLE: Amplitude, Width, Height, alfa, beta, gamma, delta. It is expressed in a min/max/iterator way, for factorial design

2.6.2 Member Data Documentation

2.6.2.1 int SLEParameter::iter

iterator value.

2.6.2.2 int SLEParameter::max

maximum value.

2.6.2.3 int SLEParameter::min

minimum value.
The documentation for this struct was generated from the following file:

- Visual Studio 2005/Projects/Steery/Steery/SLE.h

2.7 SLESettings Struct Reference

Experiment settings.
#include <SLE.h>

Public Attributes

- SLEParameter parameter [7]
- int type
- int solidwalls
- HookeForce repwalls
- HookeForce attrwalls
- HookeForce repgoal
- HookeForce attrgoal
- int shuf e
- int subjects
- int trialRepetitions
- int trialBreak
- int orientation
- int solidSurfaceOrientation
- int shadows3D
- int audio
- TextureParameters surface2DTexture
- HookeForce tunnelViscosity
- HookeForcetunnelFriction

2.7.1 Detailed Description

Experiment settings. Structure linked to graphical interface layer.

2.7.2 Member Data Documentation

2.7.2.1 HookeForce SLESettings::attrgoal

dynamic forces: repulsive goal. If enabled, the cursor if forced stay away from the trial start position

2.7.2.2 HookeForce SLESettings::attrwalls

dynamic forces: attractive walls. If enabled, the cursor is forced to stay away from the center of the path

2.7.2.3 int SLESettings::audio

If enabled, alarm sound loop is played when the user leaves the tunnel

2.7.2.4 SLESettings::HookeForcetunnelFriction

Tunnel friction. Texture parameters for 3D experiments

2.7.2.5 int SLESettings::orientation

only for 2D. Defines the orientation of the experiment. Default is horizontal=0 as a table, and vertical=1 as a board.

2.7.2.6 SLEParameter SLESettings::parameter[7]

Amplitude, Width, Height, alfa, beta, gamma, delta.

2.7.2.7 HookeForce SLESettings::repgoal

dynamic forces: repulsive goal. If enabled, the cursor if forced stay away from the trial goal position

2.7.2.8 HookeForce SLESettings::repwalls

dynamic forces: repulsive walls. If enabled, the cursor is forced to remain in the center of the path

2.7.2.9 int SLESettings::shadows3D

If enabled, tunnel and cursor shadows are generated (only 3D experiments)

2.7.2.10 int SLESettings::shuf e

if selected, trials will be shuf ed in the SLE initialization

2.7.2.11 int SLESettings::solidSurfaceOrientation

If enabled, creates haptic feedback, forcing the cursor to stay in an imaginary surface with Y=0. (or Z=0 in vertical orientation)

2.7.2.12 int SLESettings::solidwalls

If selected, haptic force is generated to simulate experiment’s walls

2.7.2.13 int SLESettings::subjects

number of participants of the experiment

2.7.2.14 TextureParameters SLESettings::surface2DTexture

Texture parameters for 2D experiments

2.7.2.15 int SLESettings::trialBreak

Tells the GUI when to pop-up the break window

2.7.2.16 int SLESettings::trialRepetitions

Trials will be multiplied per this number when generated

2.7.2.17 HookeForce SLESettings::tunnelViscosity

Viscosity of the tunnel. Texture parameters for 3D experiments

2.7.2.18 int SLESettings::type

integer casted from enum SLEtype. Associated to the experiment's type of tunnel
The documentation for this struct was generated from the following file:

- Visual Studio 2005/Projects/Steery/Steery/SLE.h

2.8 TextureParameters Struct Reference

Texture parameters of the experiment.
#include <SLE.h>

Public Attributes

- int ON
- oat hap_stiffness
- oat hap_damping
- oat hap_static_friction
- oat hap_dynamic_friction
- int height

2.8.1 Detailed Description

Texture parameters of the experiment. Its parameters are linked with the GUI texture panel.

2.8.2 Member Data Documentation

2.8.2.1 oat TextureParameters::hap_damping

Haptic damping value.

2.8.2.2 oat TextureParameters::hap_dynamic_friction

Haptic dynamic friction.

2.8.2.3 oat TextureParameters::hap_static_friction

Haptic static friction.

2.8.2.4 oat TextureParameters::hap_stiffness

Haptic stiffness value.

2.8.2.5 int TextureParameters::height

Height position of the simulated surface.

2.8.2.6 int TextureParameters::ON

Texture Panel activator. When is equal to zero, panel is disabled
The documentation for this struct was generated from the following file:

- Visual Studio 2005/Projects/Steery/Steery/SLE.h

Index

Appendix II. Steery User's guide

Steery User's Guide

Ren Laboratory

Kochi University of Technology, Japan

March 2010

User's Guide Table of Contents

Preface

1. About this guide

The Steery User’s Guide describes the features and usability of the Steery platform. You will find information on how to use and perform experiments with this software.

2. Steery User Manual Authors and Contributors

Content Writers
Alejandro Tatay Pascual

Proof Reading
Lawrie Hunter, Sun Minghui, Xiangshi Ren

Graphics, **Stylesheets**
Alejandro Tatay Pascual

Build System, Technical Contributors
Alejandro Tatay Pascual, Minghui Sun, Xiangshi Ren, Yoshinobu Onishi, Hironori Ishiyama

I. Getting started

Chapter 1 Introduction

1.1. Welcome to Steery

Steery is a Steering Law Experiment platform. Steery is suitable for research on trajectory-based interactions in a variety of environments.

Steery is designed to work with force-feedback haptic devices. It works with any haptic device of Sensable Technologies PHANTOM product line. Besides, Steery provides a different type of environment by using the haptic properties of these devices.

Steery is extensible. It is designed to be augmented by developing new types of experiments. For further information, check the Steery developer's manual.

Steery is an academic software developed for Ren Laboratory and its members. Any external use should be reported to Kochi University of Technology.

1.2. Authors

The first version of the Steery was written by Alejandro Tatay Pascual. Other labmates have contributed to its development: Sun Minghui, Yoshinobu Onishi, Hironori Ishiyama and the support of Prof. Xiangshi Ren.

1.3. Scope and Purpose

In the field of Human-Computer Interaction (HCI) Steering Law is a well-known predictive model concerning the user's performance in trajectory tasks. Nowadays, Steering Law experiments are used frecuently, but the experiment development process can be tedious and repetitive for the researcher.

Steery speeds up the workload of the researcher by substracting the development stage. Once the researcher runs Steery, he only has to set up the parameters and launch the experiment.

1.4. Features and Capabilities

The following list is a short overview of some of the features and capabilities which Steery offers:

- Variety of tunnels: 2-dimensional, 3-dimensional, straight, curved, ….
- Surface orientation in 2-dimensional tunnels
- Tunnel parameters: Iterator-based selection
- Texture material simulation
- Dynamic forces: haptic-force guidance
- Save data as txt, csv or even as a jpeg graph.

Chapter 2. Running Steery

2.1. System Requirements

The Steery platform requires certain hardware and software components to function properly.

- An Intel® processor based personal computer (minimum of Pentium® II class processor is recommended) or an equivalent AMD® processor personal computer.
- IEEE-1394a-2000 complaint FireWire® port.
- Windows 2000 or Windows XP.
- A Sensable Technologies PHANTOM haptic device
- There are no specific memory requirements to run the software; however, a minimum of 64 MB RAM is recommended for overall system performance.
- The PHANTOM Device driver (PDD), version 4.2.x. See the software documentation for any haptically enabled applications you will be running for specific PDD requirements .
- A SensAble supplied 6-6 pin FireWire cable or a 3rd party FireWire cable that exceeds IEEE 1394 implementation recommendations. **Laptop Users** please note that SensAble recommends using a 6-6 pin cable and add-in card instead of 6-4 pin. See www.sensable.com for the latest information.

Chapter 3. First Steps

3.1. Basic Concepts

3.1.1 SLE

Derived from Fitts' Law, Steering Law is a predictive model of human movement in trajectory-based tasks, concerning the speed and total time with which user may navigate through a 2-dimensional tunnel. This tunnel is defined by its amplitude *A* and width *W*, and the Steering Law can be expressed as

$$T = a + b\frac{A}{W}$$

where *T* is the average time to navigate through the path. In a Steering Law Experiment (SLE), a researcher can obtain *a,b* constants empirically, by linear regression.

3.1.2. Haptics

Haptics, or Tactile Feedback Technology, takes advantage of the user's sense of touch by applying forces and vibrations motions to the user. This simulation assists in the creation of virtual objects, as wall, tunnel texture, and dynamic forces directly applied to the device, among others.

3.1.3. Dynamic Forces

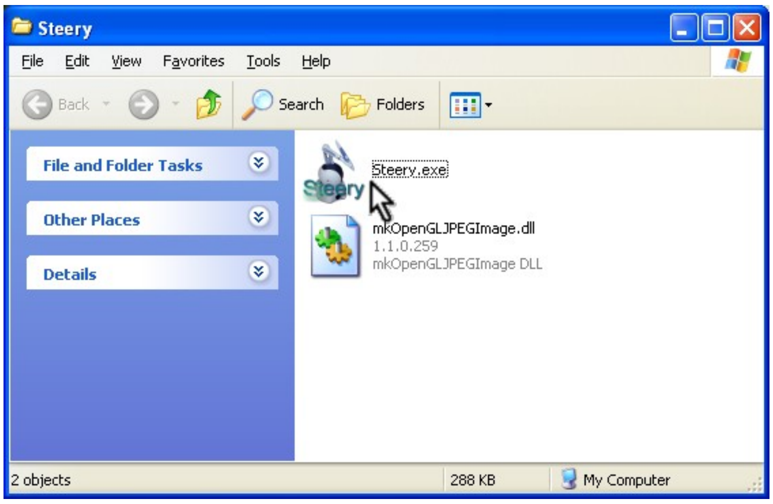
Haptic-guidance during steering task can improve the task performance. Steery allows the use of a variety of dynamic forces with several grades of force.

3.1.4. Trial

A SLE consists of a number of trials, each one with different parameters. Every trial data is recorded during the experiment, so the experiment designer can save it into a file.

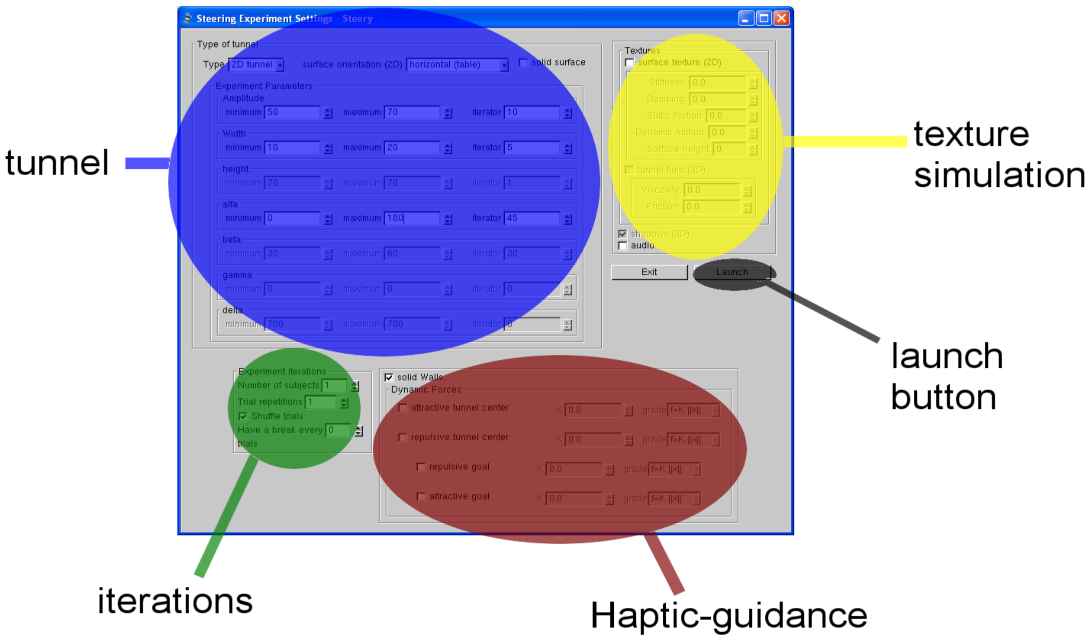
3.2. Starting the system

The Steery platform can be opened by double clicking on the Steery.exe icon, located in the Steery folder.



3.3. Main Window

Once the system has started, the main window of the application is opened. In this starting window, all the SLE settings can be selected. The main window is divided in four parts: *tunnel*, where the parameters and type of tunnel are selected; *iterations* selects the general settings of the SLE; *Haptic-guidance*, for dynamic forces; and *Texture simulation*.



Type of tunnel
Type 2D tunnel   surface orientation (2D) horizontal (table)   ☐ solid surface

Experiment Parameters

	minimum	maximum	iterator
Amplitude	50	70	10
Width	10	20	5
height	70	70	1
alfa	0	180	45
beta	30	60	30
gamma	0	0	0
delta	700	700	0

Experiment iterations
Number of subjects 1
Trial repetitions 1
☑ Shuffle trials
Have a break every 0 trials

☑ solid Walls
Dynamic Forces

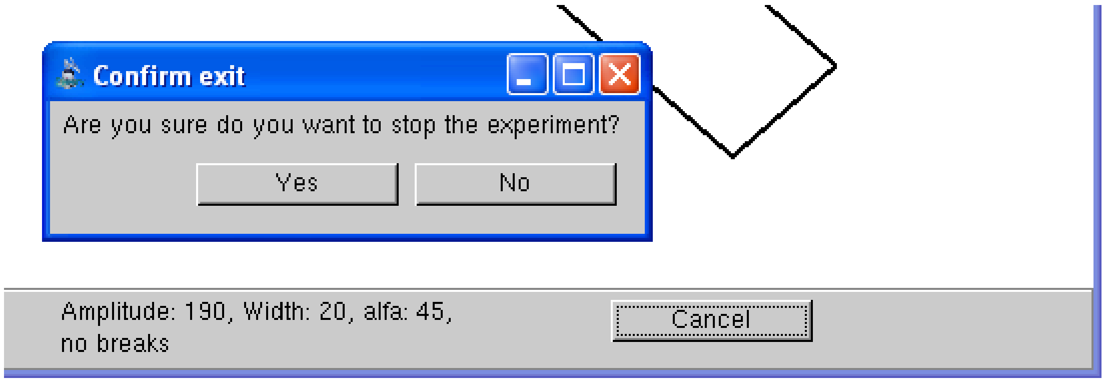
	K	grade
☐ attractive tunnel center	0.0	f=K ||x||
☐ repulsive tunnel center	0.0	f=K ||x||
☐ repulsive goal	0.0	f=K ||x||
☐ attractive goal	0.0	f=K ||x||

Textures
☐ surface texture (2D)
Stiffness: 0.0
Damping: 0.0
Static friction: 0.0
Dynamic friction: 0.0
Surface height: 0
☐ tunnel fluid (3D)
Viscosity: 0.0
Friction: 0.0
☑ shadows (3D)
☐ audio

Once the experiment is set up, it can already start by pressing the *launch* button.

3.4. Stopping and the system

To close the experiment before it starts, press Exit button in the main Window, close to the launch button. Otherwise, if the experiment has already started, press the Cancel button at the bottom bar (status bar). A dialog will pop up, asking for confirmation.



Are you sure do you want to stop the experiment?

Amplitude: 190, Width: 20, alfa: 45,
no breaks

3.5. Steery Quickies

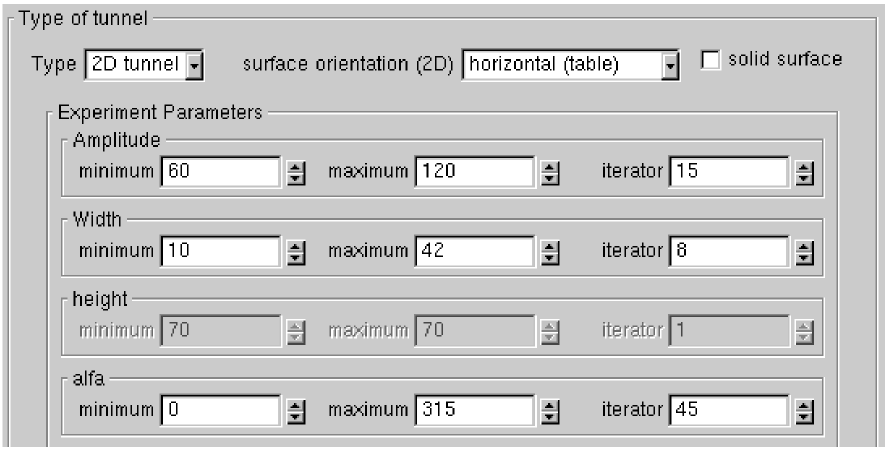
3.5.1. Perform a 2D experiment

Let’s suppose that you want to do a SLE to study the influence of the trajectory angle in the tunnel. The experiment parameters are:

• Amplitude: 60,75,90,105, 120.	x5
• Width: 10, 18, 26, 34, 42.	x5
• Trajectory angle: 0, 45, 90, 135, 180, 225, 270, 315.	x8
• Subjects: 5	x5
• Shuffle trials	

In total, this experiment has 5x5x8x5=1000 trials.

Set every iterator parameter (A, W and angle) in iterator-based notation, that is (min, max, iterator).



Type of tunnel

Type: 2D tunnel   surface orientation (2D): horizontal (table)   solid surface: ☐

Experiment Parameters

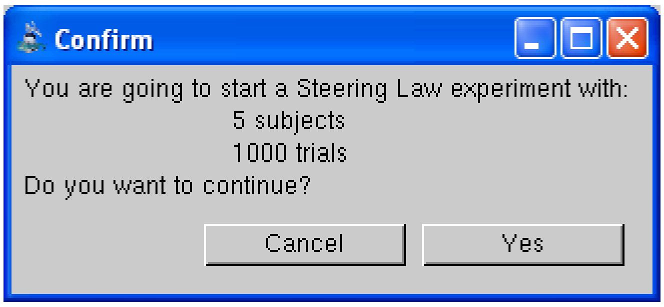
Amplitude: minimum 60, maximum 120, iterator 15

Width: minimum 10, maximum 42, iterator 8

height: minimum 70, maximum 70, iterator 1

alfa: minimum 0, maximum 315, iterator 45

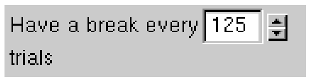
Start the experiment clicking on *launch* button. A confirmation dialog will pop up, so you can check if the number of trials and subjects are correct.

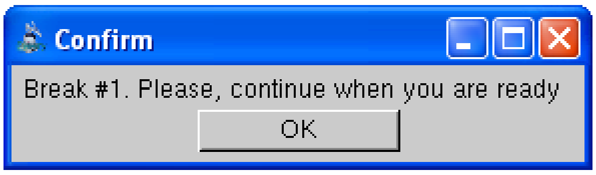


Confirm

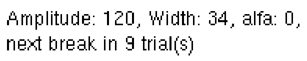
You are going to start a Steering Law experiment with:
5 subjects
1000 trials
Do you want to continue?

It is recommended to have a break between subjects, so they don’t perform other subject’s trials. For this reason, you may be interested in setting the “*have a break*” spinner, so a dialog will pop up when each subject finishes. During the experiment, in the status bar will appear information about the remaining trials until next break.

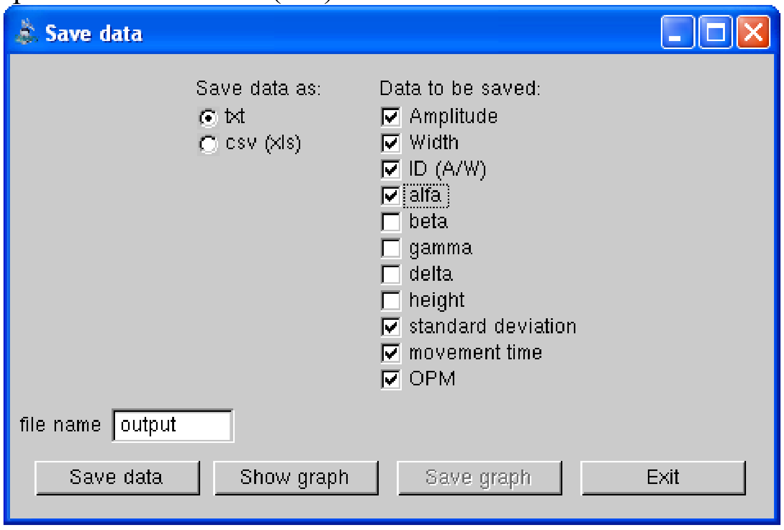

Have a break every 125 trials



Break #1. Please, continue when you are ready


Amplitude: 120, Width: 34, alfa: 0, next break in 9 trial(s)

After finishing all the trials, the *Save data* dialog pops up. You can select or deselect a variety of fields per trial that would appear (or not) in the output file. Also, you can choose between a Text file (txt) or a Comma-Separated Values files (csv).



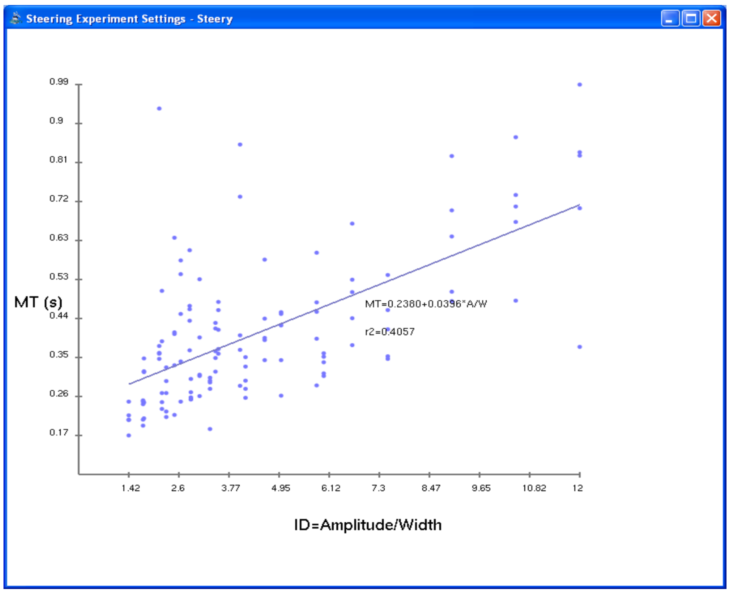
Save data as: [✓ txt] csv (xls)

Data to be saved:
- [✓] Amplitude
- [✓] Width
- [✓] ID (A/W)
- [✓] alfa
- [ ] beta
- [ ] gamma
- [ ] delta
- [ ] height
- [✓] standard deviation
- [✓] movement time
- [✓] OPM

file name output

3.5.2. Graph output results

Steery can represent the Index of Difficulty ($ID = \frac{A}{W}$) related to Movement Time in a graph, calculating it by linear regression of all the trials. In the *Save data* dialog, click on *Show graph* button, and the graph will be generated.



Next, save the graph as a jpeg file by clicking on the *Save graph* button. Notice that sometimes graph can not be represented if the basic conditions for regression calculus are not achieved.

II. Functionality

Chapter 4. GUI dialogs

4.1. Dialog Introduction

Dialogues are the most common means of setting options and controls in the Steery. The most important dialogues are explained in this section.

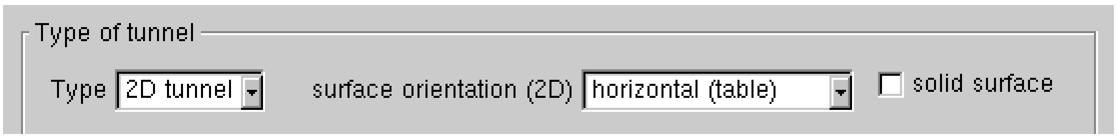
4.2. Tunnel selection

The tunnel selection interface covers most of the main window, and it's composed by two panels: type of tunnel and tunnel parameters:

4.2.1. Type

The first setting to select in a SLE is the type of tunnel, which are explained on Types of tunnel.

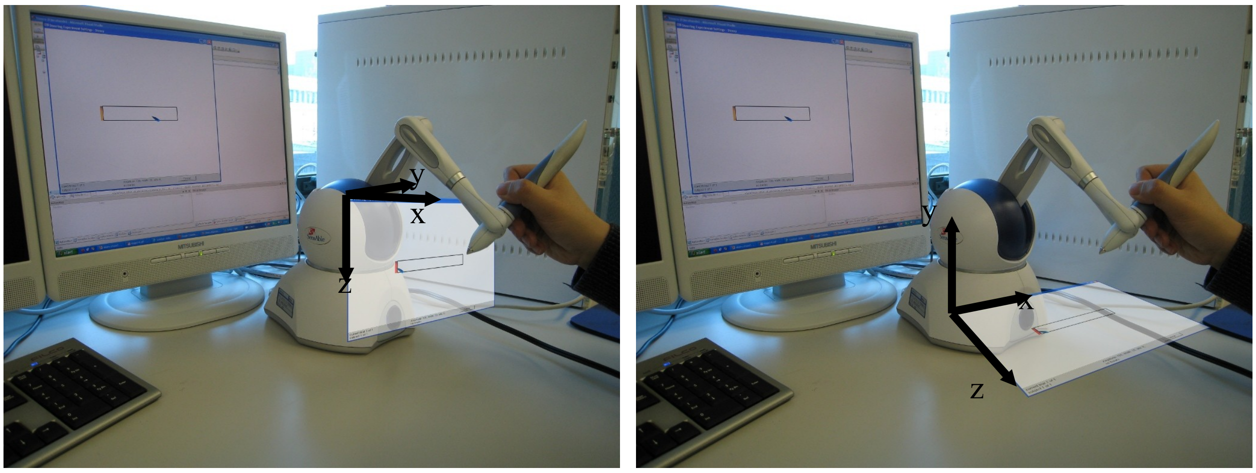
Any PHANTOM haptic device has a 3-dimensional workspace, but the designer may be only interested in 2-dimensional tunnels. In 2-dimensional tunnels, Steery allows the option to choose between horizontal or vertical orientation of the tunnel surface.


Type of tunnel

Type: 2D tunnel surface orientation (2D): horizontal (table) ☐ solid surface

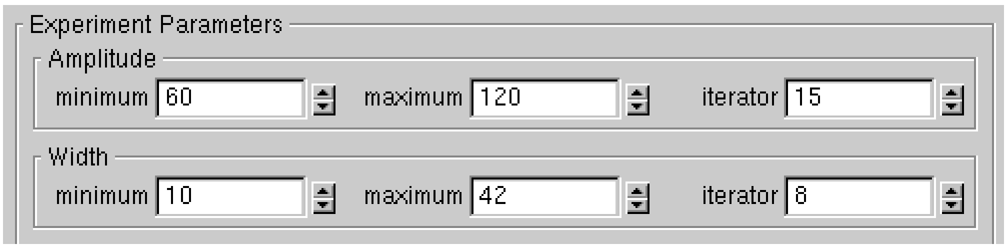
Thus, the trials will be performed in a horizontal orientation, simulating a steering task on table, or in vertical orientation, simulating it on a blackboard.

Next to the orientation listbox there is a *solid surface* checkbox. By clicking on it, it forces the PHANTOM stylus to be in y=0 during the trial. In other words, the stylus remains on the surface, either in horizontal or vertical.



4.2.2. Parameters

Below the type of tunnel panel there are the parameters settings. Seven parameters can be defined: amplitude (*A*), width (*W*), height (*H*), α (alpha), β (beta), γ (gamma) and δ (delta). *A*, *W* and *H* are lineal values, and α, β, γ and δ correspond to angles, represented by degrees, not radians.


Experiment Parameters

Amplitude

minimum 60 maximum 120 iterator 15

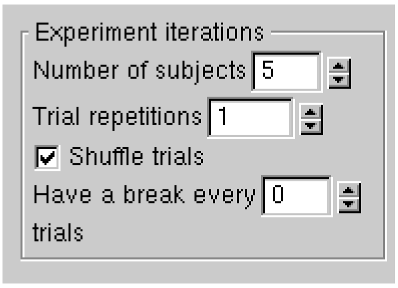
Width

minimum 10 maximum 42 iterator 8

Each parameter has three values to set up: minimum, maximum and iterator. By selecting this values, the trials will be generated combining all the posible values from minimum to maximum by iterator value increment.

4.3. Iterations

Below the tunnel panel there is the Experiment Iterations panel. Here it is possible to specify the number of subjects, trial repetitions, to shuffle trials and to select the frecuency of the break Dialog. This dialog will pop up during the experiment according to the value selected. If 0 is selected, no window will pop up.


Experiment iterations

Number of subjects 5

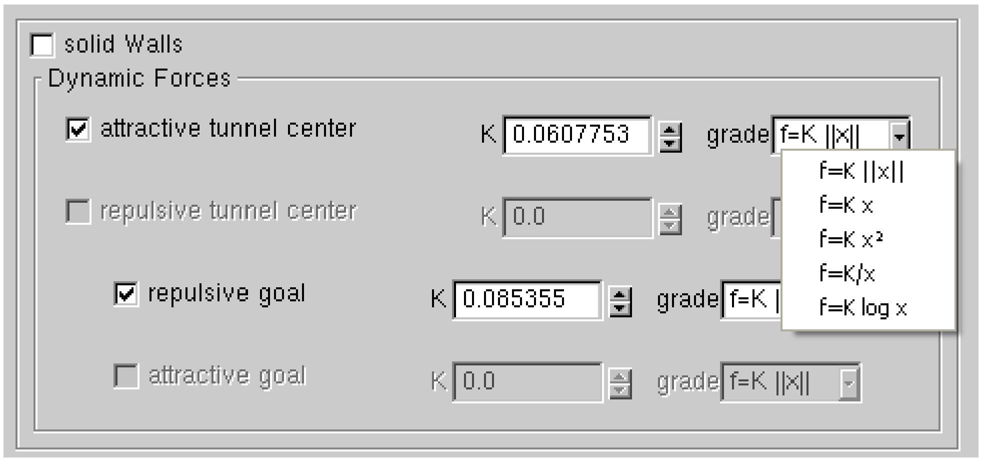
Trial repetitions 1

[✓] Shuffle trials

Have a break every 0 trials

4.4. Dynamic Forces

This panel is at the right side of iteration panel. Here you can define the Dynamic Forces or set the *solid Walls* on. By doing this, the tunnel walls become solid, preventing the stylus from getting outside the tunnel. There is an exception: in *boxwalls* tunnel, this checkbox is used to “solidify” the walls and floor of the box.


☐ solid Walls

Dynamic Forces

[✓] attractive tunnel center K 0.0607753 grade f=K ||x||

☐ repulsive tunnel center K 0.0 grade

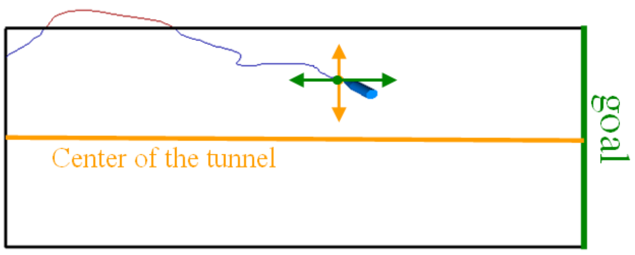
[✓] repulsive goal K 0.085355 grade f=K |

☐ attractive goal K 0.0 grade f=K ||x||

The final force behaves depending on the function grade selected. There are five mathematical grades available: constant, lineal, quadratic, decelerated (1/x) and logarithmic.

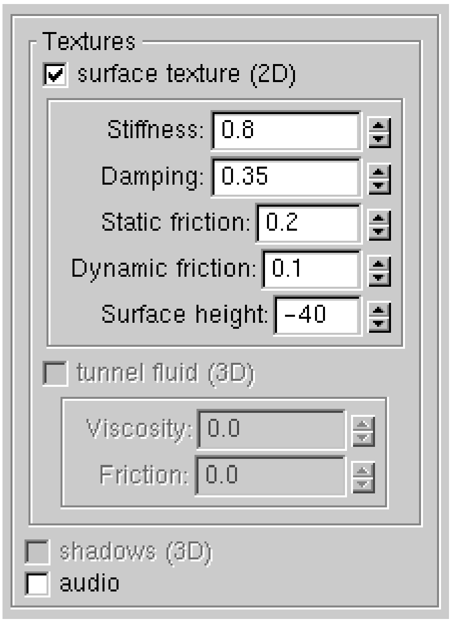
But what are these dynamic forces? These forces are applied to the stylus every moment during the whole trial to provide haptic-guidance to the subject. These forces vary according to a constant *K* and a distance *x*. There are 4 forces available:

- Attractive tunnel center: x is the distance from the cursor to the center of the tunnel
- Repulsive tunnel center: opposite to the previous one, they can't be used at the same time.
- Repulsive goal: x is the distance from the cursor to the goal line.
- Attractive goal: opposite to the previous one, they also are incompatible.



4.5. Texture Feedback

The Feedback panel is the interface to select the texture simulation variables. For 2-dimensional tunnels, these parameters are offered by PHANTOM API (Application Programming Interface) and are:



Textures
☑ surface texture (2D)
Stiffness: 0.8
Damping: 0.35
Static friction: 0.2
Dynamic friction: 0.1
Surface height: -40
☐ tunnel fluid (3D)
Viscosity: 0.0
Friction: 0.0
☐ shadows (3D)
☐ audio

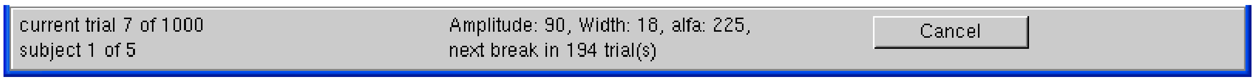
- *Stiffness*: hardness of the surface.
- *Damping*: springiness of the surface.
- *Static friction*: resistance of the surface to tangential motion when the cursor is not moving. In other words, how hard it is to slide along the surface starting from a complete stop.
- *Dynamic friction*: resistance of the surface to tangential motion when the cursor is moving. That is, how hard it is to slide along the surface once already moving.
- *Surface height*: position of the tunnel in the Y-axis.

In the other hand, for 3-dimensional tunnels the concept of texture is replaced for *fluid* inside the tunnel. In this context, two parameters can be set:
Viscosity: spring force opposed to the cursor movement.
Friction: resistance force opposed to the cursor movement.

At the bottom of this panel there are two checkbox: *shadows*, to draw the shadows in 3-dimensional tunnels (recommended), and *audio*. Audio feedback plays a loop error sound during the trial every time the cursor comes out of the tunnel boundaries, and stops it when it comes back.

4.6. Trial status bar

During the experiment, there is always a status bar at the bottom of the window which shows information of the current trial.



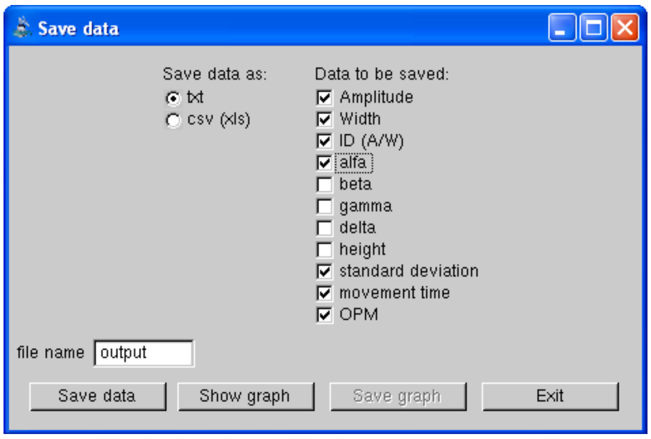
current trial 7 of 1000
subject 1 of 5
Amplitude: 90, Width: 18, alfa: 225,
next break in 194 trial(s)

It shows the number of the trial, the current subject, the trial parameters and trials left until next break. Besides, the *Cancel* button allows to interrupt the experiment before it finishes, as it was explained in Page 69.

4.7. Save data

Once the experiment is finished, the Save data dialog will pop up. At this point, all the trials information is in the Steery memory, and now it can be saved to a file. First, choose the output file type: plane text (txt) or Comma-Separated Values file (csv). The csv file type is useful to export the data to any Spreadsheet software.

At the right side, a variety of trial parameters can be selected to export it to the output file. A, W, Index of Difficulty, α, β, γ, δ, H, Standar Deviation to the center of the tunnel, Movement Time of the trial and OPM. Out-of-Path Movement or OPM is the percentage of sample positions of the cursor outside the tunnel in the trial.



Save data

Save data as:
[✓ txt]
csv (xls)

Data to be saved:
- [✓] Amplitude
- [✓] Width
- [✓] ID (A/W)
- [✓] alfa
- [ ] beta
- [ ] gamma
- [ ] delta
- [ ] height
- [✓] standard deviation
- [✓] movement time
- [✓] OPM

file name: output

Finally, type the name of the output file and press the *Save data* button. The file will be saved in: <Steery folder>/output/ folder.

Next to this button, there is the Show graph button. As explained in Page 71, if you click you will see in the main window a chart representing the results. Notice that sometimes graph can not be represented if the basic conditions for regression calculus are not achieved.

Next, save the graph as a jpeg file by clicking on the *Save graph* button. The picture will be saved in the same location as the data file.

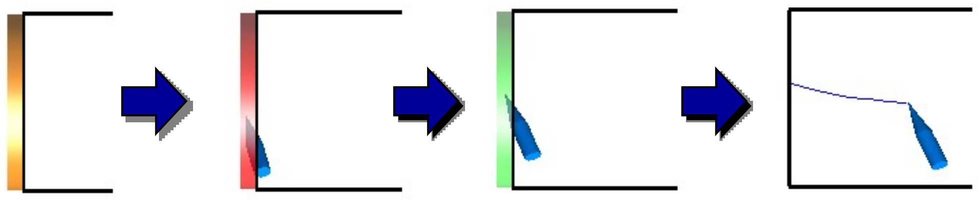
Chapter 5. Types of tunnel

There are different types of tunnel that can be used in a SLE. Every type of tunnel that Steery offers and its parameters are explained in the next section, in order of appearance

Once the tunnel type is chosen in the listbox, the parameters will be selectable to be set up.

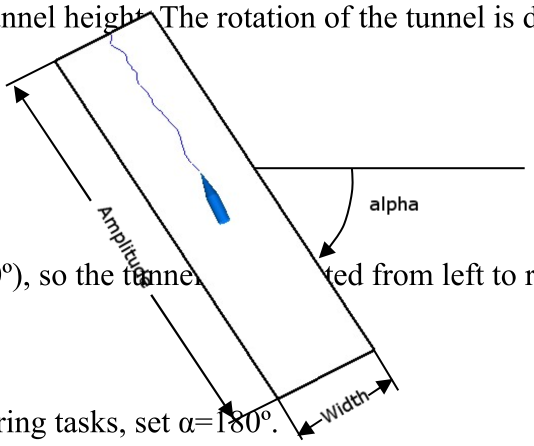
5.1. Trial execution

To carry out a SLE, the subject needs to know how to execute a trial. At the trial starting, a yellow starting region is drawn. Once the cursor moves over the region, it will change to red, to notify that the cursor is on the right place. Then, the subject press the device button, and the starting region color will change to green, indicating that the trial can start. At this point the cursor crosses the start position, the trial counter starts, and the subject steers the cursor to the end of the tunnel.



5.2. 2D tunnel

The basic SLE tunnel, is defined by three parameters: Amplitude, Width and angle (α). Amplitude represents the tunnel length, and Width the tunnel heigh The rotation of the tunnel is defined by alpha, measured in degrees.

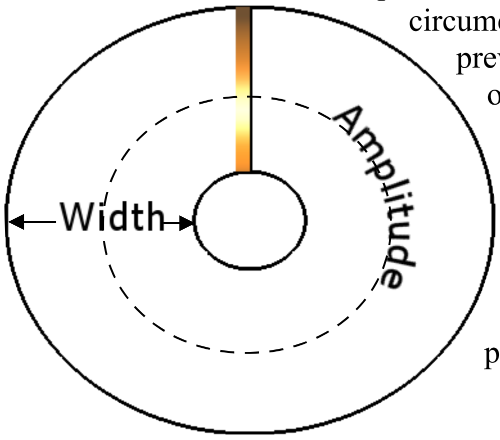


The starting position by default is at left (α=0º), so the tunne ted from left to right.

In case you want to perform right-to-left steering tasks, set α=180º.

5.3. 2D torus

This is a circular tunnel, to perform curved steering tasks. This tunnel is formed by two circumeferences. The parameters to be set are the same ones as the previous tunnel: A, W, α. In this case, Amplitude is the perimeter of the curved tunnel, so its radius r= 2π/A. Width is the thickness between both circumferences. Thus, inner circumference has a radius r1=r-W/2, and the outer one has radius r2=r+W/2.



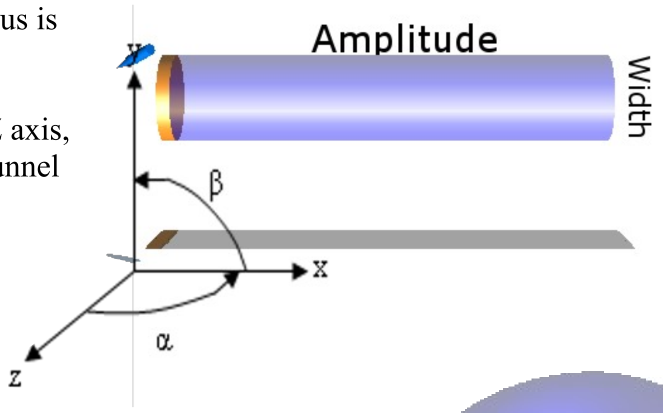
Starting position is by default at the top, and the task is always performed clockwise. Alpha rotates the starting position clockwise.

5.4. 2D curve

This experimental option is unavailable by default, subject to further study and development. The purpose of this option is to shape the tunnel in a non-regular, free curve.

5.5. 3D tunnel

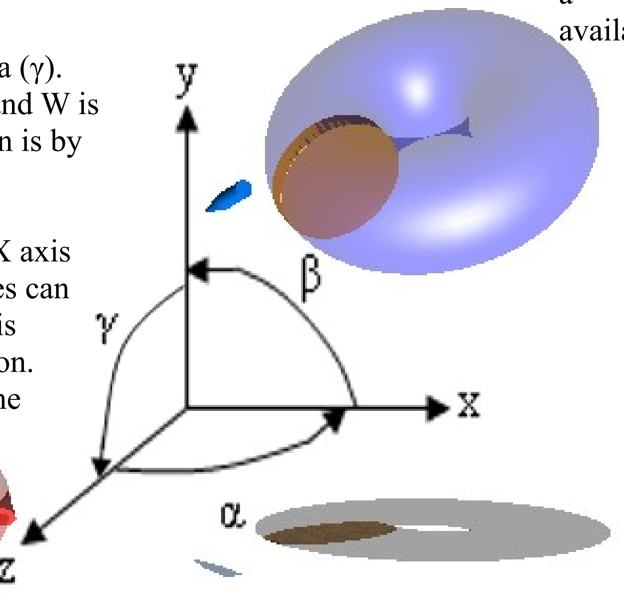
3D tunnel is shaped as a cylinder in a three dimensional space. The available parameters are: A, W, alpha (α) and beta (β). Cylinder's base radius is r=W/2, and Cylinder's height equal to A.



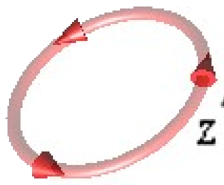
Alpha and beta rotate the tunnel in Y and Z axis, respectively. Combining both angles, the tunnel can be orientated in any position.

5.6. 3D torus

This tunnel is shaped as a torus or toroid, a doughnut-shaped object. The parameters available in this tunnel are: A, W, alpha (α), beta (β) and gamma (γ). Similar to 2D torus, A is the perimeter of the tunnel, and W is the thickness of the torus (diameter). The start position is by default situated at the left side.



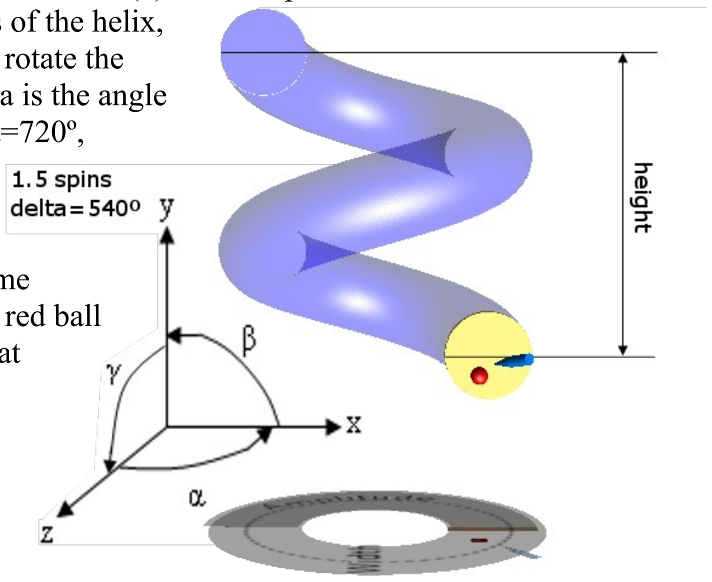
Alpha, beta and gamma rotate the tunnel in Y, Z and X axis respectively. Since the combination of this three angles can produce a confusing orientation, some user feedback is given to help the subject understand the trial orientation. This feedback is located at the bottom left corner of the screen. It consist on a thiny red torus orientated in the same way as the tunnel, with 3 arrows representing the task direction.



5.7. 3D helix

This tunnel is the most complex of Steery. It is shaped as a helix, a regular curve in three-dimensional space, comonly called coil spring or spiral. All the seven parameters are used to draw it: A, W, H, alpha (α), beta (β), gamma (γ) and delta (δ). A is the perimeter of the helix floor projection (a 2D-torus); W is the thickness of the helix, same as 3D torus; Alpha, beta and gamma rotate the tunnel in Y, Z and X axis, repectively; delta is the angle spinning of the helix. For instance, if delta=720º, the helix makes 2 spins; and H is the height of the helix.

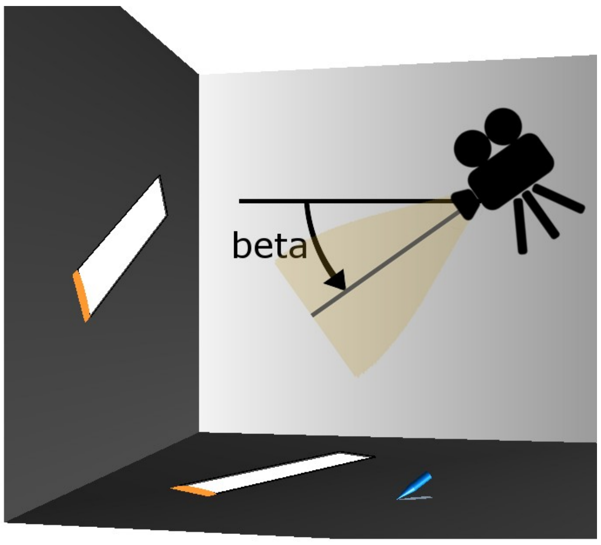
Because of the complexity of this task, some feedback is provided to help the subject: a red ball situated in the center of the helix tunnel that represents the closest point to the cursor.

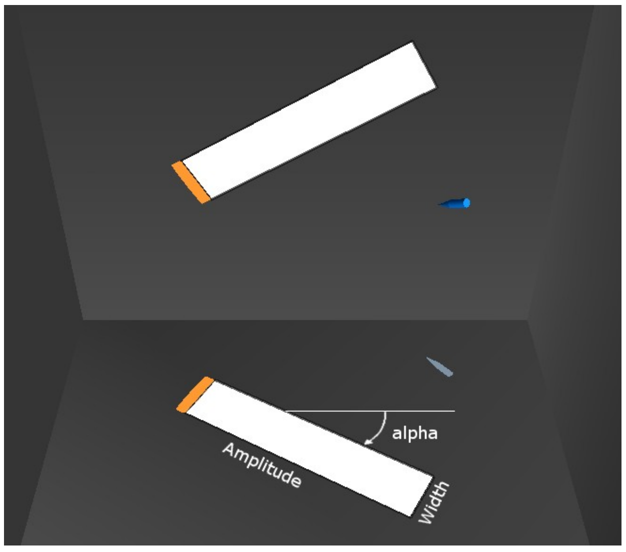


5.8. boxwalls

Boxwalls is a special SLE designed specifically for our research in March 2010. This experiment simulates a 3-dimensional space inside a room, with floor and walls. It is recommended to set the "*solid walls*" checkbox on to make them solid.

The experiment creates a 2D-tunnel in both floor and front wall with the purpose of comparing both steering tasks and camera angles.The parameters available are: A, W, alpha (α) and beta (β). As other experiments, A is amplitude, W is width and alpha is tunnel rotation. Beta, however, is the camera pitch, or camera angle inclination. Beta covers β=[0,90], where β=0º is equivalent to stare at the front wall and β=90º to stare at the floor.





Chapter 6. Troubleshooting

Steery is an academic software subject to further development, and you can experiment some issues. This section tries to answer some of the most common Steery issues.

I'm moving the Phantom stylus but it doesn't appear on the screen.

1-Move the stylus to the centre of the workspace. Sometimes cursor is not drawn at the boundaries of the workspace.
2-Check if the device is already plugged in.
3-Check that PHANTOM Device driver (PDD) is already installed. Launch Phantom Test software to confirm that the device is detected.

Cursor is over the start region, but its colour doesn't change. Why can't I reach it?

Because the cursor is not inside the start region. In 3-dimensional tunnels, it's difficult for the subject to perceive the depth. In those cases it's recommended to activate shadows checkbox, so the cursor's shadow helps to perceive its depth.

Tunnel size overflows the screen, so I can't reach the start position.

The parameters selected are too big for the device workspace, so please select a smaller parameters and start again.

For any technical troubleshooting, please check the Developer's Manual.

Appendix A. Glossary of Terms

Dynamic force
Simulated force that can be induced by the haptic device during the trial.

Haptic-guidance
Simulated force by the device which improves steering task skills.

Haptics
Technology that interfaces with the user through the sense of touch.

HCI
Human-Computer interaction, the study of how people interacts with computers.

Helix
Curve in 3-dimensional space mathematically defined by $\{\cos(t), \sin(t), t\}$.

Iterator
Object that allows to traverse through the elements of a collection.

MT
Movement time, time that takes a trial to be crossed.

OPM
Out-of-the-Path Movement, percentage of the crossing task outside the boundaries.

SD
Standard Deviation. Mathematically, dispersion variability of a collection.

SLE
Steering Law Experiment.

Steering Law
Predictive model which studies the user's performance in trajectory tasks.

Torus
Surface of revolution generated by a revolving circle about a coplanar axis.

Trial
Unit test of a single trajectory-based task.

Appendix B. Bibliography

1. SensAble Technologies, Inc. 15 Constitution Way, Woburn, MA 01801 www.sensable.com

2. Steery Developer’s Documentation. Alejandro Tatay Pascual. Ren Lab. 2010

3. Ren Laboratory. Kochi University of Technology. Kochi, Japan.

Appendix III. Boxwalls experiment report

Boxwalls experiment report: Limitations of 2D Steering tasks in 3D environments

Alejandro Tatay Pascual

Ren Laboratory

Kochi University of Technology, Japan

March 2010

Abstract

Interaction techniques in 3D scenarios can be performed with indirect devices like an arm attached stylus. Usually these techniques have a series of drawbacks while displayed in common screens, such as arm fatigue or user depth perception. In this paper we study and model trajectory based tasks in a box-like 3D environment with haptic-guided device Phantom Omni. Through this experiment we analyze the validation, applicability and limitation of Steering Law in the third dimension.

1 Introduction

In the field of Human-Computer Interaction (HCI) Steering Law is a well-known predictive model which studies the user's performance in navigating through a tunnel. On the other hand, Phantom Omni is a 3D haptic pointing device constituted by a stylus attached to a robotic arm. The manufacturer[8] provides to the developers OpenHaptics library to develop C++ applications.

Steery project[7] is a platform to design and launch Steering Law experiment with the Haptic Omni device, taking advantage of the OpenHaptics API. Thus, the designer can study the performance of the device in different environments: 2D, 3D, straight, curved, with haptic guidance, on textured surface, ...

We discuss the prior work related to existing models for pointing and steering. Fitt's Law(1) is a formal relationship that models speed/accuracy tradeoffs in aimed movements. It predicts the time T needed to point to a target of width W and at distance A is logarithmically related to the inverse of the spatial error $\frac{A}{W}$, that is:

$$T = a + b\log_2\left(\frac{A}{W} + c\right) \quad (1)$$

where a and b are empirically determined constants, and c is set in the interval [0,1]. The factor $\log_2\left(\frac{A}{W} + c\right)$, called the index of difficulty (ID), describes the difficulty to achieve the task: the greater ID, the more difficult task.

In the last years, researches have introduced new models derived from this formula. Accot and Zhai *et al.*[1] refine the model for 2-dimensional tunnels, where the difficulty of the task is not related to the logarithm of $\frac{A}{W}$, but to $\frac{A}{W}$. That leads to equation 2:

$$T = a + b\frac{A}{W} \quad (2)$$

Similarly, several studies such as [2], [3], [4] and [5] have shown that the performance of steering tasks can be improved by providing users with tactile feedback during movement along a path. Xing-Dong et al. [6] propose a model for force-enhanced goal crossing task (3),

$$T = a + b\left(\frac{A}{W + \eta * S}\right) \quad (3)$$

where the difficulty of the task is inversely proportional to the spring stiffness S that represents the intensity of the guiding force. η is an empirically constant determinined by the contributions of W and T.

In conclusion, Steering Law has been commonly applied to study 2-dimensional crossing tasks. However, a three dimensional devices with 3 or more Degrees Of Freedom (DOF) can be exploited to study crossing tasks in 3D scenarios. Can the Steering Law model predict 3D crossing tasks with the same stability and precision as it does in 2D tasks? Obviously, this kind of tasks have several drawbacks, such as arm fatigue and lack of depth perception in a common 2D screen. But how far can the Steering Law be applied?

Actually, this experiment doesn't study 3D crossing tasks, but 2D crossing tasks in a box-like 3D scenario. The box abstraction is a natural and very common paradigm for any user, where the camera is placed inside a room (box) with walls and floor. Real 3D box-like interfaces can be tested and used currently as a desktop applications, such as Bumptop [9] or Shock Desktop 3D[10]. In these scenarios, the limitation of the mouse prevents the use of a third DOF, and the user can perform crossing tasks, like dragging objects in floor and walls, while the camera changes between different angles. Besides, this tasks represent natural movements for the user, such as sweeping the floor of the room or painting the walls. Nevertheless, swapping time between floor and walls depends mainly in the initial camera angle relative to floor and wall, so it is crucial to choose a value comfortable enough to navigate through the task as well as suitable enough for a short swapping time.

2 Experiment

This study analyzes several parameters of the Steering Law in a 3D environment. The software displayed a scenario similar to a room, with floor and walls, where the cursor can move with 6 degrees of freedom. Haptic feedback was provided by the Phantom device to simulate the floor and walls strength, hence the user was able to feel these surfaces. Because of the nature of this experiment, it has been named as *boxwalls*.

The participants were asked to perform crossing tasks in the front wall and floor, both horizontal and vertical tasks. Also the camera angle varies between middle angles close to 45 degrees to study the performance in each angle.

2.1 Apparatus

The experiment ran on an Intel Pentium(R) 4 CPU 3.01 GHz PC with 1 GB RAM. The software used was Steery platform[7] running on MS Windows XP Professional SP2. A 17" TFT LCD screen with a 1280x1024 pixel resolution displayed the image. A Phantom OMNI device[8] was used for input, controlled by the Steery program. The device was positioned at 43 cm from the edge of the table, and elevated 4 cm from the table.

2.2 Participants

Twelve volunteers participated in this study, three females and nine males between ages of 19 to 30. One of them was left-handed, and three of them had previous experience with the Phantom OMNI device.

2.3 Procedure

Participants where asked to hold the stylus of the phantom device. According to each participant's preferred hand, the device was placed at right or left side of the screen. All participants were seated comfortably and controlled the Phantom OMNI stylus with their dominant hand. Besides, they were allowed to lean their arm on the table while completing the task, much like a natural arm rest with a common pen. Before starting the experiment, participants practiced crossing tasks in the scenario until they feel ready to start.

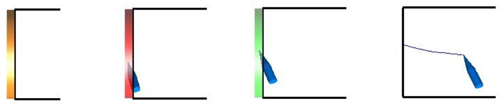


Figure 1: Trial starting procedure

In every trial, a straight tunnel is drawn in one of the walls (floor or front wall). To start the trial, participants were required to place the cursor inside a yellow start region attached to one side of the tunnel. Once the cursor was placed over the start region, it changed its color to red. Then participants clicked the stylus button to indicate the are ready to proceed, so the start region turned into green. For a successful trial, the cursor had to cross the tunnel from that side to the other. Once the trial was completed, the next shuffled trial takes place.

A trial started as soon as the cursor enters to the tunnel, and finished as soon as the end goal was crossed. Trials had different heights, distances, angles (horizontal or vertical) and camera angles.

2.4 Design

Owing to the fact that the experiment studies several values, each participant had to perform a wide quantity of trials. Thus, study employed a 4x4x2x4 within-subject factorial design. The independent variables are the width of the goals W(5, 10, 15 and 20 pixels), the amplitude of the tunnel A(10, 30, 50, 70), the tunnel orientation α(0 horizontal or 90 vertical), the camera angle β (30, 40, 50 and 60 degrees) and the wall selection γ(front wall or floor). The combination of this independent variables resulted in 256 trials. See Fig. 2.

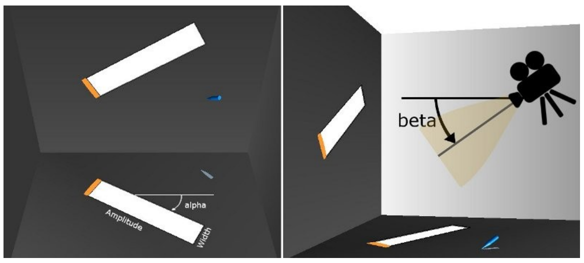


Figure 2: Experiment parameters design

The experiment was organized into 3 blocks. Each of these blocks contained 256 trials, and each of this trial represented an $A * W * \alpha * \beta * \gamma$ combination. These 3 blocks were performed by everyone of the 12 participants. This resulted in a total of 9216 trials.

It is important to explain the $\alpha * \gamma$ combination. Alpha values were 0 and 90 degrees, and gamma values were floor and wall. The combination of both generated four types of crossing tasks: Floor left-to-right, floor back-to-front, wall left-to-right and wall down-to-top crossing tasks, as shown in Fig. 3.

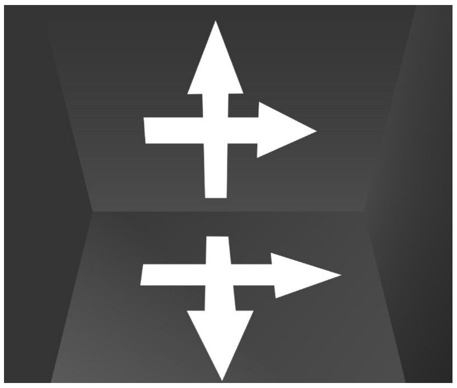


Figure 3: Crossing task types

2.5 Results

Movement Time (MT), OPM (Out of the Path Movement, or time percentage that the cursor is out of the tunnel) and standard deviation were recorded in every trial. To discard wrong trials, we removed outliners with a standard deviation above 3 from the mean. Also, trials with $MT > 4$ seconds and $OPM > 0.1$ were rejected. These parameters generated 7283 correct trials, which correspond to the 79.02% of the trials, so the error rate resulted in 20.98%, higher than expected.

Data was analyzed according to the type of move, camera angle and learning block.

Type of move

As commented above, the $\alpha * \gamma$ combination generates four types of crossing tasks: Floor left-to-right, floor back-to-front, wall left-to-right and wall down-to-top. Data was divided in this four groups looking for the steering law relation between Movement Time and Index of Difficulty. Using a linear regression method we estimated the value of the empirically determined constants a and b in ecuation (2) and the correlation coefficient R^2 for every group. The results are shown at *Table 1*. For instance, *Fig. 4* shows the obtained results in "wall L2R" tasks.

Table 1: Crossing task type

	Floor L2R	Floor B2F	Wall L2R	Wall D2T
a	0.104	0.079	0.077	0.067
R	0.594	0.436	0.547	396

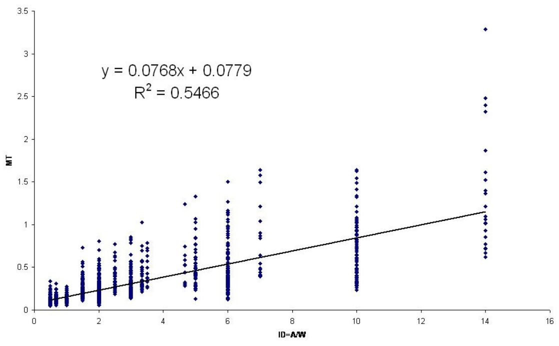
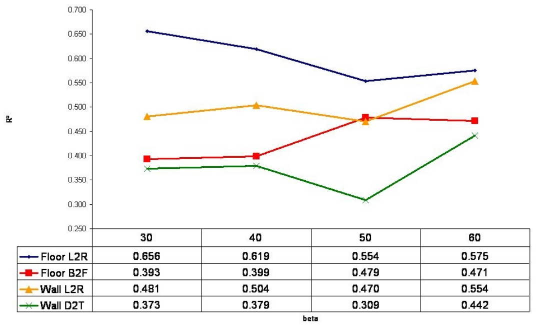


Figure 4: Front wall left-to-right

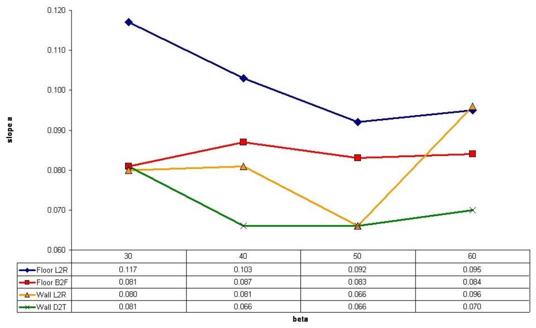
Camera angle

The camera angle relative to the front wall was studied to look for a optimal balance between both walls. Trials were executed with angles of 30, 40, 50 and 60 degrees. In each of this groups, each one of the crossing task types was separated in order to study the correlation coefficient and the slope a. Results are shown in *Fig. 5* an *Fig. 6*. Both charts reveal a low correlation between the camera angle and the difficulty of the task. Correlation coefficient fluctuates between $0.3<R^2<0.66$, and ID slope oscillates around the interval [0.06,0.12]. Nevertheless, higher R^2 correlation an more linear slope can be observed for the *Floor left-to-right* task.



	30	40	50	60
Floor L2R	0.656	0.619	0.554	0.575
Floor B2F	0.393	0.399	0.479	0.471
Wall L2R	0.481	0.504	0.470	0.554
Wall D2T	0.373	0.379	0.309	0.442

Figure 5: Camera angle R^2

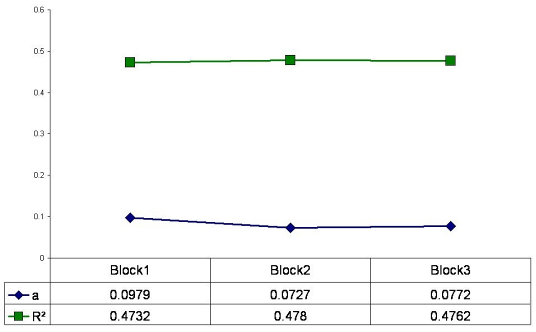


	30	40	50	60
Floor L2R	0.117	0.103	0.092	0.095
Floor B2F	0.081	0.087	0.083	0.084
Wall L2R	0.080	0.081	0.066	0.096
Wall D2T	0.081	0.066	0.066	0.070

Figure 6: Camera angle slope a

Learning block

For every participant the experiment was segmented into 3 blocks of 256 trials each. We analyzed the learning curve of this blocks looking for a decrement in ID, hence the subjects will improve their ability to navigate in this scenario. However, the results reveals that after the first block, the task difficulty remained in the same values. Results can be observed in *Fig. 7*, which shows the correlation coefficient and the Steering Law formula slope a.



	Block1	Block2	Block3
a	0.0979	0.0727	0.0772
R²	0.4732	0.478	0.4762

Figure 7: Learning curve and R^2

Subjective evaluation

After the experiment, participants were asked to answer a question paper to rate the difficulty of every type of task from 1 (very difficult) to 5 (very easy). Subjectively, the easiest average difficulty resulted Floor L2R Diff=3.4. As mentioned before, this task revealed more correlation to the steering law formula than the other task, whose subjective difficulty average was: floor B2F=2.9, wall L2R=2.8, wall D2T=2.8.

3 Discussion and future work

The results of our study showed that the rejection of wrong trials produced high error rates. Aditionally, the steering law correlation was lower than expected in this type of experiment, which commonly tends to be $R^2>0.8$. This can be attributed to the complexity of the task and participant's lack of skill. Subjects had experience in performing 2D crossing tasks with common devices (mouse, pencil, brush, ...) in the real world, which has 3 dimensions. Conversely, extrapolating these tasks into a 2D screen removes the third dimension from the scenario, hence the user faces the lack of depth perception. This problem is poorly balanced with cursor shadow and cursor perspective feedback. Therefore, here lies the main disadvantage of this experiment: depth perception depends on each participant's visual-spatial intelligence, which differs widely from one participant to another. As a consequence, Steering Law can not be generally applied in this scenario.

That being said, we analyze differences among the four types of crossing tasks. Floor L2R task revealed a higher correlation with the Steering Law formula than the rest. Furthermore, participants coincided to elect this task as the easier one. This may be due to the fact that participants have experience in crossing tasks over horizontal surfaces, and left-to-right movements are more suitable to human body.

Concerning camera angle, we demonstrated that an appropiate angle reduces the difficulty of the task. However, this improvement is not related proportionally with the angle. On the other hand, camera perspective distorts the tunnel shape, hence its dimensions A,W and the distorted Index of Difficulty derived from them makes the Steering Law harder to apply.

In summary, we presented a experiment to have a closer look to the limitations of Steering Law in a 3D scenario. We found that applying the general method for 2D steering law experiment is not enough to analyze 3D scenarios. This suggests that future work can focus on the search of dependent and influencing parameters which would define crossing tasks' behaviour in 3D scenarios, as well as 3D crossing tasks. We expect that this report will be useful for future students to have a better understanding of 3D Steering Law applications.

References

[1] Accot, J., Zhai, S.: Beyond Fitts Law: Models for trajectory-based HCI tasks. In: ACM CHI, pp. 295-302 (1997)

[2] Ahlström, D.: Modeling and improving selection in cascading pull-down menus using Fitts' law, the steering law and force fields. In: ACM CHI, pp. 61- 70 (2005)

[3] Campbell, C., Zhai, S., May, K., Maglio, P.: What You Feel Must Be What You See: Adding Tactile Feedback to the Trackpoint. In: INTERACT, pp. 383-390 (1999)

[4] Dennerlein, J., Martin, D., Hasser, C.: Force-feedback improves performance for steering and combined steering-targeting tasks. In: ACM CHI, pp. 423-429 (2000)

[5] Forsyth, B., MacLean, K.: Predictive Haptic Guidance: Intelligent User Assistance for the Control of Dynamic Tasks. IEEE Transactions on Visualization and Computer Graphics 12(1), 102-113 (2006)

[6] XingDong Y., Pourang I., Boulanger P., Bischof W.: A Model for Steering with Haptic-Force Guidance

[7] Tatay Pascual, A. *Steery: a Steering Law Experiment Platform with Phantom Omni*, Ren Lab Symposium, Kochi University of Technology (2010)

[8] SensAble Technologies, Inc. 15 Constitution Way, Woburn, MA 01801 www.sensable.com

[9] BumpTop. Bump Technologies Inc. http://bumptop.com/

[10] Shock Desktop 3D. http://www.docs.kr/