



ugr | Universidad
de **Granada**

TRABAJO FIN DE GRADO
DOBLE GRADO INGENIERÍA INFORMÁTICA Y MATEMÁTICAS

Sistema de adaptación motora con entorno de realidad virtual

Autor
Celia Arias Martínez (alumno)

Directores
Eduardo Ros (tutor1)
Jesús (tutor2)



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
Granada, mes de 201

Sistema de adaptación motora con entorno de realidad virtual

Celia Arias Martínez (alumno)

Palabras clave: palabra_clave1, palabra_clave2, palabra_clave3,

Resumen

Poner aquí el resumen.

Project Title: Project Subtitle

First name, Family name (student)

Keywords: Keyword1, Keyword2, Keyword3,

Abstract

Write here the abstract in English.

Yo, **Nombre Apellido1 Apellido2**, alumno de la titulación **TITULACIÓN de la Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI XXXXXXXXXX, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Nombre Apellido1 Apellido2

Granada a X de mes de 201 .

D. **Nombre Apellido1 Apellido2 (tutor1)**, Profesor del Área de XXXX del Departamento YYYY de la Universidad de Granada.

D. **Nombre Apellido1 Apellido2 (tutor2)**, Profesor del Área de XXXX del Departamento YYYY de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Título del proyecto, Subtítulo del proyecto***, ha sido realizado bajo su supervisión por **Nombre Apellido1 Apellido2 (alumno)**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a X de mes de 201 .

Los directores:

Nombre Apellido1 Apellido2 (tutor1)
Apellido2 (tutor2)

Nombre Apellido1 Apellido2 (tutor1)

Agradecimientos

Poner aquí agradecimientos...

Índice general

1. Introducción	9
1.1. Contexto del proyecto	9
1.2. Objetivos	10
1.3. Fases del proyecto	11
1.4. Planificación del proyecto	13
1.5. Material	15
1.5.1. Touch	15
1.5.2. OpenHaptics	17
2. Fundamentos matemáticos	18
2.1. Álgebra lineal, Geometría afín y Geometría Proyectiva	18
2.1.1. Álgebra lineal	19
2.1.2. Geometría afín	22
2.1.3. Espacio Proyectivo	25
2.2. Aplicaciones en informática gráfica	26
2.2.1. Paso de coordenadas locales a coordenadas de vista . .	29
2.2.2. Paso de coordenadas de vista a coordenadas de recorte	30
2.2.3. Paso de coordenadas de recorte a coordenadas de pantalla	33
2.2.4. Paso de coordenadas de pantalla a coordenadas finales	34
3. Desarrollo del proyecto	36
3.1. Planificación del experimento	36
3.1.1. Diseño de nuestro experimento	36
3.2. Desarrollo de la aplicación	40
3.2.1. Instalación del software de OpenHaptics	41
3.2.2. Implementación del código	42
3.2.3. Script de python	48
3.3. Obtención de resultados	49
3.4. Análisis de resultados	51
3.4.1. Análisis de la misma fase entre individuos	52
3.4.2. Análisis de diferentes fases entre el mismo individuo .	90

ÍNDICE GENERAL

8

4. Discusion y conclusiones

98

Capítulo 1

Introducción

1.1. Contexto del proyecto

El cerebro ha sido ampliamente estudiado en relación con el aprendizaje y el control de movimientos. Experimentos científicos han revelado su papel fundamental en la adquisición de habilidades motoras y en la coordinación precisa de los movimientos, mostrando que los individuos con daños en el cerebro presentan dificultades para adaptarse a nuevos patrones de movimiento o para realizar ajustes precisos en la fuerza y la dirección de sus movimientos.

La realidad virtual ha sido utilizada como una herramienta para investigar la función del cerebro en la percepción espacial, la coordinación motora y el aprendizaje de movimientos. En los experimentos los participantes son expuestos a entornos virtuales que simulan situaciones específicas -manipulación de objetos virtuales, laberintos, simulaciones de deportes, etc-. Durante estos experimentos se analiza la precisión de los movimientos de los individuos, la percepción espacial y la diferencia entre la capacidad de adaptación de los individuos sanos y los individuos con lesiones en el cerebro.

Podemos definir la realidad virtual como una tecnología que permite a los usuarios sumergirse en un entorno generado por computadora que simula la realidad física o crea un mundo virtual completamente nuevo. Los dispositivos de realidad virtual rastrean los movimientos de la cabeza y, en algunos casos, también los movimientos de las manos y del cuerpo, para permitir que los usuarios exploren y manipulen el entorno virtual de manera natural.

Algunos de los dispositivos de realidad virtual son: los cascos de realidad virtual (o Head Mounted Display), que se colocan en la cabeza y cubren los ojos del usuario, ofreciendo una experiencia inmersiva al mostrar imágenes estereoscópicas en 3D; los simuladores de realidad virtual, como los simuladores de vuelo, donde la inmersión es completa y reconoce todos los movi-

mientos de nuestro cuerpo; los dispositivos h谩pticos que simulan respuestas t谩ctiles acordes con las im谩genes que vemos, etc.

Dentro de los experimentos relacionados con el estudio de los movimientos asociados al cerebro utilizando estos dispositivos podemos encontrar los siguientes art铆culos:

En "*The vestibulo-ocular reflex and angular displacement perception in darkness in humans: adaptation to a virtual environment*"[15] los autores utilizan gafas de realidad virtual para medir la respuesta vestibulo-ocular de los sujetos en un escenario de oscuridad. El reflejo vestibulo-ocular (RVO) es un mecanismo que coordina los movimientos oculares con los movimientos de la cabeza para mantener la estabilidad visual. En este caso en el entorno visual se generaron movimientos de rotaci髇 y se midieron los movimientos oculares de los participantes para estudiar si la falta de se˜nales visuales influye en la precisi髇 de la percepci髇 espacial.

En "*Assessing Saccadic Eye Movements With Head-Mounted Display Virtual Reality Technology*"[16] los autores utilizan el dispositivo HTC VIVE Pro Eye[7] (unas gafas de realidad virtual con eye tracking) para estudiar los movimientos oculares sac醡icos, es decir, los desplazamientos r醕pidos de la fijaci髇 de un punto a otro del campo visual.

En "*Cerebellar Contributions to Reach Adaptation and Learning Sensory Consequences of Action*"[8] los autores utilizan un brazo robot que se proyecta en una pantalla, mientras que se oculta la visi髇 del propio brazo. Se utiliza para estudiar los procesos de aprendizaje de dos tipos diferentes de movimientos: los de modelo directo (se asocia un comando motor con sus consecuencias sensoriales) y los de modelo inverso (se asocia un objetivo con el comando motor necesario para alcanzarlo).

Los resultados evidencian que la tecnolog韆 de realidad virtual proporciona un entorno controlado y estandarizado para estudiar los movimientos asociados con el cerebro, facilitando la comparaci髇 entre los participantes y la realizaci髇 de an醠isis cuantitativos. Sin embargo, la mayor韆 de los experimentos realizados hasta el momento se apoyan en el uso de gafas de realidad virtual. Nuestro objetivo en este proyecto es comprobar si los dispositivos h谩pticos ofrecen las capacidades necesarias para llevar a cabo este tipo de experimentos.

1.2. Objetivos

Dentro de los objetivos del proyecto vamos a distinguir entre los objetivos t閏nicos, relacionados con el estudio de cmo los dispositivos h谩pticos podrn usarse en el diseo de experimentos sobre la relacin del cerebro con determinados tipos de movimientos, y los objetivos did醉ticos, relacionados con el aprendizaje de cmo debe llevarse a cabo una experimentacin cientfica, y en concreto con el desarrollo de una aplicacin que utilice un

dispositivo háptico.

■ Objetivos técnicos:

- Estudiar cómo los dispositivos hápticos, en concreto el dispositivo System3D Touch, pueden ser usados para el estudio de movimientos balísticos controlados por el cerebro.
- Estudiar si los datos obtenidos con System3D Touch son lo suficientemente precisos para que el dispositivo pudiera utilizarse en este tipo de experimentos.
- Estudiar qué tipo de datos podemos obtener a través de los experimentos realizados con System3D Touch, así como qué interpretaciones podemos elaborar utilizando dichos datos.

■ Objetivos didácticos:

- Aprender a usar el dispositivo System3D Touch.
- Aprender a utilizar alguna de las APIs de System3D Touch y ser capaz de desarrollar una aplicación que utilice System3D Touch.
- Comprender en profundidad los conceptos relacionados con los cambios de sistemas de coordenadas, y su relación con el espacio afín y el proyectivo, así como sus aplicaciones en informática gráfica.
- Ser capaz de plantear y llevar a cabo un experimento en el que podamos observar los resultados de diferentes sujetos.
- Realizar un análisis de los datos obtenidos.

1.3. Fases del proyecto

El proyecto ha sido desarrollado en cuatro fases: planificación del experimento, desarrollo de la aplicación, obtención de resultados mediante la utilización de la aplicación en distintos usuarios y análisis de los resultados obtenidos.

1. **Planificación del experimento:** en esta etapa se definió el experimento que íbamos a realizar, teniendo en cuenta contenido, fases de las que consistiría y grupos de población en los que íbamos a realizarlo.
2. **Desarrollo de la aplicación:** esta etapa se dedicó a desarrollar la aplicación que se utilizaría después para la realización de los experimentos. A su vez esta etapa se dividió en tres subetapas:
 - a) Instalación de los paquetes necesarios para utilizar el dispositivo System 3D Touch.

- b) Desarrollo del código necesario para implementar la aplicación.
- c) Revisión del código para utilizar los parámetros idóneos. Esta etapa se realizó una vez acabada la primera prueba de realización del experimento.

3. **Obtención de resultados:** en esta etapa distintos usuarios llevaron a cabo el experimento planeado, haciendo uso de la aplicación implementada en la fase anterior. A su vez esta etapa se dividió en dos:

- a) Fase de prueba, en la que se eligió a un primer grupo de sujetos para la realización del experimento, cada uno con un set de parámetros diferentes. Esta fase nos ayudó a decidir los parámetros que queríamos utilizar finalmente.
- b) Fase final, en la que se eligió a un segundo grupo de sujetos (ninguno de los cuales había participado en la fase de prueba) y se realizó el experimento con los parámetros que finalmente habíamos elegido. Estos resultados fueron los que se utilizaron para fase *Análisis de los resultados*.

4. **Análisis de los resultados:** en esta etapa analizamos las gráficas obtenidas con los datos de la etapa anterior, para así obtener conclusiones del experimento.

Por tanto podemos decir que el proyecto ha seguido un modelo de ciclo de vida en cascada, en el que en algunas etapas ha habido realimentación, dado que la información conseguida en algunas etapas nos ha servido para modificar etapas anteriores. Este proceso puede verse reflejado en la figura 1.1:

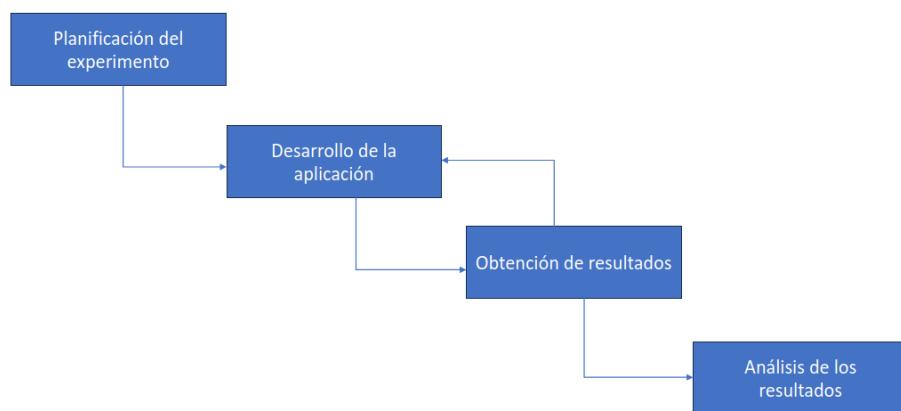


Figura 1.1: Fases del proyecto

Por otro lado la toma de decisiones en las distintas fases se ha llevado a cabo mediante reuniones regulares (telemáticas y presenciales) y la comunicación mediante correo electrónico.

1.4. Planificación del proyecto

El proyecto fue pensado para ser desarrollado entre los meses de septiembre y junio. La planificación inicial fue:

- Septiembre: planificación del experimento.
- De octubre a diciembre: desarrollo de la aplicación.
- De enero a marzo: obtención de resultados y revisión de los parámetros.
- De marzo hasta mayo: análisis de resultados y redacción de la memoria.

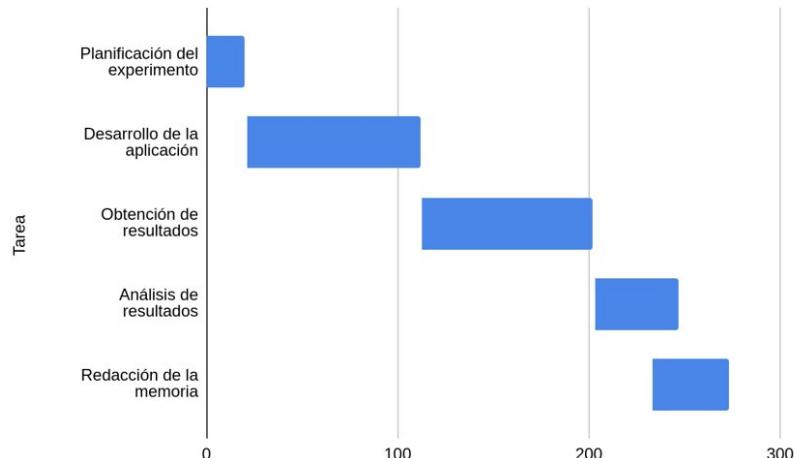


Figura 1.2: Diagrama de Gantt

El reparto de horas respecto a las distintas tareas fue:

- Planificación del experimento: 20 horas.
- Desarrollo de la aplicación:
 - Instalación del software y familiarización con él: 60 horas.
 - Desarrollo del programa: 180 horas.
- Obtención de resultados: 50 horas.

- Análisis de resultados: 70 horas.
- Redacción de la memoria: 70 horas.
- Total de horas: 450 horas.

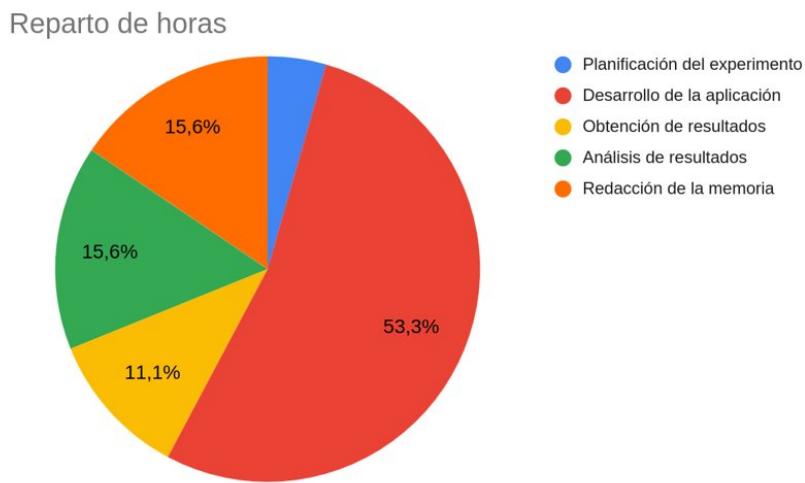


Figura 1.3: Reparto de horas

Como puede verse en la figura 1.3, la etapa más larga fue la de *Desarrollo de la aplicación* pues es la que sustentaba todo el trabajo que podíamos realizar después. Por eso fue la que más carga de trabajo acumuló, y a la que más horas se le dedicó. La etapa de *Obtención de resultados*, sin embargo, no tuvo tanta carga de trabajo, pero como podemos ver en la figura 1.2 la duración de esta tarea se prolongó más en el tiempo, principalmente por la dificultad de encontrar personas de diferentes edades para realizar el experimento.

Por tanto, teniendo en cuenta que el salario medio de un profesional junior es de 20€/hora, el coste en personal sería de 9000€.

Por otro lado tenemos que añadir el coste del dispositivo System3D Touch:

Por último, dada la necesidad de realizar el experimento en individuos externos al proyecto y teniendo en cuenta la duración media de este (45 minutos) decidimos ofrecer una compensación material a los individuos que se presentaran voluntarios, en concepto de coste de transporte. La compensación la hemos fijado en 10€/persona, y dado que hemos tenido 13 voluntarios (8 para la fase de prueba y 5 para la fase final), el coste adicional será de 130€.

Por tanto el presupuesto final fue de – euros, repartido como puede verse en la figura 1.4:

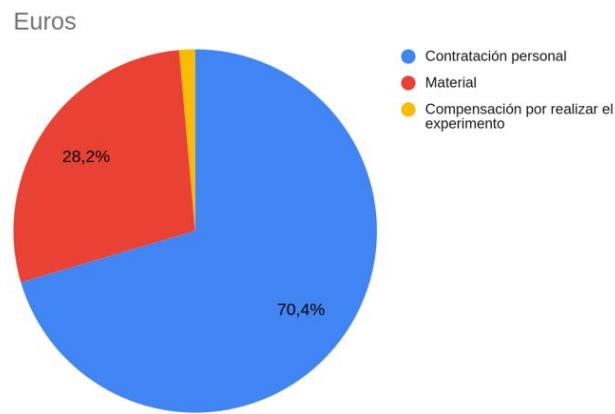


Figura 1.4: Coste del proyecto

1.5. Material

En esta sección vamos a hablar del material (hardware y software) que se ha utilizado para la realización de este proyecto: el dispositivo Touch y el software de OpenHaptics.

1.5.1. Touch

Touch[13] ha sido el dispositivo sobre el que se ha hecho la aplicación, y que se ha utilizado posteriormente para la realización del experimento.

Es un dispositivo háptico desarrollado por la empresa 3D Systems[1]. Los dispositivos hápticos simulan respuestas táctiles, mediante las cuales podemos percibir objetos tridimensionales en un ambiente de realidad virtual. Según la propia documentación de 3D Systems: *"Touch es un dispositivo motorizado que aplica retroalimentación de fuerza a la mano del usuario, lo que le permite sentir objetos virtuales y producir sensaciones táctiles reales a medida que el usuario manipula los objetos 3D en la pantalla"*. Touch es utilizado en aplicaciones de control robótico, rehabilitación, medicina y cirugía, etc.

Especificaciones técnicas

Área de trabajo y retroalimentación de fuerza	431 ancho. × 348 altura. × 165 profundidad (mm).
Tamaño (espacio físico que la base del dispositivo ocupa en una superficie)	168 ancho x 203 profundidad (mm).
Peso (solo dispositivo)	1,42 kg
Rango de movimiento	Movimiento de la mano con giro de la muñeca
Resolución de la posición nominal	0,055 mm
Fricción de accionamiento trasero	< 0,26 N
Fuerza de esfuerzo máxima (en la posición nominal de los brazos ortogonales)	3,3 N
Fuerza de esfuerzo continua (24 h)	> 0,88 N
Rigidez	Eje X > 1,26 N/mm Eje Y > 2,31 N/mm Eje Z > 1,02 N/mm
Inercia (masa aparente en la punta)	~ 45 g
Retroalimentación de fuerza	x, y, z
Detección de posición	x, y, z (codificadores digitales)
Cardán de lápiz	Inclinación, giro, dirección ($\pm 5\%$ de potenciómetros de linealidad)
Interfaz	USB 2.0/puerto 3.0 o hub USB que admita USB 2.0/3.0. Frecuencia de actualización 1 KHz.

Cuadro 1.1: Especificaciones del dispositivo Touch

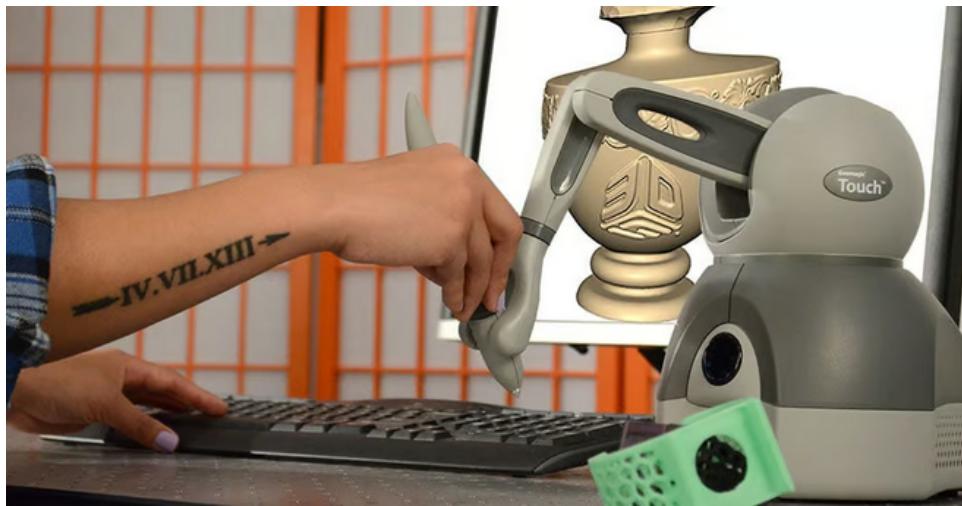


Figura 1.5: Dispositivo System3D Touch

1.5.2. OpenHaptics

OpenHaptics[12] es un software desarrollado por 3D Systems para desarrolladores hapticos que permite crear aplicaciones que utilicen Touch. Está basado en la API de OpenGL[6] y está disponible para Windows de 64 bits 10 y 11 y para Linux.

Según la propia página web de 3D Systems: *OpenHaptics permite a los desarrolladores de software incorporar tecnología haptica y navegación 3D real a una amplia gama de aplicaciones, incluidas las de modelado y diseño 3D, aplicaciones médicas, juegos, entretenimiento, visualización y simulación. Este kit de herramientas hapticas se basa en la API de OpenGL, lo que hace que sea perfecto para programadores gráficos y facilita la integración con las aplicaciones de OpenGL. Con el kit de herramientas OpenHaptics, los desarrolladores pueden aprovechar el código de OpenGL existente para especificar geometrías y complementarlo con los comandos de OpenHaptics para imitar las propiedades de materiales hapticos, como la fricción y la rigidez.*

OpenHaptics incluye varias APIs desarrolladas en C++, y viene con varios ejemplos de aplicaciones, una guía de usuario y dos guías para los desarrolladores: OpenHaptics Programmers Guide y OpenHaptics API Reference Guide.

Capítulo 2

Fundamentos matemáticos

Un proceso clave a la hora de desarrollar la aplicación ha sido trasladar las coordenadas obtenidas a través del dispositivo Touch a la pantalla del ordenador. De esta forma hemos podido obtener las trayectorias que cada sujeto ha realizado tal y como ellos las han visto reflejadas en la pantalla.

Para explicar el proceso de transformación de coordenadas vamos a introducir previamente algunos conceptos matemáticos que necesitaremos después. Las transformaciones que aplicaremos incluyen traslaciones, rotaciones, escalados y proyecciones. Para estudiar estos movimientos introduciremos dos conceptos matemáticos muy importantes: las transformaciones afines y las proyecciones. Estos conceptos se basan en el estudio de la Geometría afín y proyectiva, que a su vez tienen una base en el Álgebra lineal.

2.1. Álgebra lineal, Geometría afín y Geometría Proyectiva

Podemos encontrar los primeros antecedentes del Álgebra lineal en el *Libro del Cálculo*, escrito por el sacerdote egipcio Ahmés hacia el año 1650 a.C. En él se describen algunos problemas relacionados con la agricultura, que involucran sistemas de ecuaciones de primer grado. Los matemáticos babilonios, griegos, chinos e islámicos siguieron estudiando estos tipos de problemas, pero no fue hasta el siglo XIX cuando Hamilton, Cayley y Grassmann introdujeron las nociones de vector y espacio vectorial, desarrollando gran parte de los conceptos que estudiaremos en este capítulo. En el año 1850 Sylvester introdujo el término matriz, y en 1858, en su libro *Memoir on the theory of matrices*, Cayley utiliza las matrices para explicar las transformaciones lineales [3].

La geometría afín surge con Euler y se consolida con Klein en 1872 mediante el Programa de Erlangen, un programa de investigación con el objetivo de proponer una solución a los problemas que la Geometría se enfrentaba en ese momento. Klein define la geometría afín como una generalización de

la geometría asociada al álgebra lineal. El espacio afín se construye sobre el espacio vectorial, con la diferencia de que no tenemos un vector de referencia, lo que nos dará la capacidad de hacer traslaciones. A su vez Klein engloba la geometría afín dentro de la geometría proyectiva.

En el Programa de Erlangen, Klein también introduce el concepto de Grupo (un conjunto en el que hay definido una operación que cumple ciertas propiedades) en la Geometría. Para Klein cada Geometría es el estudio de las propiedades que no cambian al aplicar un tipo de transformaciones. Por ejemplo, el álgebra lineal estudia las propiedades de los espacios al aplicar movimientos lineales (escalados, rotaciones sobre el origen), la geometría afín al aplicar movimientos afines (traslaciones) y la geometría proyectiva al aplicar movimientos proyectivos [4].

En este capítulo vamos a introducir primero el concepto de espacio vectorial para después definir el concepto de espacio afín. Sobre el espacio afín estudiaremos como representar los movimientos afines en forma de matrices, y el concepto de sistema de referencia y por último cómo realizar un cambio de un sistema de referencia a otro. Este último resultado será el que utilizaremos a la hora de explicar las operaciones que hemos realizado para hacer los cambios de sistemas de coordenadas.

Por último estudiaremos brevemente el espacio proyectivo para poder explicar la proyección en perspectiva que realizaremos a la hora de proyectar el movimiento del cursor del dispositivo Touch en la pantalla.

2.1.1. Álgebra lineal

El Álgebra lineal estudia los conceptos de espacios vectoriales y sus transformaciones lineales. En esta sección nuestro objetivo es definir el concepto de espacio vectorial, que utilizaremos para definir después el espacio afín. Para ello necesitamos definir previamente los conceptos de grupo, anillo y cuerpo.

También introduciremos el concepto de Base en un espacio vectorial, para el que necesitaremos definir los conceptos de combinación lineal de vectores, vectores linealmente independientes y sistemas de generadores. Por último terminaremos la sección con una proposición que nos permitirá hacer cambios de base con los vectores [2].

Definición 1. *Sea G un conjunto. Decimos que (G, \cdot) es un **grupo** si existe una operación o ley de composición interna:*

$$\begin{aligned} \cdot : G \times G &\rightarrow G \\ (x, y) &\mapsto x \cdot y \end{aligned} \tag{2.1}$$

que cumpla las siguientes propiedades:

1. *Asociativa:* $x \cdot (y \cdot z) = (x \cdot y) \cdot z, \forall x, y, z \in G.$

2. *Elemento neutro:* $\exists e \in G$ (*elemento neutro*) tal que $e \cdot x = x \cdot e = x$, $\forall x \in G$.
3. *Elemento simétrico.* $\forall x \in G$, $\exists \bar{x} \in G$ (*elemento simétrico de x*) tal que $x \cdot \bar{x} = \bar{x} \cdot x = e$.

Si además cumple:

- *Commutativa:* $x \cdot y = y \cdot x$, $\forall x, y \in G$.

*diremos que el grupo es **abeliano**.*

Definición 2. *Diremos que $(A, +, \cdot)$ es un **anillo** si verifica:*

1. $(A, +)$ es un grupo conmutativo.
2. (A, \cdot) verifica la propiedad asociativa.
3. $(A, +, \cdot)$ verifica la propiedad distributiva de la suma respecto al producto, esto es,

$$x \cdot (y + z) = x \cdot y + x \cdot z = (y + z) \cdot x = y \cdot x + z \cdot x \quad (2.2)$$

para todo $x, y, z \in A$.

Un anillo unitario no trivial es un anillo cuyo producto tiene elemento neutro distinto de cero.

Definición 3. *Un **cuerpo** $(K, +, \cdot)$ es un anillo unitario no trivial en el que el producto verifica la propiedad elemento simétrico. El cuerpo es conmutativo si el producto \cdot verifica la propiedad conmutativa.*

Definición 4. *Sea $(K, +, \cdot)$ un cuerpo conmutativo. Un **espacio vectorial** sobre $(K, +, \cdot)$ es una terna $(V, +, \cdot K)$ formada por un conjunto V dotado de una operación (ley de composición interna) en V ,*

$$+ : V \times V \rightarrow V \quad (2.3)$$

y una aplicación

$$\cdot : K \times V \rightarrow V \quad (2.4)$$

o ley de composición externa de K sobre V tales que:

1. $(V, +)$ es un grupo conmutativo. Esto es, la operación $+$ en V , que asocia a cada par $(u, v) \in V \times V$ un único $u + v \in V$, verifica las propiedades de grupo abeliano.
2. La ley de composición externa $\cdot : K \times V \rightarrow V$, que asocia a cada $a \in K$ y cada $v \in V$ un único vector que denotaremos $a \cdot v \in V$, verifica las propiedades:

- a) *Pseudoasociativa:* $(a \cdot b) \cdot v = a \cdot (b \cdot v)$ para todo $a, b \in K$ y todo $v \in V$.
- b) *Unimodular:* $1 \cdot v = v$, para todo $v \in V$, donde 1 es el elemento neutro de $(K \setminus \{0\}, \cdot)$.
- c) *Distributiva respecto de la suma en K :* $(a + b) \cdot v = a \cdot v + b \cdot v$, para cualesquiera $a, b \in K$ y $v \in V$.
- d) *Distributiva respecto de la suma en V :* $a \cdot (u + v) = a \cdot u + a \cdot v$, para todo $a \in K$ y cualesquiera $u, v \in V$.

Definición 5. Sea $V(K)$ un espacio vectorial y $S = \{v_1, \dots, v_m\}$ una familia finita no vacía de vectores de V . Una **combinación lineal** de S es cualquier vector de V obtenido al multiplicar cada v_i por un escalar $a_i \in K$ y después sumar los vectores resultantes:

$$a_1 \cdot v_1 + \dots + a_m \cdot v_m, \text{ donde } a_i \in K \quad \forall i = 1, \dots, m. \quad (2.5)$$

Llamaremos $L(S)$ al subconjunto de V formado por los vectores obtenidos como combinación lineal de S :

$$L(S) = L(v_1, \dots, v_m) = \{a_1 \cdot v_1 + \dots + a_m \cdot v_m : a_i \in K, \forall i = 1, \dots, m\}. \quad (2.6)$$

Definición 6. Sea $V(K)$ y $S = \{v_1, \dots, v_m\}$ una familia finita no vacía de vectores de V . Diremos que S es **linealmente independiente** si se cumple:

1. Caso $m = 1$: $v_1 \neq 0$.
2. Caso $m \geq 2$: ningún vector de S es combinación lineal de los restantes vectores de S , esto es, $v_i \notin L(S - \{v_i\})$, para cada $i = 1, \dots, m$.

Definición 7. Sea $V(K)$ y sea $S \subset V$ un conjunto no vacío. Se dice que S es un **sistema de generadores** si $V = L(S)$. Esto equivale a que todo vector de V se expresa como combinación lineal finita de vectores de S , es decir, para cada $v \in V$, existen $m \in N$, vectores $v_1, \dots, v_m \in S$ y escalares $a_1, \dots, a_m \in K$, tales que $v = a_1 \cdot v_1 + \dots + a_m \cdot v_m$.

Definición 8. Sea V un espacio vectorial sobre un cuerpo K . Una **base** de V es una familia no vacía $B \subset V$ tal que B es un sistema de generadores de V y B es linealmente independiente.

Definición 9. Supongamos V un espacio vectorial sobre un cuerpo K con $\dim_k(V) = n$. Tomamos dos bases $B = \{v_1, \dots, v_n\}$ y $B' = \{v'_1, \dots, v'_n\}$ de V y $v \in V$. La **matriz de cambio** de base de B a B' es la matriz que denotaremos $M(I, B' \leftarrow B)$ cuya columna j -ésima contiene las coordenadas del vector v_j de B respecto de B' . Lo simbolizamos así:

$$M(I, B' \leftarrow B) = ((v_1)_{B'} | \dots | (v_n)_{B'}) \quad (2.7)$$

Proposición 1. En las condiciones anteriores se tiene que:

$$v_{B'} = M(I, B' \leftarrow B) \cdot v_B \quad (2.8)$$

Demostración. Supongamos que:

$$v_j = a_{1j} \cdot v'_1 + \dots + a_{nj} \cdot v'_n, \text{ para cada } j = 1, \dots, n \quad (2.9)$$

De esta forma la j -ésima columna de $M(I, B' \leftarrow B)$ contiene exactamente a los escalares a_{ij} con $i = 1, \dots, n$. Supongamos también que:

$$v = x_1 \cdot v_1 + \dots + x_n \cdot v_n \quad (2.10)$$

es decir, v_B contiene a los escalares x_i con $i = 1, \dots, n$. Buscamos expresar v como combinación lineal de B' . Sustituyendo la primera ecuación en la segunda tenemos la igualdad:

$$\begin{aligned} v &= x_1 \cdot (a_{11} \cdot v'_1 + \dots + a_{n1} \cdot v'_n) + \dots + x_n \cdot (a_{1n} \cdot v'_1 + \dots + a_{nn} \cdot v'_n) \\ &= (a_{11} \cdot x_1 + \dots + a_{n1} \cdot x_n) \cdot v'_1 + \dots + (a_{1n} \cdot x_1 + \dots + a_{nn} \cdot x_n) \cdot v'_n \end{aligned} \quad (2.11)$$

que expresa v como combinación lineal de B' . Esto significa que los coeficientes de la combinación lineal anterior son las coordenadas de v respecto de B' . Por definición de producto de matrices, la entrada i -ésima de $v_{B'}$ es el producto de la fila i -ésima de $M(I, B' \leftarrow B)$ con el vector columna v_B . \square

2.1.2. Geometría afín

En esta sección vamos a definir la base teórica para realizar cambios de sistemas de coordenadas. Las transformaciones que aplicaremos son escalados, giros y traslaciones, y para ello definiremos el concepto de espacio afín. Despues definiremos el concepto de sistema de referencia afín para terminar demostrando la fórmula del cambio de sistema de referencia, que es la que utilizaremos en el apartado práctico [5].

Definición 10. Un *espacio afín* es una tripleta (A, V, \rightarrow) donde

- A es un conjunto no vacío
- $V \equiv (V, +, \cdot \mathbb{R})$ es un espacio vectorial real
- $\rightarrow : A \times A \rightarrow V, (p, q) \mapsto \vec{pq}$, es una aplicación satisfaciendo
 - $A_1 : \vec{pq} + \vec{qr} = \vec{pr}$.
 - $A_2 : \forall p \in A, \forall v \in V, \exists! q \in A : \vec{pq} = v$.

Al espacio vectorial V lo vamos a denotar como \overrightarrow{A} . La dimensión de A es la de \overrightarrow{A} .

Definición 11. Una colección de puntos $\{p_0, \dots, p_k\}, k \in K$, en un espacio afín A se dice **afínmente independiente** si los vectores $\{\overrightarrow{p_0p_1}, \dots, \overrightarrow{p_0p_k}\}$ son linealmente independientes, es decir, si:

$$\dim \langle \{p_0, p_1, \dots, p_k\} \rangle = k \quad (2.12)$$

Definición 12. Dado un espacio afín A con $\dim A = n \in \mathbb{N}$, un **sistema de referencia** R en A es un sistema ordenado $\{p_0, \dots, p_n\}$ de $n+1$ puntos afínmente independientes, o equivalentemente satisfaciendo:

$$\langle \{p_0, p_1, \dots, p_n\} \rangle = A \quad (2.13)$$

Al punto p_0 se le llama *origen* del sistema, y a la base ordenada $B = \{\overrightarrow{p_0p_1}, \dots, \overrightarrow{p_0p_n}\}$ de \vec{A} se le llama *base asociada de las direcciones* de R . Si $B = \{v_1, \dots, v_n\}$ es una base ordenada de \vec{A} y $p_0 \in A$, el sistema ordenado de puntos

$$R = \{p_0, p_0 + v_1, \dots, p_0 + v_n\} \quad (2.14)$$

es un sistema de referencia de A con origen p_0 y base asociada de direcciones B . Por tanto, y de forma alternativa, podríamos definir un sistema de referencia como un par

$$R = \{p_0, B\} \quad (2.15)$$

donde p_0 es un punto de A y B una base ordenada de \vec{A} .

Definición 13. Fijado $p \in A$, denotaremos por $F_p : A \rightarrow \vec{A}$ a la aplicación dada por:

$$F_p(q) = \overrightarrow{pq} \quad (2.16)$$

Definición 14. Sea A un espacio afín, y consideremos un sistema de referencia afín $R = \{p_0, \dots, p_n\} \equiv \{p_0, B\}$, donde $B = \{\overrightarrow{p_0p_1}, \dots, \overrightarrow{p_0p_n}\}$. La aplicación biyectiva

$$\begin{aligned} \Phi_R : A &\rightarrow \mathbb{R}^n \\ \phi_R &= \phi_B \circ F_{p_0} \end{aligned} \quad (2.17)$$

es conocida como la **aplicación asignación de coordenadas** (con notación columna) en el sistema de referencia R . De forma simplificada escribiremos

$$p_R := \Phi_B(F_{p_0}(p)) \equiv (\overrightarrow{p_0p})_B \in \mathbb{R}^n, \quad (2.18)$$

y diremos que p_R son las coordenadas de $p \in A$ en R .

Proposición 2. Sea A un espacio afín y $R = \{p_0, \dots, p_n\} \equiv \{p_0, B\}$ un sistema de referencia afín, donde $B = \{\overrightarrow{p_0p_1}, \dots, \overrightarrow{p_0p_n}\}$. Entonces

1. $(p + v)_R = p_R + v_B$

$$2. (\overrightarrow{pq})_B = q_R - p_R$$

Demostración. Usando la definición 14 y que Φ_B es lineal, tenemos que:

$$(p + v)_R = (\overrightarrow{p_0(p+v)})_B = (\overrightarrow{p_0p} + v)_B = (\overrightarrow{p_0p})_B + v_B = p_R + v_B \quad (2.19)$$

lo que prueba (i). Análogamente,

$$q_R - p_R = (\overrightarrow{p_0q})_B - (\overrightarrow{p_0p})_B = (\overrightarrow{p_0q} - \overrightarrow{p_0p})_B = (\overrightarrow{pp_0} + \overrightarrow{p_0q})_B = (\overrightarrow{pq})_B \quad (2.20)$$

lo que prueba (ii). \square

Proposición 3. *La fórmula del cambio de sistema de referencia es:*

$$p_{R'} = (p_0)_{R'} + M(Id_{\vec{A}}, B, B')p_R \quad (2.21)$$

Demostración. Utilizando la proposición 2 y la definición 14:

$$p_{R'} = (p_0 + \overrightarrow{p_0p})_{R'} = (p_0)_{R'} + (\overrightarrow{p_0p})_{B'} = (p_0)_{R'} + M(Id_{\vec{A}}, B, B') \cdot (\overrightarrow{p_0p})_B, \quad (2.22)$$

y por tanto,

$$p_{R'} = (p_0)_{R'} + M(Id_{\vec{A}}, B, B')p_R \quad (2.23)$$

\square

Observación 1. *La ecuación anterior puede escribirse de forma compacta utilizando una única matriz de orden $n + 1$. Esta matriz es de la forma:*

$$M(Id_A, R, R') := \begin{pmatrix} M(Id, B, B') & (p_0)_{R'} \\ 0 & 1 \end{pmatrix} \quad (2.24)$$

Podemos escribir la fórmula del cambio de sistema de referencia como:

$$\begin{pmatrix} 1 \\ p_{R'} \end{pmatrix} = M(Id_A, R, R') \cdot \begin{pmatrix} 1 \\ p_R \end{pmatrix} \quad (2.25)$$

Por tanto hemos puesto la base para conocer las coordenadas en un sistema de referencia R' de un punto $p \in A$ a partir de las coordenadas de p en otro sistema de referencia R teniendo:

- Coordenadas en R' del origen p_0 de R .
- Matriz del cambio de base en \vec{A} de la base B de las direcciones de R a la base B' de las direcciones de R' .

2.1.3. Espacio Proyectivo

En esta sección vamos a definir los principios básicos del espacio proyectivo, que utilizaremos como base teórica para poder explicar después la proyección en perspectiva que aplicaremos a las coordenadas para representarlas en la pantalla del ordenador [5].

Antes de definir el espacio proyectivo necesitamos dos conceptos previos: relación de equivalencia y espacio cociente [2].

Definición 15. *R es una relación de equivalencia sobre K si:*

1. $\forall x \in K : x R x$
2. $\forall x, y \in K : x R y \Rightarrow y R x$
3. $\forall x, y, z \in K : x R y, y R z \Rightarrow x R z.$

Definición 16. *Dado X un espacio vectorial y R una relación de equivalencia, el espacio cociente generado por ella está definido por:*

$$X \setminus \sim := \{[x] : x \in X\} \quad (2.26)$$

Nota: A partir de ahora llamaremos $E^* = E \setminus \{0\}$.

Definición 17. *Sea E un espacio vectorial con $\dim_E = n + 1$, llamamos $P(E)$, espacio proyectivo, al espacio cociente*

$$P(E) = E^* / \sim \quad (2.27)$$

siendo la relación de equivalencia:

$$v \sim w \Leftrightarrow v = \lambda w \text{ para algún } \lambda \in \mathbb{R}^* \quad (2.28)$$

Llamaremos $[v] \in P(E)$ a la clase de equivalencia de $v \in E^*$, es decir, al punto de $P(E)$ determinado por la recta vectorial generada por v :

$$[v] = L(\{v\})^* := \{\lambda v : \lambda \in \mathbb{R}^*\} \quad (2.29)$$

Por lo que podemos decir que los puntos de $P(E)$ son rectas vectoriales de E .

El concepto de coordenadas en $P(E)$ es diferente, ya que no hay un concepto natural de base. Es por eso por lo que se introducen las coordenadas homogéneas.

Definición 18. *Las coordenadas homogéneas de un punto $p \in P(E)$ en la base $B = \{v_0, v_1, \dots, v_n\}$ de E las definimos como:*

$$p_B = \{\lambda v_B : \lambda \in \mathbb{R}^*\} \quad (2.30)$$

donde $v_B \in (\mathbb{R}^{n+1})$ son las coordenadas en B de cualquier vector v no nulo tal que $[v] = p$. Si $v_B = (x_0, v_1, \dots, x_n)$, escribiremos:

$$p_B = (x_0 : x_1 : \dots : x_n) := \{\lambda(x_0, x_1, \dots, x_n) : \lambda \in \mathbb{R}^*\} \quad (2.31)$$

Nuestro objetivo ahora es poder relacionar el espacio euclíadiano \mathbb{R}^n con el proyectivo \mathbb{P}^n .

Podemos ver que todo espacio afín A se puede ver como un hiperplano afín que no pasa por el origen en un espacio vectorial E de una dimensión mayor. Esto lo podemos conseguir fijando un origen $e_0 \in A$ y definiendo $E = \mathbb{R} \times A$:

$$A \ni p \longleftrightarrow (1, \overrightarrow{e_0 p}) \in \{1\} \times \overrightarrow{A} \subset E^* \quad (2.32)$$

De esta forma llevamos el origen e_0 al punto $(1, 0)$.

Siguiendo este razonamiento podemos definir el hiperplano $A \subseteq \mathbb{R}^{n+1}$ donde $\{x_0 = 1\}$ como:

$$A = \{1\} \times \mathbb{R}^n \subseteq \mathbb{R}^{n+1} \quad (2.33)$$

e identificar \mathbb{R}^n con A de la siguiente forma:

$$\mathbb{R}^n \rightarrow A, (x_1, \dots, x_n) \mapsto (1, x_1, \dots, x_n) \quad (2.34)$$

El embebimiento canónico de \mathbb{R}^n en $\mathbb{P}^n = P(\mathbb{R}^{n+1})$ queda:

$$e : \mathbb{R}^n \rightarrow A, e((x_1, \dots, x_n)) = (1 : x_1 : \dots : x_n) \quad (2.35)$$

Por tanto la relación entre el espacio euclíadiano \mathbb{R}^n y el proyectivo \mathbb{P}^n es:

$$\mathbb{R}^n \xrightarrow{e} \mathbb{P}^n \setminus \mathbb{R}_{\infty}^n, (x_1, \dots, x_n) \mapsto (1 : x_1 : \dots : x_n) \quad (2.36)$$

$$\mathbb{P}^n \setminus \mathbb{R}_{\infty}^n \xrightarrow{e^{-1}} \mathbb{R}^n, (x_0 : x_1 : \dots : x_n) \mapsto \left(\frac{x_1}{x_0}, \dots, \frac{x_n}{x_0} \right) \quad (2.37)$$

donde $\mathbb{R}_{\infty}^n = \{(0 : x_1 : \dots : x_n) : (x_1 : \dots : x_n) \in \mathbb{P}^{n-1}\}$

Esto último es lo que utilizaremos para hacer la proyección que necesitamos para transformar las coordenadas del dispositivo a la pantalla.

2.2. Aplicaciones en informática gráfica

El problema que queremos resolver es: teniendo las coordenadas que nos proporciona el dispositivo Touch, ¿qué transformaciones tenemos que hacer para obtener las coordenadas del cursor de Touch en la pantalla?

Para ello tenemos que entender los distintos sistemas de coordenadas que utiliza OpenGL y el cauce de transformaciones que aplica a los objetos para representarlos en la pantalla[10]. Podemos verlo representado en la figura 2.1:

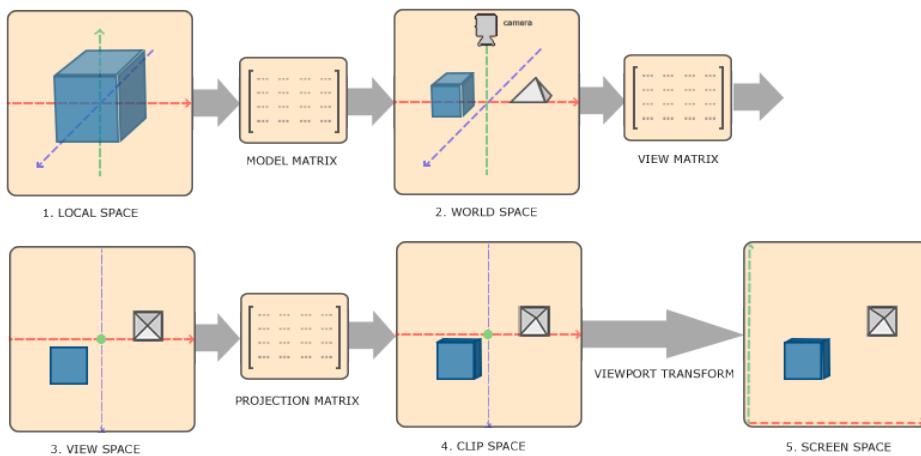


Figura 2.1: Sistemas de coordenadas de OpenGL

Podemos ver que OpenGL utiliza diferentes sistemas de coordenadas:

- **Coordenadas locales** (local coordinates): coordenadas del objeto relativas a su propio origen.
- **Coordenadas del mundo** (world coordinates): coordenadas del objeto relativas a un origen global del sistema, donde están situados los demás objetos.
- **Coordenadas de vista** (view coordinates): coordenadas del objeto desde el punto de vista de la cámara.
- **Coordenadas de recorte** (clip coordinates): coordenadas respecto a un cubo virtual desde el que determinaremos qué vértices se verán en la pantalla. El cubo tiene dimensiones $[-1, 1] \times [-1, 1] \times [-1, 1]$.
- **Coordenadas de pantalla** (screen coordinates): coordenadas del objeto tal y como lo vemos representado en la pantalla. El origen de las coordenadas de pantalla está situado en la en la esquina inferior izquierda de la pantalla.

Además en la aplicación que vamos a desarrollar queremos que se representen una serie de puntos. Estos puntos estarán situados sobre un nuevo sistema de referencia, con un punto que llamaremos "de referencia", situado en el origen, y los demás puntos que llamaremos "targets", situados en una circunferencia de radio 2. Sobre este sistema de referencia realizaremos el análisis de los datos que obtengamos al realizar el experimento, y a estas coordenadas las vamos a llamar coordenadas finales.

- **Coordenadas finales:** sistema de coordenadas que hemos introducido nosotros y que utilizaremos para situar de una forma más cómoda los puntos que representaremos en la aplicación.

Las coordenadas que Touch nos proporciona están en el sistema de referencia local. Por lo tanto nuestro objetivo es estudiar las transformaciones que se realizan para, a partir de esas coordenadas, obtener las coordenadas finales.

Como hemos estudiado en la sección anterior las transformaciones entre sistemas de referencia se realizan a través de la multiplicación de las matrices de cambio de sistema de referencia. Estas matrices en OpenGL tienen los siguientes nombres:

- **Matriz modelo** (model matrix): matriz de cambio del sistema de referencia local al sistema de referencia del mundo.
- **Matriz de vista** (view matrix): matriz de cambio del sistema de referencia del mundo al sistema de referencia de vista.
- **Matriz de proyección** (projection matrix): matriz de cambio del sistema de referencia de vista al sistema de referencia de recorte.
- **Viewport**: matriz de cambio de sistema de referencia de recorte al sistema de referencia de pantalla.

OpenGL combina la matriz modelo y la matriz vista en una matriz llamada **modelview**. Además, como hemos introducido otro sistema de referencia, tenemos que definir otra matriz para pasar de sistemas de coordenadas de pantalla a sistema de coordenadas finales. Esta matriz la llamaremos la **matriz final**.

Por tanto el diagrama que vamos a seguir es:

1. Aplicamos las transformaciones de OpenGL, es decir:

$$C.\text{Locales} \rightarrow C.\text{Vista} \rightarrow C.\text{Recorte} \rightarrow C.\text{Pantalla} \quad (2.38)$$

2. Aplicamos la matriz final para obtener las coordenadas finales

$$C.\text{Pantalla} \rightarrow C.\text{Finales} \quad (2.39)$$

Vamos a definir ahora la notación que vamos a utilizar para representar los distintos sistemas de coordenadas:

- R : sistema de coordenadas locales
- R' : sistema de coordenadas de vista
- R'' : sistema de coordenadas de recorte
- R''' : sistema de coordenadas de pantalla

- R^{iv} : sistema de coordenadas finales

Y las correspondientes matrices de cambio de coordenadas son:

- (Id, R', R) : modelview
- (Id, R'', R') : proyección
- (Id, R''', R'') : viewport
- (Id, R^{iv}, R''') : matriz final

En las siguientes secciones vamos a tomar un punto v en el sistema de coordenadas R y vamos a estudiar las transformaciones que OpenGL le aplica para obtener el punto en el sistema de coordenadas R^{iv} . Para ello, utilizando la función `glGetDoublev` de `Gl.h`, obtendremos las matrices que OpenGL aplica en cada paso y las analizaremos utilizando los conceptos que hemos definido previamente.

2.2.1. Paso de coordenadas locales a coordenadas de vista

OpenGL aplica la transformación de coordenadas locales a coordenadas de vista utilizando una única matriz: la matriz modelview. En este paso vamos a tomar p_R y queremos obtener $p_{R'}$ utilizando (Id, R', R) , que será la matriz modelview. La matriz modelview que utiliza la aplicación es:

$$(Id, R', R) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -6.49 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

y, utilizando la proposición 2.25 el cambio de sistema de coordenadas será:

$$\begin{pmatrix} 1 \\ p_{R'} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -6.49 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ p_R \end{pmatrix}$$

Como hemos explicado antes, esto es una notación abreviada de:

$$p_{R'} = \begin{pmatrix} 0 \\ 0 \\ -6.49 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot p_R \quad (2.40)$$

Por tanto lo que estamos haciendo es una transformación afín, en concreto una traslación del origen del sistema de coordenadas en el eje z.

2.2.2. Paso de coordenadas de vista a coordenadas de recorte

Ahora vamos a hacer el cambio de sistemas de coordenadas de vista al sistema de coordenadas de recorte.

El objetivo de este cambio de sistema de coordenadas es transformar la región visible del espacio en un cubo $[-1, 1] \times [-1, 1] \times [-1, 1]$, utilizando una proyección en perspectiva, como podemos ver en la figura 2.2.

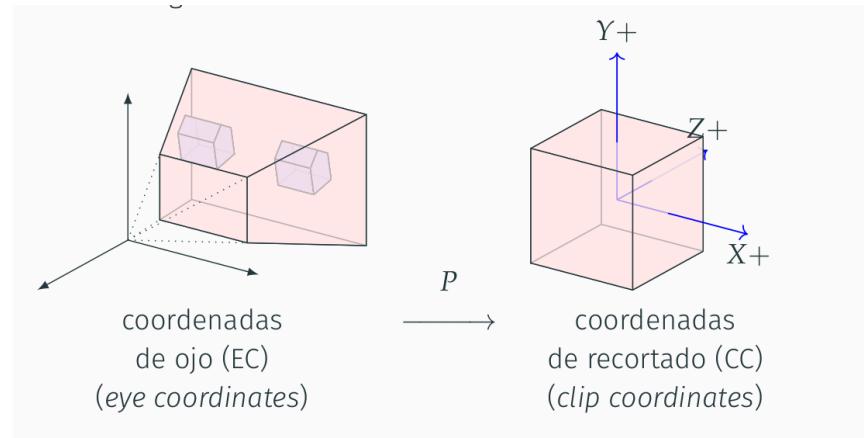


Figura 2.2: Proyección en perspectiva

La notación que utilizaremos la podemos ver en la figura 2.3:

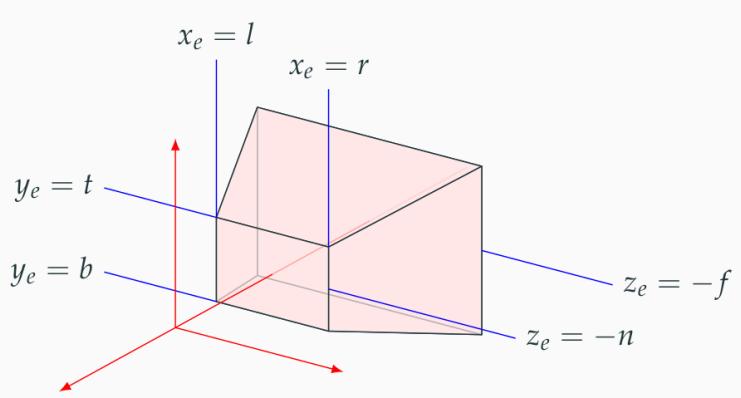


Figura 2.3: Notación

Para ello primero calcularemos la proyección en el eje z de las coordenadas x, y . Como hemos estudiado antes, para obtener la proyección sobre el plano euclídeo tenemos que dividir por la última coordenada. En este caso como además queremos que los puntos se proyecten sobre el valor $z = -n$, en lugar de $z = 1$, tendremos que multiplicar por $-n$. Si tenemos (x', y', z')

en el sistema de referencia R' .

$$x_e'' = x' \cdot \frac{-n}{z'} \quad (2.41)$$

$$y_e'' = y' \cdot \frac{-n}{z'} \quad (2.42)$$

$$z_e'' = z' \cdot \frac{-n}{z'} = -n \quad (2.43)$$

Tenemos que (x_e'', y_e'') sería el punto que obtendríamos sobre el plano euclídeo (plano situado sobre $z = -n$).

Sabemos ahora que $x_e'' \in [l, v]$ y $y_e'' \in [b, t]$. Para conseguir que ambos estén en el intervalo $[-1, 1]$ tenemos que aplicar un escalado y una traslación.

$$x'' = 2 \cdot \left(\frac{x_e'' - l}{r - l} \right) - 1 \quad (2.44)$$

$$y'' = 2 \cdot \left(\frac{y_e'' - b}{t - b} \right) - 1 \quad (2.45)$$

Sustituyendo en las ecuaciones anteriores, y utilizando las constantes a_0, a_1, b_0, b_1 .

$$x'' = \frac{a_0 \cdot x' + a_1 \cdot z'}{-z'} - a_1 = \frac{a_0 \cdot x' + a_1 \cdot z'}{-z'} \quad (2.46)$$

$$y'' = \frac{b_0 \cdot y' + b_1 \cdot z'}{-z'} - b_1 = \frac{b_0 \cdot y' + b_1 \cdot z'}{-z'} \quad (2.47)$$

donde

$$a_0 = \frac{2n}{r - l} \quad (2.48)$$

$$a_1 = \frac{r + l}{r - l} \quad (2.49)$$

$$b_0 = \frac{2n}{t - b} \quad (2.50)$$

$$b_1 = \frac{t + b}{t - b} \quad (2.51)$$

Estas transformaciones no se pueden implementar en una matriz, ya que no son lineales (aparece un denominador). Para poder escribirlo en forma de matriz vamos a hacer uso de una coordenada adicional, que nos va a servir para guardar el término no lineal, en este caso $-z$:

$$x'' = a_0 \cdot x' + a_1 \cdot z' \quad (2.52)$$

$$y'' = b_0 \cdot y' + b_1 \cdot z' \quad (2.53)$$

$$w'' = -z' \quad (2.54)$$

La matriz que transforma las coordenadas (x', y', z') en las coordenadas (x'', y'', w'') es:

$$\begin{pmatrix} x'' \\ y'' \\ w \end{pmatrix} = \begin{pmatrix} a_0 & 0 & a_1 \\ 0 & b_0 & b_1 \\ 0 & 0 & -1 \end{pmatrix} * \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$$

De esta forma, una vez aplicadas las transformaciones afines, podemos obtener las coordenadas del plano de la siguiente forma:

$$(x'', y'', w'') \rightarrow \left(\frac{x''}{w''}, \frac{y''}{w''} \right) \quad (2.55)$$

Con este procedimiento tendríamos calculadas las coordenadas x, y de pantalla. Sin embargo OpenGL utiliza EPO (eliminación de partes ocultas) para dibujar los objetos en la pantalla, lo que significa que tenemos que guardar información sobre la profundidad, para saber qué puntos están más cerca de otros y así poder superponerlos adecuadamente.

Para ello vamos a utilizar una función lineal en la coordenada z , que lleve el rango $[-f, -n]$ a $[-1, 1]$:

$$z'' = \frac{c_0 \cdot z' + c_1}{-z'} \quad (2.56)$$

donde

$$c_0 = \frac{n + f}{n - f} \quad (2.57)$$

$$c_1 = \frac{2fn}{n - f} \quad (2.58)$$

Incluyendo ahora esta nueva ecuación en la matriz anterior tenemos:

$$\begin{pmatrix} x'' \\ y'' \\ z'' \\ w \end{pmatrix} = \begin{pmatrix} a_0 & 0 & a_1 & 0 \\ 0 & b_0 & b_1 & 0 \\ 0 & 0 & c_0 & c_1 \\ 0 & 0 & -1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}$$

Este sería el cambio de sistema de coordenadas que estábamos buscando. Comprobamos ahora que la matriz de proyección que utilizamos en la aplicación tiene esta forma (la obtenemos con `glGetDoublev`):

$$\begin{pmatrix} 2.74748 & 0 & 0 & 0 \\ 0 & 2.74748 & 0 & 0 \\ 0 & 0 & -3.74748 & -22.5922 \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

La matriz tiene la estructura que hemos definido, siendo:

$$a_0 = 2.74748; a_1 = 0; b_0 = 2.74748; b_1 = 0; c_0 = -3.74748; c_1 = -22.5922;$$

2.2.3. Paso de coordenadas de recorte a coordenadas de pantalla

Por lo tanto ya tenemos las coordenadas de recorte y ahora vamos a calcular las coordenadas de pantalla. Las coordenadas de pantalla se expresan en píxeles y nos dicen cuál va a ser la representación del objeto en la pantalla del ordenador.

Los parámetros de la ventana que vamos a utilizar son:

- x_l, y_b : número de columna y fila que ocupa la esquina inferior izquierda de la ventana.
- w, h : anchura y altura de la ventana
- n_d, f_d : rango de valores de z, siendo n_d la profundidad más cercana al observador y f_d la más lejana.

Podemos verlo en la figura 2.4:

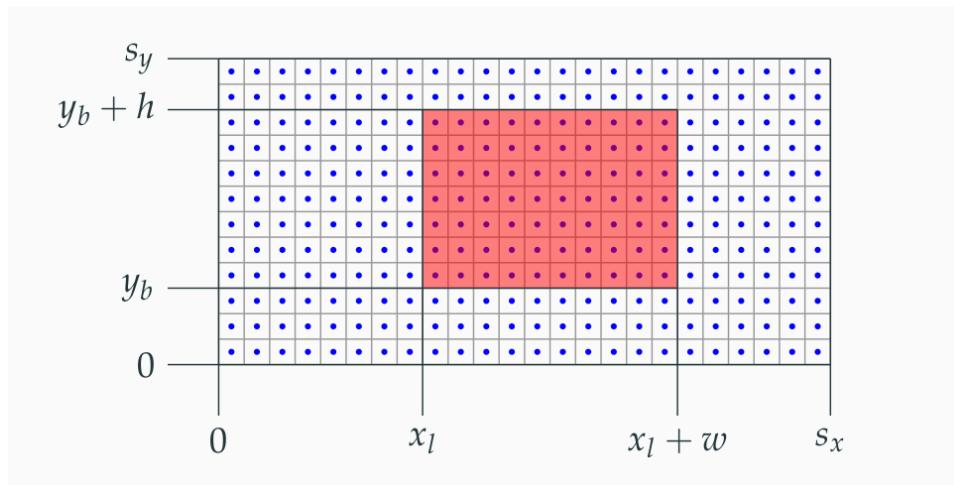


Figura 2.4: Notación Viewport

Las condiciones que tenemos que aplicar para pasar de coordenadas de recorte a coordenadas de pantalla son:

- en el eje de coordenadas x tenemos que llevar el intervalo $[-1, 1]$ al intervalo $[x_l, x_l + w]$.
- en el eje de coordenadas y tenemos que llevar el intervalo $[-1, 1]$ al intervalo $[y_b, y_b + h]$.

- en el eje de coordenadas z tenemos que llevar el intervalo $[-1, 1]$ al intervalo $[n, f]$.

Por tanto las transformaciones que tenemos que aplicar son:

$$x''' = (x'' + 1) \cdot \frac{w}{2} + a \quad (2.59)$$

$$y''' = (y'' + 1) \cdot \frac{h}{2} + b \quad (2.60)$$

$$z''' = (z'' + 1) \cdot \frac{f - n}{2} + n \quad (2.61)$$

Las cuales las podemos representar en la matriz:

$$\begin{pmatrix} x''' \\ y''' \\ z''' \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{w}{2} & 0 & 0 & \frac{w}{2} + a \\ 0 & \frac{h}{2} & 0 & \frac{h}{2} + b \\ 0 & 0 & \frac{f-n}{2} & \frac{f+n}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x'' \\ y'' \\ z'' \\ 1 \end{pmatrix} \quad (2.62)$$

En la que podemos ver que estamos aplicando un escalado $[\frac{w}{2}, \frac{h}{2}, \frac{f-n}{2}]$ y una traslación $[\frac{w}{2} + a, \frac{h}{2} + b, \frac{f+n}{2}]$. OpenGL toma $f = 1$ y $n = 0$ y guarda los valores de (a, b, w, h) en el vector viewport.

2.2.4. Paso de coordenadas de pantalla a coordenadas finales

Por último tenemos que pasar de coordenadas de pantalla a las coordenadas finales que son las que utilizaremos para dibujar las trayectorias.

Para ello vamos a realizar dos transformaciones afines:

- Queremos que nuestro "punto de referencia" esté en el origen del sistema de coordenadas. Al haber hecho una proyección para pasar de sistemas de coordenadas locales a sistemas de coordenadas de pantalla sabemos que un punto en las coordenadas de pantalla corresponde a una recta en las coordenadas de dispositivo. En concreto el punto de referencia que estamos buscando es la recta que pasa por el punto $(0, 0)$ en las coordenadas de dispositivo. Por tanto, utilizando solo un punto de esa recta (el $(0, 0)$) podemos saber cómo se representa en la pantalla. Aplicamos las transformaciones que hemos estudiado antes (modelo-vista-proyección-viewport) al punto $(0, 0)$ y nos da $(250, 250)$. Por lo tanto la traslación que estamos buscando es la que lleva el punto $(250, 250)$ al punto $(0, 0)$, es decir:

$$x^{iv} = x''' - 250 \quad (2.63)$$

$$y^{iv} = y''' - 250 \quad (2.64)$$

- Por otro lado queremos que en nuestro sistema de referencia los "targets" estén situados en la circunferencia de radio 2, por lo tanto tenemos que aplicar un escalado. Para saber el escalado que tenemos que hacer aplicamos la transformación modelo-vista-proyección-viewport al "target" que en el sistema de referencia final está situado en $(2 \cdot \cos \frac{\pi}{4}, 2 \cdot \sin \frac{\pi}{4})$. Después calculamos la distancia del punto que nos ha dado al punto origen, y dividimos la distancia de nuestro punto origen a nuestro punto objetivo entre esa distancia. El resultado es 0.00945962, que será el factor de escala.

Por último vamos a representar estas transformaciones afines en forma de matriz:

$$\begin{pmatrix} x^{iv} \\ y^{iv} \\ 1 \end{pmatrix} = \begin{pmatrix} 0.0009 & 0 & -250 \\ 0 & 0.0009 & -250 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x''' \\ y''' \\ 1 \end{pmatrix} \quad (2.65)$$

Y de esta forma ya tenemos todas las transformaciones que vamos a hacer para obtener las coordenadas con las que haremos el análisis de los datos obtenidos.

Capítulo 3

Desarrollo del proyecto

En este capítulo vamos a hablar de cómo se han desarrollado las diferentes fases del proyecto así como de los resultados obtenidos.

3.1. Planificación del experimento

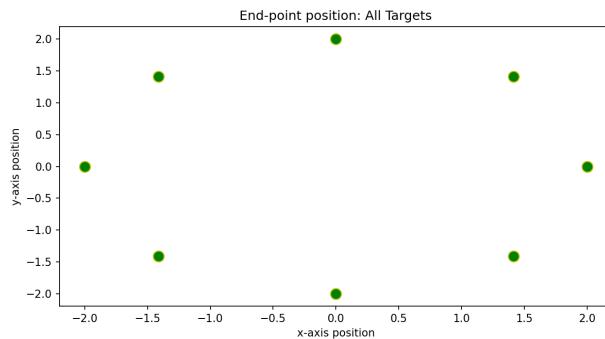
En esta fase del proyecto decidimos cuál iba a ser la forma que iba a tener el experimento que posteriormente íbamos a realizar.

Desde el primer momento sabíamos que el objetivo era tener una aplicación en la que aparecieran una serie de puntos en la pantalla, y en la que el usuario tuviera que mover el cursor de un punto a otro haciendo movimientos balísticos. Posteriormente queríamos ser capaces de guardar las trayectorias de los movimientos realizados para poder analizarlas después.

3.1.1. Diseño de nuestro experimento

Dado que el objetivo del proyecto era probar el funcionamiento de System3D Touch en este tipo de experimentos, y no disponiendo de otros dispositivos complementarios, decidimos que el experimento consistiera en la aparición en pantalla de una serie de puntos sobre los que el sujeto debería situar el cursor.

La dinámica del experimento es la siguiente: primero tenemos un punto de referencia en el centro, y cuando el sujeto se haya posado sobre él, desaparece este y aparece otro sobre una circunferencia alrededor del punto de referencia, en una de las siguientes posiciones:



Al haber situado el cursor sobre el punto objetivo, o al haber terminado el tiempo límite para realizar el movimiento, el punto objetivo desaparece y aparece otra vez el punto de referencia, volviendo a empezar otra iteración. Las trayectorias que nos interesan son las del punto de referencia al punto objetivo.

Tenemos además cinco fases, en las que cambian las condiciones en como se comporta System3D Touch:

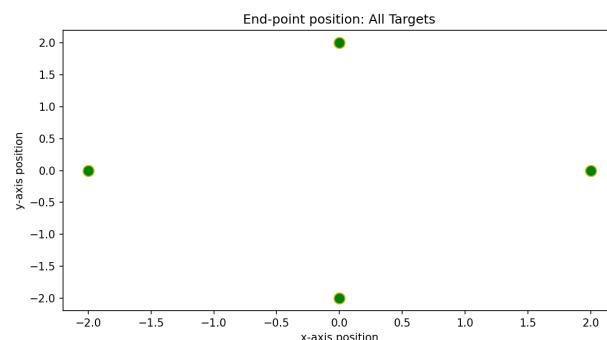
- **Fase de adaptación:** en esta fase solo se muestra el cursor sin ningún punto objetivo. El sujeto es libre para hacer cualquier tipo de movimiento. Esta fase sirve para que el sujeto se adapte a la utilización del dispositivo, así como para que tenga en cuenta las condiciones en las que el dispositivo no funciona correctamente, como cuando se acerca o se aleja demasiado en el eje *z*.
- **Fase inicial**, sin ninguna fuerza y mostrando el cursor.
- **Segunda fase**, en la que se muestra el cursor y se aplica una fuerza constante en la dirección *x*, en sentido negativo.
- **Tercera fase**, en la que no se aplica ninguna fuerza y se oculta el cursor en los movimientos desde el punto de referencia al punto objetivo. Al terminar el movimiento se vuelve a mostrar el cursor para volver al punto de referencia.
- **Cuarta fase**, en la que se aplica la misma fuerza de la segunda fase y se oculta el cursor siguiendo lo explicado en la tercera fase.

Los parámetros que tuvimos que elegir para desarrollar la aplicación fueron:

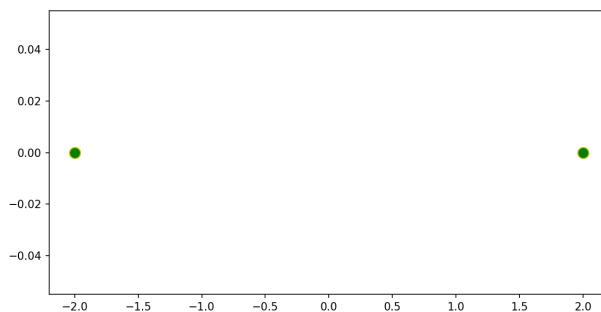
- **Número de repeticiones por fase.** Dado que la realización del experimento requiere de movimientos del brazo durante un periodo de tiempo algo extenso, teníamos que encontrar un acuerdo entre tener

un número suficiente de repeticiones por punto objetivo y que el experimento no fuese demasiado exigente para los individuos. Es por eso por lo que en la fase de prueba probamos diferentes configuraciones con sujetos de diferentes edades, desde 20 a 90 años. Finalmente elegimos tener 80 repeticiones por fase, para así tener 10 repeticiones por punto. Habría que tener en cuenta que en el caso de finalmente realizar el experimento con personas mayores de 80 años, sería recomendable bajar el número de repeticiones.

- **Tiempo para la realización del movimiento.** Este es un parámetro clave a la hora de diseñar el experimento, pues queríamos que los sujetos estuviesen obligados a hacer movimientos balísticos. Por lo tanto teníamos que encontrar un balance entre tener tiempo suficiente para que pudiesen alcanzar el punto y la obligación de que esos movimientos fuesen rápidos. Aquí también encontramos grandes diferencias en los resultados de personas jóvenes y más mayores, pero finalmente decidimos utilizar un tiempo de 2 segundos, con el que, una vez pasado un periodo de aprendizaje, todos los sujetos fueron capaces de alcanzar en ese tiempo el punto objetivo.
- **Número y disposición de los puntos objetivos.** Hicimos pruebas con diferentes sujetos con 8 puntos (como hemos mencionado arriba), 4 puntos en las siguientes posiciones:



y 2 puntos:



Pensamos que el experimento más interesante era con 8 puntos, pues así podíamos estudiar las diferencias en el aprendizaje de movimientos cuando hay que mover el cursor en un solo eje a cuando hay que moverlo en dos ejes a la vez. En todo caso, y en relación con la dificultad de realizar un número grande de iteraciones en personas mayores, podría estudiarse el caso de reducir el número de puntos para así poder reducir a su vez el número de iteraciones.

- Disposición de los **puntos objetivos siguiendo un orden o de forma aleatoria**.

Otra decisión que tomamos fue la del criterio para considerar que un movimiento es correcto. En primer lugar pensamos que el criterio fuera que el sujeto pasase el cursor por encima del punto objetivo, de forma similar a como hacen en "*Cerebellar Contributions to Reach Adaptation and Learning Sensory Consequences of Action*"[8]. Después pensamos que sería interesante estudiar también, no solo que los sujetos realizaran el movimiento en la dirección correcta, sino con la distancia correcta. Por eso, para que el movimiento se considere correcto, el sujeto debe situar durante un momento el cursor encima del punto. Hicimos lo mismo a la hora de situar el cursor dentro del punto de referencia, e iniciar así el movimiento. Esto fue debido a que, en una pequeña demo inicial vimos que, si lo hacíamos solo con pasar el cursor sobre el punto de referencia, todos los movimientos empezaban con un desplazamiento en sentido contrario, el correspondiente al movimiento de volver al punto de referencia. De esta forma el sujeto está obligado a pararse dentro del punto inicial e iniciar el movimiento desde ahí.

También introdujimos un sonido cuando el movimiento no se ha realizado de forma correcta. Esto sirve de feedback al usuario, para saber cuándo el movimiento se ha realizado correctamente y cuándo no.

3.2. Desarrollo de la aplicación

El objetivo de esta fase es desarrollar una aplicación sobre la que llevar a cabo el experimento diseñado en la fase de planificación, y además, desarrollar un script para analizar los resultados que tengamos al realizar el experimento. Es decir, queremos tener como resultado:

- Una aplicación diseñada para ejecutarse desde el ordenador, pues necesitamos utilizar a su vez el dispositivo Touch.

Dicha aplicación debe tener:

- Un menú de selección para poder elegir entre los 5 modos de uso: libre, sin fuerza, con fuerza, sin fuerza y sin cursor y con fuerza y sin cursor.

- Al elegir el modo (salvo eligiendo el modo libre) debe aparecer una tarea que tendrá que realizarse un número determinado de iteraciones y que consistirá en:

1. Aparece el target de referencia, en la posición 0 (el centro).

2. Al parar el cursor sobre el target de referencia este desaparece y aparece en su lugar un target situado sobre la circunferencia de radio 2 y centro el target de referencia. El nuevo target puede aparecer en cualquiera de las 8 posiciones mencionadas antes.

3. Al parar el cursor sobre el target objetivo dentro del tiempo estipulado, o al término de dicho tiempo, este desaparece y vuelve a aparecer el target de referencia.

- Se deben poder guardar las trayectorias y los tiempos del movimiento desde el target de referencia al target objetivo. Cada una de las 4 fases se guardará en un fichero csv diferente.

- Un script de python que lea los ficheros csv generados al realizar el experimento y genere gráficas con los resultados en cada una de las fases de:

- Errores (distancia al target objetivo) en cada iteración. Por un lado queremos tener una gráfica con todos los errores de la fase, y por otro separados según el target objetivo.

- Tiempo transcurrido al realizar el movimiento. Al igual que con los errores, por fase y por target objetivo dentro de la fase.

- Distancia y velocidad al punto objetivo a través del tiempo, en cada uno de los ejes. Separado por targets objetivos.

La fase del desarrollo de la aplicación tuvo tres partes: instalación del software necesario, implementación del código y revisión del código una vez elegidos los parámetros.

3.2.1. Instalación del software de OpenHaptics

La primera decisión que tuvimos que tomar fue si instalar el software de OpenHaptics en una máquina Ubuntu, o en una máquina Windows. Aunque en un principio la opción inicial fue instalarlo en Ubuntu 20.04, después de una prueba nos dimos cuenta de que Windows soportaba mejor la interfaz gráfica necesaria para llevarlo a cabo. Por tanto procedimos a realizar la instalación en Windows.

Los paquetes utilizados pueden encontrarse dentro de la página web de 3D Systems [14]. Tenemos que instalar las librerías de OpenHaptics y los drivers de Touch, que podemos encontrar en la misma página.

El toolkit de OpenHaptics tiene ciertos requerimientos hardware:

- Procesador:
 - Intel i5 / i7, 5^a Generación o mayor.
 - CPU mínimo 2.5 GHz de frecuencia.
 - RAM 4 GB.
- Tarjeta gráfica de mínimo 256 MB VRAM .
- Espacio de disco 512 MB.
- Resolución de pantalla 1280 x 800 (mínimo).

Y software:

- Sistema Operativo 64-bit Windows 7, 8.1, y 10.

Tenemos tres documentos que nos servirán de ayuda para instalar y desarrollar la aplicación:

- **OpenHaptics Installation Guide:** guía de instalación del toolkit de OpenHaptics.
- **Programmer's Guide:** guía que explica la arquitectura de OpenHaptics, cómo funciona y lo que se puede hacer. Hay ejemplos de código, una explicación de cómo configurar Visual Studio para trabajar con la librería y una visión general de las tres APIs: QuickHaptics, HLAPI y HD API.
- **API Reference Guide:** contiene documentación sobre las funciones de QuickHaptics, HLAPI y HD API.

Las librerías de OpenHaptics están en c++, por lo tanto será el lenguaje de programación que utilizaremos. Las tres APIs que incluye son, como hemos comentado antes:

- **QuickHaptics:** permite escribir rápido y fácilmente nuevas aplicaciones hápticas, o añadir funcionalidades hápticas a aplicaciones ya existentes.
- **HDAPI:** provee un acceso a bajo nivel al dispositivo háptico, lo que permite a los programadores controlar las fuerzas que se pueden aplicar.
- **HLAPI:** provee un acceso a alto nivel, similar a la API de OpenGL. Permite la reutilización de código de OpenGL.

En la figura 3.1 podemos ver cómo se relacionan las tres APIs:

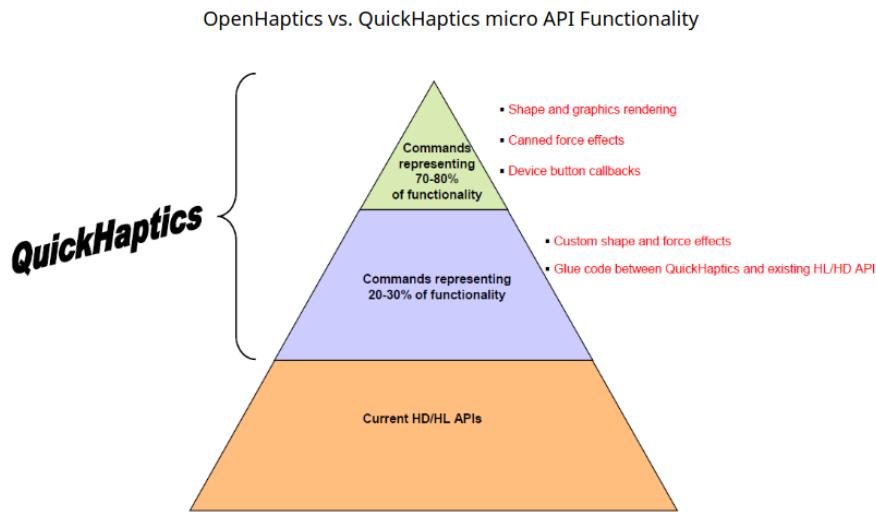


Figura 3.1: OpenHaptics APIs

En nuestro caso utilizaremos HDAPI, pues necesitamos controlar la fuerza que le aplicamos al dispositivo. También utilizaremos la biblioteca GLUT [11], una biblioteca de utilidades de OpenGL que nos permite manejar ventanas y representaciones gráficas.

3.2.2. Implementación del código

La estructura de un programa que utilice OpenHaptics la podemos ver en la figura 3.2:

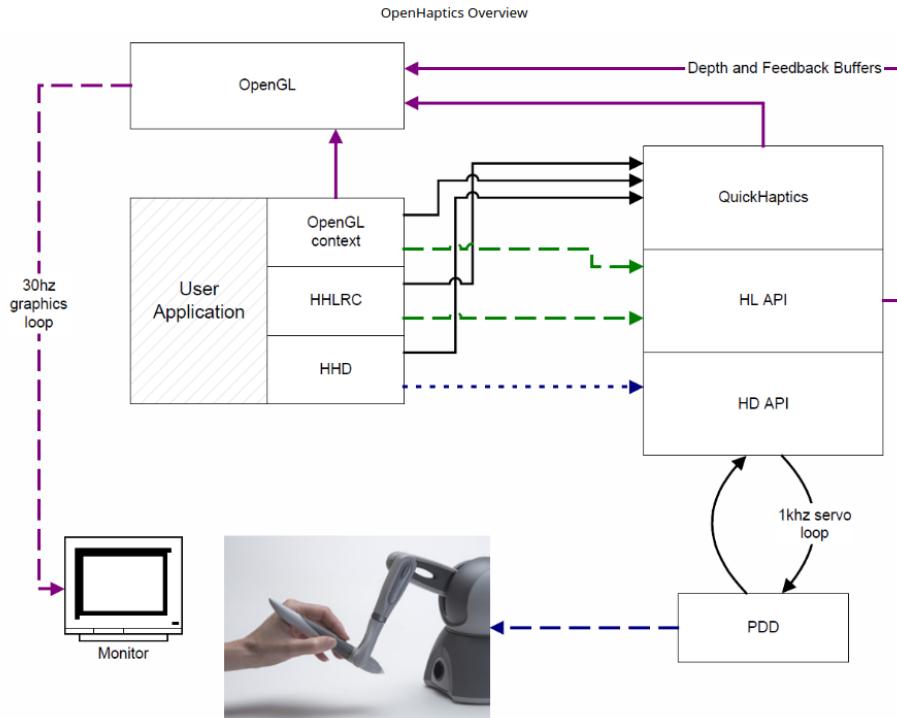


Figura 3.2: Estructura de un programa de OpenHaptics

Como podemos ver tenemos dos bucles: el **bucle gráfico** y el **servoloop**.

El servoloop se usa para controlar las fuerzas que se mandan al dispositivo haptico. Para mantener un feedback consistente este bucle debe ejecutarse al menos a una frecuencia de 1000Hz, por lo que se ejecuta en una hebra diferente, con prioridad máxima. Para inicializar el servoloop utilizamos la función de HDAPI `hdStartScheduler()`. Se debe llamar una vez que se han inicializado las rutinas del dispositivo y se han añadido las callbacks asíncronas del scheduler. Una vez inicializado el servoloop empiezan a ejecutarse las callbacks que hemos definido, en cada iteración del bucle.

El bucle gráfico se ejecuta en una hebra diferente, y dado que el ojo humano no es capaz de procesar tantas imágenes por segundo, la frecuencia del bucle será menor, en concreto de 30Hz. El bucle gráfico se encarga de actualizar la representación gráfica, según los eventos que ocurran. Para inicializar el bucle gráfico utilizamos la función de GLUT `glutMainLoop()`. Una vez inicializado el bucle gráfico se llamarán en cada iteración a las callbacks que hayamos definido.

Dado que la frecuencia a la que se ejecutan los dos bucles es diferente, podemos tener problemas a la hora de que la hebra gráfica pida información a la hebra haptica sobre el estado del dispositivo. Para que el estado del

dispositivo sea consistente se utiliza el scheduler, que provee sincronización entre las dos hebras.

Las callbacks del servoloop pueden ser **síncronas** o **asíncronas**. Las callbacks síncronas se usan principalmente para tener una instantánea del estado del dispositivo. Las callbacks síncronas solo devuelven el control de la hebra una vez terminadas, por lo que la aplicación tiene que esperar a que la función se haya terminado. Las callbacks asíncronas, por el contrario, devuelven el control justo cuando son programadas, por lo que la aplicación puede seguir ejecutándose. Son adecuadas para representar un efecto haptico, pues pueden perdurar en el tiempo, ya que en cada iteración se aplicará el efecto de la callback al dispositivo.

Las callbacks pueden devolver dos tipos de valores: *DONE* y *CONTINUE*. Las callbacks con return code *DONE* no volverán a ser ejecutadas en el bucle del servoloop hasta que vuelvan a ser programadas. Por el contrario, las callbacks con return code *CONTINUE* se vuelven a programar para la próxima iteración del bucle, hasta que explícitamente se especifique que se desprograme.

En nuestro caso las callbacks del servoloop son:

- *beginUpdateCallback*: callback asíncrona, con return code *CONTINUE*. Se programa al inicializar el dispositivo háptico y es la callback principal de la hebra haptica. Actualiza el estado del dispositivo háptico y escribe en el fichero csv las coordenadas del dispositivo (en caso de que se haya empezado el movimiento de ir desde el target de referencia al target objetivo, a lo que nos referiremos con que se ha inicializado la acción).

Dependiendo del caso en el que estemos:

- Si la acción se ha inicializado comprueba el tiempo que ha transcurrido desde que se empezó la acción: si ha superado el tiempo límite programa la callback *actionFinished*.
 - Si el cursor está en contacto con el punto de referencia, no se ha inicializado la acción y no estamos en la fase libre del experimento: comprueba si el cursor ha estado parado el tiempo necesario sobre el punto de referencia, si es así programa la callback *actionInitialized*.
 - Si el cursor está en contacto con el target objetivo en esa iteración, la acción se ha inicializado y no estamos en la fase libre del experimento: si el cursor ha estado parado el tiempo necesario sobre el target objetivo se programa la callback *actionFinished*.
- *setDeviceTransformationCallback*: callback síncrona, con return code *DONE*. Se programa al actualizar las transformaciones de coordenadas y calcula las operaciones que hay que realizar.

- *forceCallback*: callback asíncrona, con return code *CONTINUE*. Se programa en la callback *actionInitialized* y aplica la fuerza que hemos configurado al dispositivo.
- *clearForceCallback*: callback asíncrona, con return code *CONTINUE*. Se programa en la callback *actionFinished* y elimina la fuerza que se había aplicado antes, poniendo el vector fuerza nuevamente a 0.
- *actionInitialized*: callback asíncrona, con return code *DONE*. Se programa en la callback *beginUpdateCallback* y sus funciones son:
 - Oculta el target de referencia.
 - Calcula el siguiente target objetivo y lo hace visible.
 - Si estamos en algunas de las fases en las que se aplica fuerza al dispositivo, programa la callback *forceCallback*.
 - Inicializa el tiempo de inicio de acción.
 - Si estamos en alguna de las fases en las que se oculta el cursor, hace el cursor no visible.
- *actionFinished*: callback asíncrona con return code *DONE*. Se programa en la callback *beginUpdateCallback* y sus funciones son:
 - Oculta el target objetivo.
 - Muestra el target de referencia.
 - Programa la callback *clearForceCallback*.
 - Muestra el cursor.
 - Si se han terminado las iteraciones diseñadas para la fase, selecciona la fase libre del experimento.

Y las del bucle gráfico:

- *glutDisplayFunc*: callback que se utiliza para actualizar la vista. En esta callback actualizamos el estado del dispositivo, capturando el último estado del servoloop. También dibujamos la escena, según el modo del menú que se haya elegido.
- *glutReshapeFunc*: callback para actualizar las dimensiones de la ventana. También se actualiza la posición de la cámara.
- *glutIdleFunc*: callback para mandar una petición de redibujar la ventana.

Vamos a hablar ahora del main y de las dos clases más importantes de la aplicación: HapticDeviceManager y PointManager.

Main

Las funciones del main son:

- Inicializar la librería de GLUT: lo hacemos con la función *glutInit()*.
- Crear y definir las dimensiones de la pantalla y el modo de visualización: *glutCreateWindow()*, *glutInitDisplayMode()* y *glutInitWindowSize()*.
- Definir las callbacks que actualizarán el entorno gráfico: *glutDisplayFunc*, *glutReshapeFunc* y *glutIdleFunc*.
- Crear el menú: *glutAttachMenu()* y *glutAddMenuEntry()*.
- Inicializar la escena que vamos a representar: inicializamos OpenGL, creamos un objeto de la clase PointManager y otro objeto de la clase HapticDeviceManager, al que le pasamos el objeto PointManager como parámetro.
- Inicializar el bucle gráfico: *glutMainLoop()*.

Clase HapticDeviceManager

Esta clase nos permite integrar las interacciones hapticas con la lógica y el estado de la aplicación.

Las funciones son llamadas desde el objeto HapticDeviceManager que hemos creado en el main. Las más importantes son:

- *setUp*: inicializamos el dispositivo háptico. y programamos la callback *beginUpdateCallback*. Inicializamos el scheduler para que empiece el bucle del servoloop. También inicializamos el atributo PointManager con el parámetro que hemos pasado a la función.
- *updateState*: se llama en cada iteración del ciclo gráfico. Sincroniza el estado con el thread háptico.
- *updateWorkspace*: calcula las transformaciones necesarias para pasar de coordenadas del dispositivo a coordenadas del mundo. También programa la callback *setDeviceTransformCallback*.
- *drawCursor*: utiliza las funciones de OpenGL para dibujar el cursor en la pantalla.
- *setManipulationStyle*: configura la fase del experimento (libre, sin fuerza, con fuerza, sin fuerza sin cursor, con fuerza sin cursor) según la opción seleccionada en el menú.

Clase PointManager

Esta clase nos permite manejar los puntos que aparecen en la escena, desde el números de puntos que hay, dónde están situados y las características que tienen. Cada punto tendrá un estado asociado: HighLighted y Selected. El punto tendrá estado HighLighted cuando aparezca en pantalla, y Selected cuando esté oculto. Las funciones más importantes de la clase son:

- *setUp*: aquí definimos el número de puntos que hay en la escena, dónde están situados y su estado. En nuestro caso definimos 9 puntos: el punto de referencia con estado HighLighted, y los demás, es decir, los puntos targets, con estado Selected.
- *updatePointSize*: Determina el factor de escala para transformar las coordenadas a coordenadas de pantalla y dibujar los puntos en dimensiones de píxeles.
- *drawPoints*: Dibuja los puntos, dependiendo del estado que tengan y teniendo en cuenta el factor de escala calculado en *updatePointSize*.
- *getPointPosition/setPointPosition*: devuelve o configura la posición de punto según el índice pasado, en coordenadas del mundo.
- *setPointHighlighted/Selected*: cambia el estado del punto a HighLighted o Selected, según la función.
- *isPointHighlighted/Selected*: devuelve un boolean indicando si el punto tiene estado HighLighted o Selected.
- *getNumPoints*: devuelve el número de puntos que hemos configurado. Los puntos se guardan en un array.

En la figura 3.3 podemos ver representada la estructura que hemos explicado.

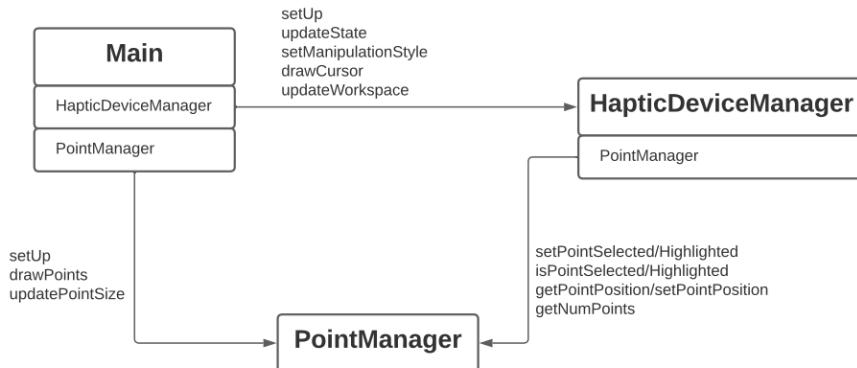


Figura 3.3: Diagrama de clases

Guardado de datos

Para guardar los datos obtenidos de las trayectorias punto_inicial-punto_objetivo utilizamos excel. Guardamos los datos en cuatro ficheros diferentes, uno por cada fase del experimento. Cada fichero contiene cinco columnas:

- Coordenada x del cursor
- Coordenada y del cursor
- Número de iteración. Una iteración es un movimiento desde el punto inicial al objetivo.
- Punto objetivo. Numerados del 1 al 8.
- Tiempo transcurrido desde que se empieza el movimiento, en nanosegundos.

Al terminar el experimento analizaremos las gráficas obtenidas con un script de python.

3.2.3. Script de python

El script de python toma los datos de los ficheros excel generados en el desarrollo de experimento, utilizando el módulo pandas.

Después, para cada individuo y cada fase representa las siguientes gráficas:

- **Gráfica con las trayectorias de todas las iteraciones.** A medida que se realizan más trayectorias en un target determinado las líneas son más oscuras. También se representan los targets finales.

- **Gráfica con los puntos finales de los movimientos.** A medida que se realizan más trayectorias en un target determinado las líneas son más oscuras. También se representan los targets finales.
- **Gráfica con la evolución de los errores** respecto al número de iteraciones realizado, agrupados por target. Cada target tiene asociado un color.
- **Gráfica con la evolución del tiempo** necesario para realizar el movimiento respecto al número de iteraciones realizado, agrupados por target. Cada target tiene asociado un color.

La distancia que utilizamos para calcular los errores es la distancia euclídea.

El código de la aplicación lo podemos encontrar en mi repositorio de GitHub[9].

3.3. Obtención de resultados

Como hemos mencionado en la sección 1.3 *Fases del proyecto*, la realización de los experimentos fue llevada a cabo en dos fases: una primera fase de prueba, para decidir los parámetros que íbamos a utilizar; y una segunda fase final para poder obtener resultados, que analizaremos en la sección 3.4 *Análisis de resultados*.

Uno de los requisitos fue tener, en cada una de las fases, representación de individuos de distintas edades, para no tener sesgo a la hora de implementar la aplicación y poder comparar los resultados de los individuos según la edad. Teniendo en cuenta esto y la disponibilidad de personas que teníamos a nuestro alrededor, hicimos el experimento con 13 sujetos, como podemos ver en la tabla 3.1:

Fase 1		Fase 2	
Sexo	Edad	Sexo	Edad
Mujer	23	Mujer	20
Mujer	24	Mujer	22
Hombre	53	Hombre	50
Mujer	52	Hombre	60
Hombre	62	Mujer	59
Mujer	60		
Hombre	90		
Mujer	90		

Cuadro 3.1: Distribución de sujetos por fase

En la segunda fase la muestra de individuos fue similar a la primera, salvo porque no tuvimos individuos mayores de 80 años. Esto fue debido en parte a la poca disponibilidad que teníamos de individuos de edad más avanzada, y en parte a que decidimos realizar el experimento en unas condiciones que les eran menos favorables, con demasiadas repeticiones. Ninguno de los sujetos del experimento tenía un problema cerebral reconocido, por lo que podemos suponer que eran individuos sanos. No pudimos probar el experimento en individuos con problemas en el cerebelo, como hubiese sido lo ideal, por falta de disponibilidad.

En la segunda fase, para poder obtener conclusiones de cómo afecta el orden de aparición de targets en la realización de los movimientos (si aparecen los targets en orden, o de forma aleatoria) a algunos de los individuos le aparecieron los targets en orden y a otros no. A continuación aparecen los sujetos de la segunda fase con sus respectivas características y el modo de experimento que hicieron (en orden o aleatorio). Esta será la nomenclatura que utilizaremos a partir de ahora.

- Sujeto 1: Mujer, 20 años, en orden
- Sujeto 2: Mujer, 22 años, aleatorio
- Sujeto 3: Mujer, 60 años, aleatorio
- Sujeto 4: Hombre, 50 años, en orden
- Sujeto 5: Hombre, 60 años, en orden

Todos los sujetos realizaron todas las fases del experimento y tuvieron 2 minutos de descanso entre fase.

En las figuras 3.4, 3.5, 3.6 podemos ver algunos ejemplos del funcionamiento de la aplicación.



Figura 3.4: Menú de la aplicación

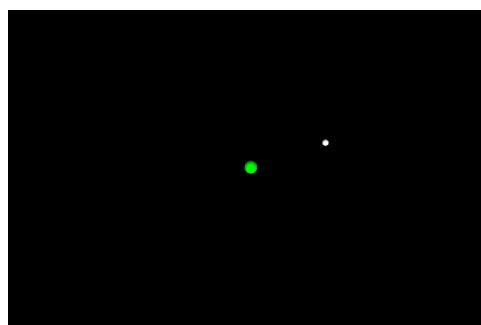


Figura 3.5: Referencia y cursor

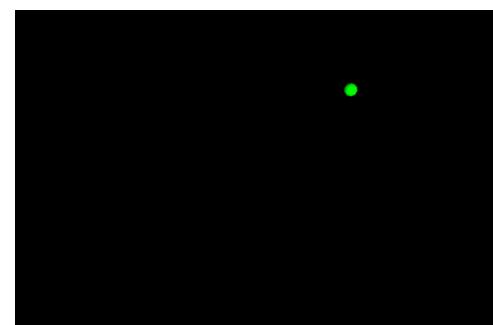


Figura 3.6: Target 2 (T2)

3.4. Análisis de resultados

En esta sección analizaremos los resultados que obtuvieron cada uno de los cinco sujetos. Para ello lo dividiremos en dos secciones: primero compararemos cuáles fueron las diferencias entre los diferentes sujetos en cada una de las fases, y luego estudiaremos las diferencias entre los resultados de cada fase de cada uno de los individuos.

3.4.1. Análisis de la misma fase entre individuos

En este análisis queremos estudiar cuál ha sido el rendimiento de cada sujeto, comparado con el rendimiento de los demás. De esta forma queremos poder tener datos para estudiar cómo influyen factores como la edad y la aparición en orden o no de los targets. Estudiaremos cada una de las fases por separado.

Fase sin fuerza

Si observamos las gráficas con los puntos finales de cada movimiento podemos observar una diferencia entre los sujetos de mayor edad y los más jóvenes. Mientras que en los sujetos más jóvenes (figuras 3.7, 3.8 y 3.10) los puntos se concentran alrededor de los puntos objetivos (hay una gran precisión), en los de mayor edad (figuras 3.9 y 3.11) están más dispersos.

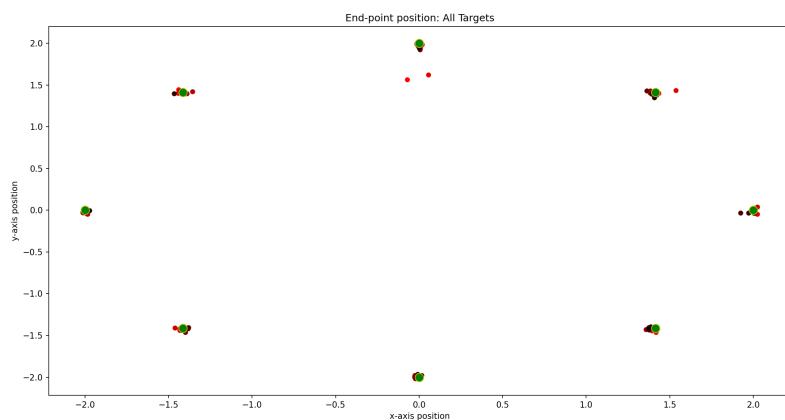


Figura 3.7: Sujeto 1, sin fuerza

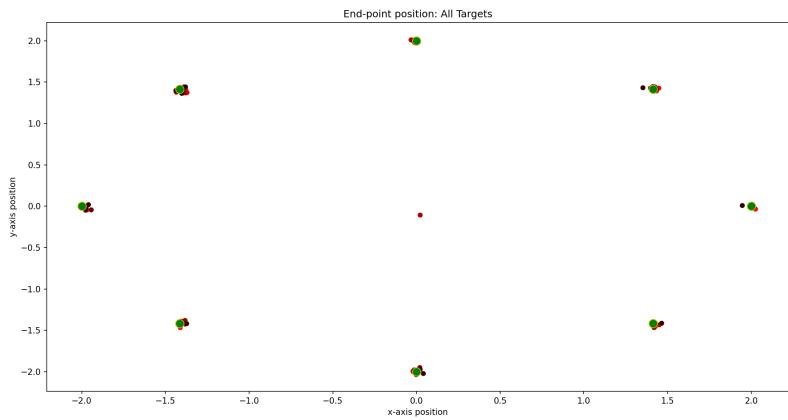


Figura 3.8: Sujeto 2, sin fuerza

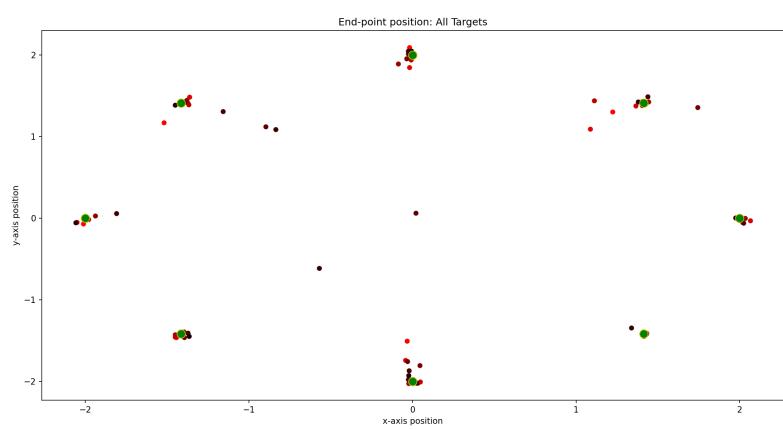


Figura 3.9: Sujeto 3, sin fuerza

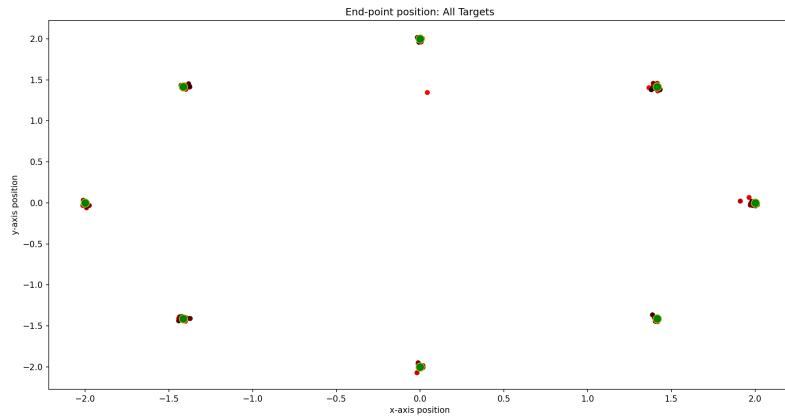


Figura 3.10: Sujeto 4, sin fuerza

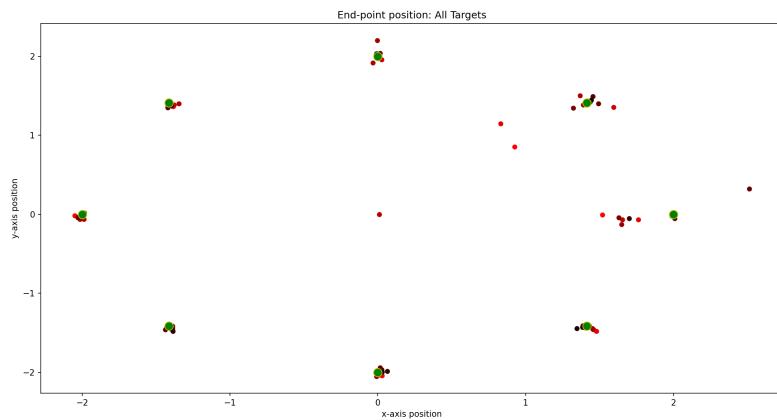


Figura 3.11: Sujeto 5, sin fuerza

También lo podemos ver en las gráficas de la evolución del error por target respecto al número de repeticiones, donde además observamos que el proceso de aprendizaje en los sujetos jóvenes es casi instantáneo (figuras 3.12, 3.13 y 3.15), mientras que en las personas mayores, en el caso de sujeto 5 (targets en orden) se aprecia un aprendizaje en los primeros puntos objetivos (T1 y T2) (figura 3.16) que luego se extrapola a los demás puntos, mientras que en el sujeto 3 (targets en orden aleatorio) ese aprendizaje es más incierto (figura 3.14).

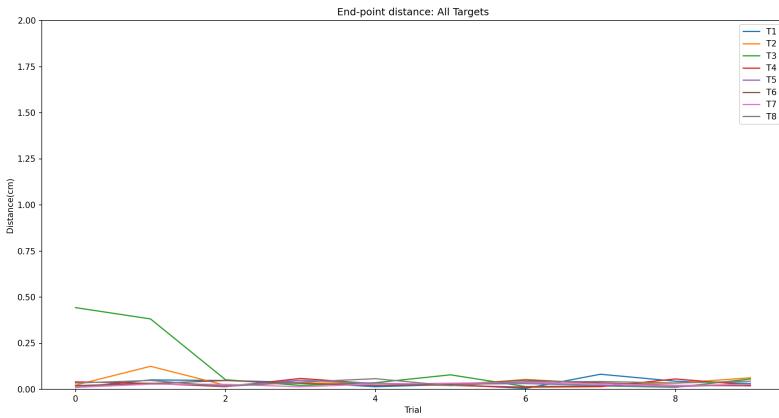


Figura 3.12: Sujeto 1, sin fuerza

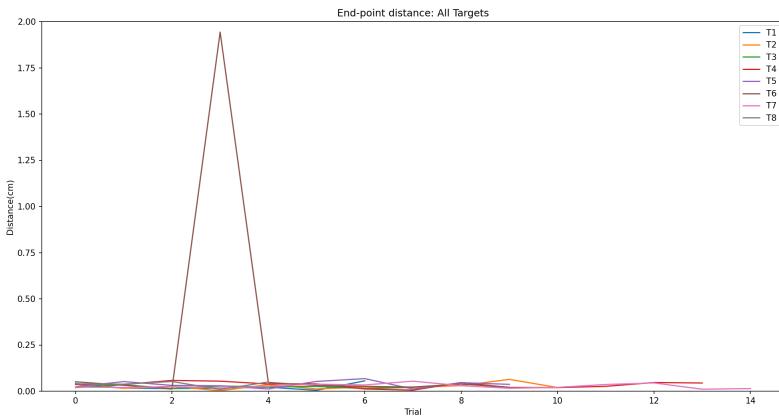


Figura 3.13: Sujeto 2, sin fuerza

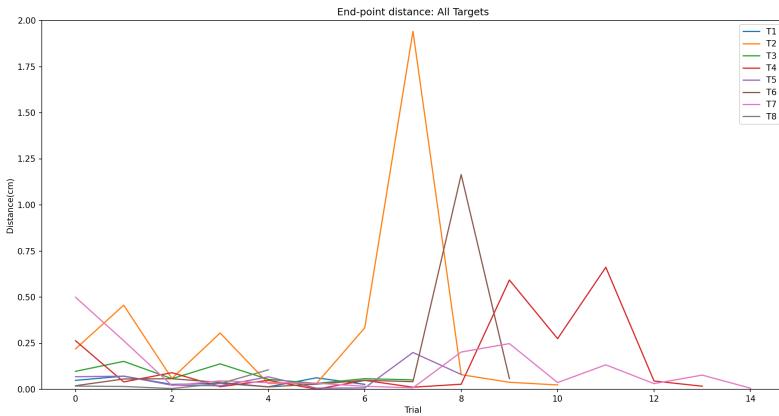


Figura 3.14: Sujeto 3, sin fuerza

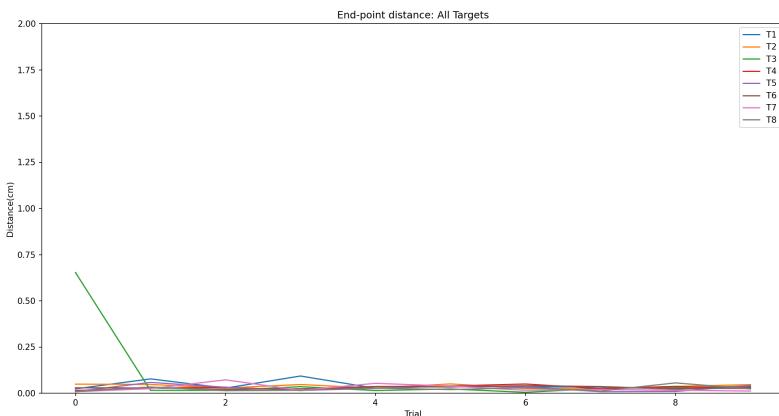


Figura 3.15: Sujeto 4, sin fuerza

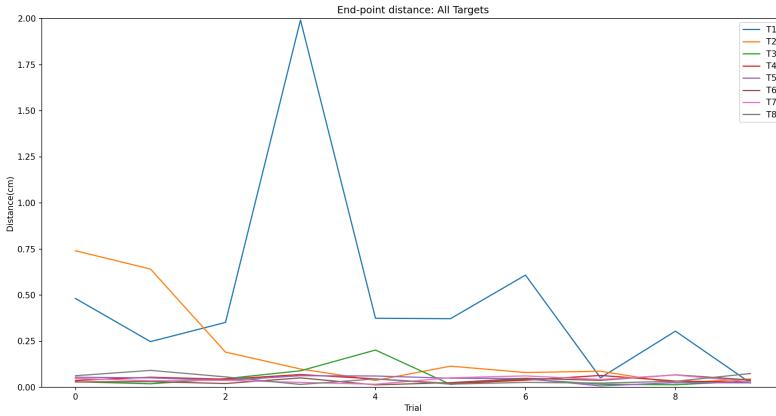


Figura 3.16: Sujeto 5, sin fuerza

Si además estudiamos la rapidez en la ejecución de los movimientos podemos ver que el sujeto que peor lo hace es el 3 -persona mayor, orden aleatorio- (figura 3.19), y el que mejor lo hace el sujeto 4 -50 años, en orden- (figura 3.20). También se puede observar que, aunque los sujetos 3 y 5 (sujetos mayores) tienen una tasa de errores similar, en el sujeto 5 (figura 3.21) podemos reconocer un aprendizaje en la rapidez del movimiento (T1 y T2 empiezan a mejorar en las últimas iteraciones y los demás movimientos comienzan a ser más rápidos, especialmente T5 y T7, donde podemos observar una curva casi completamente descendente). Sin embargo en el sujeto 3 con los puntos en orden aleatorio no lo podemos apreciar, y los tiempos de ejecución son mayores.

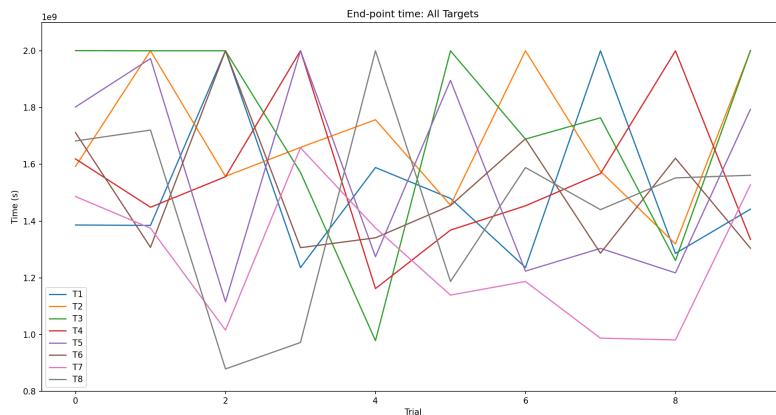


Figura 3.17: Sujeto 1, sin fuerza

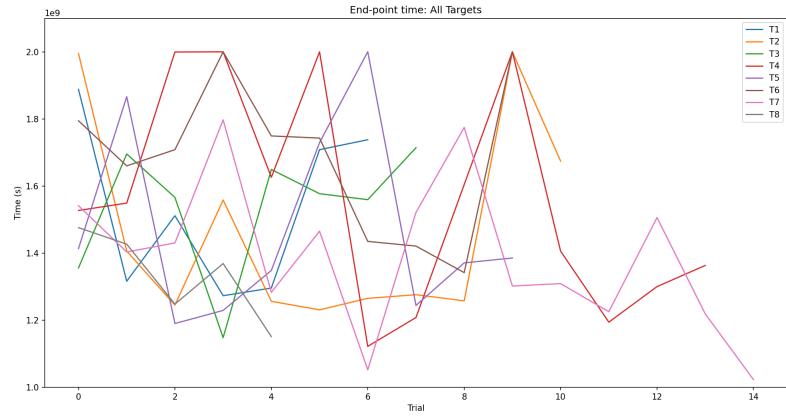


Figura 3.18: Sujeto 2, sin fuerza

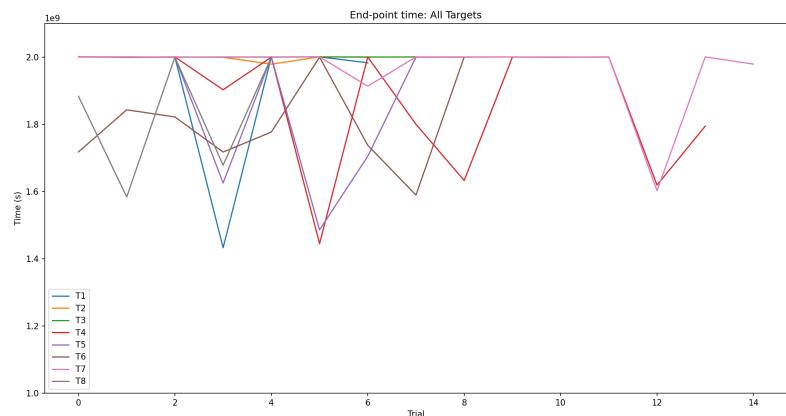


Figura 3.19: Sujeto 3, sin fuerza

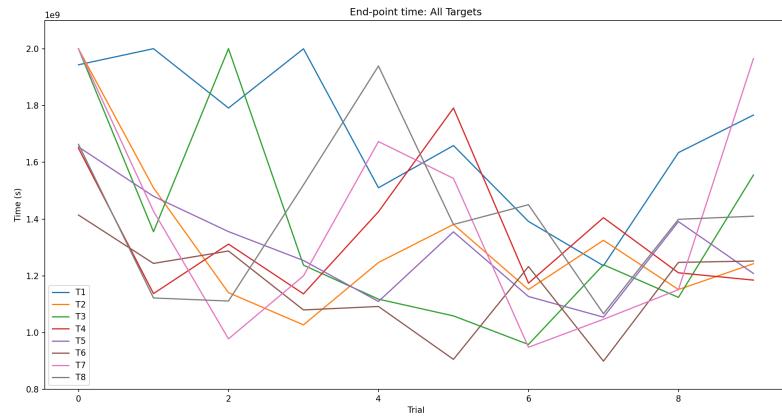


Figura 3.20: Sujeto 4, sin fuerza

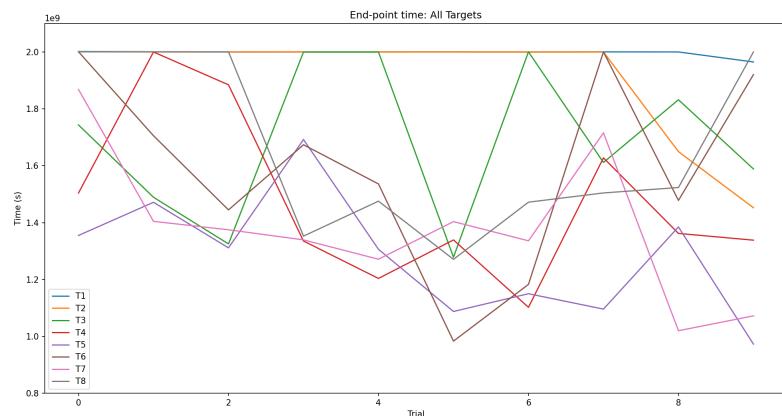


Figura 3.21: Sujeto 5, sin fuerza

Fase con fuerza

Esta fase del experimento trataba de aprender a controlar y corregir la fuerza que aplicábamos al dispositivo Touch. Esta fuerza se mantiene constante en todos los movimientos, y tiene como dirección el eje x , con sentido hacia la izquierda del sujeto.

Estudiando las trayectorias de los movimientos podemos ver que hay diferencia entre los sujetos de mayor y menor edad. Las trayectorias de los sujetos 1,2,4 (figuras 3.22, 3.23 y 3.25) están más definidas, teniendo los primeros movimientos en cada punto trayectorias más largas (las trayectorias rojas más claras) y luego siendo cada vez líneas más rectas (las más oscuras).

Las trayectorias de los sujetos 3 (figura 3.24) y 5 (figura 3.26), sin embargo, son bastante más difusas: en el sujeto 5 podemos observar cómo todas las trayectorias tienen una curva correspondiente al efecto de la fuerza aplicada, mientras que en el sujeto 3 las trayectorias son más rectas, pero se puede ver cómo el cursor se mueve en el sentido de la fuerza al iniciar el movimiento.

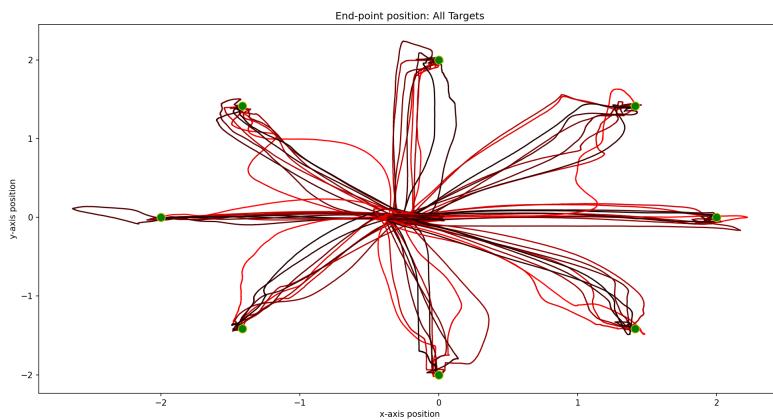


Figura 3.22: Sujeto 1, con fuerza

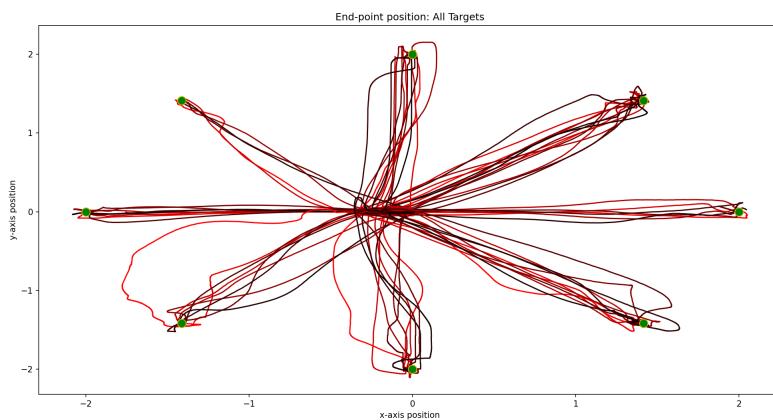


Figura 3.23: Sujeto 2, con fuerza

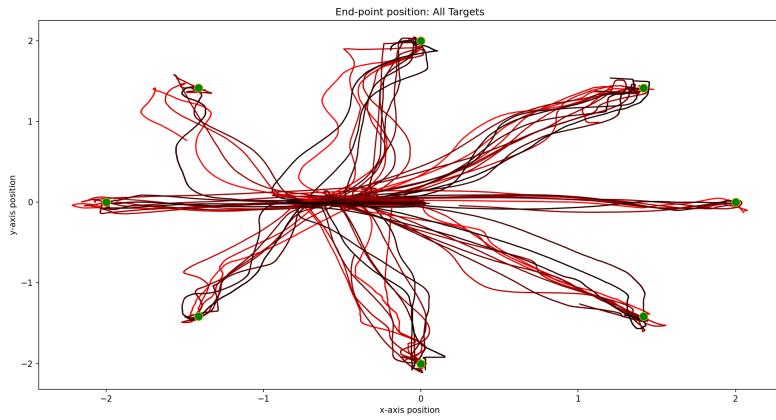


Figura 3.24: Sujeto 3, con fuerza

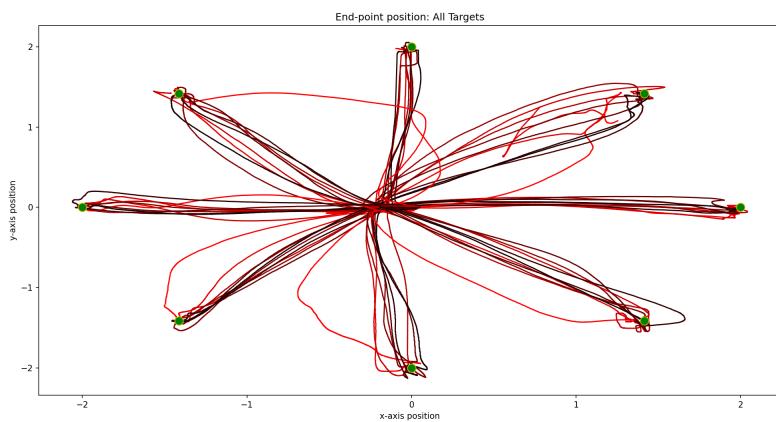


Figura 3.25: Sujeto 4, con fuerza

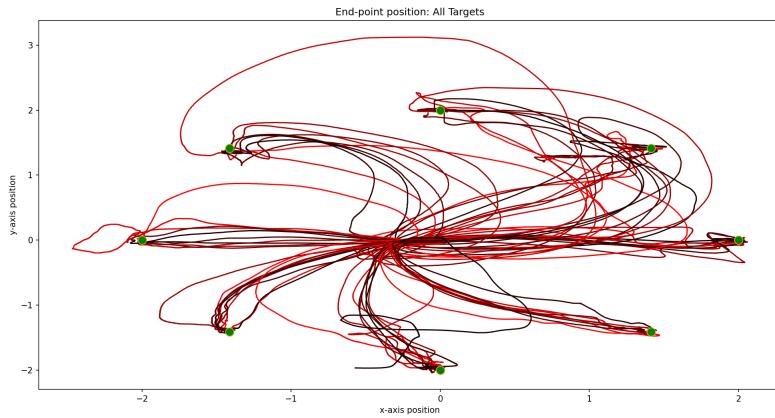


Figura 3.26: Sujeto 5, con fuerza

Estudiando ahora la distribución de puntos finales podemos observar que la precisión del sujeto 5 (figura 3.31) ha sido algo peor que la de los sujetos jóvenes (figuras 3.27, 3.28 y 3.30), cuya ejecución es casi perfecta, pero mejor que la del sujeto 3 (figura 3.29), concentrándose los errores del sujeto 5 en T2 y T7.

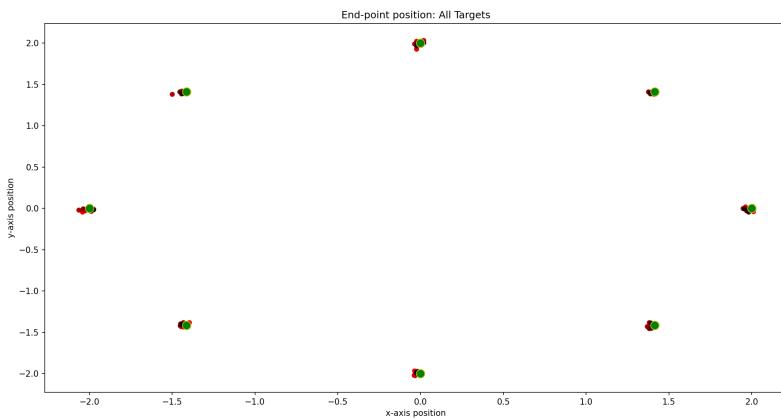


Figura 3.27: Sujeto 1, con fuerza

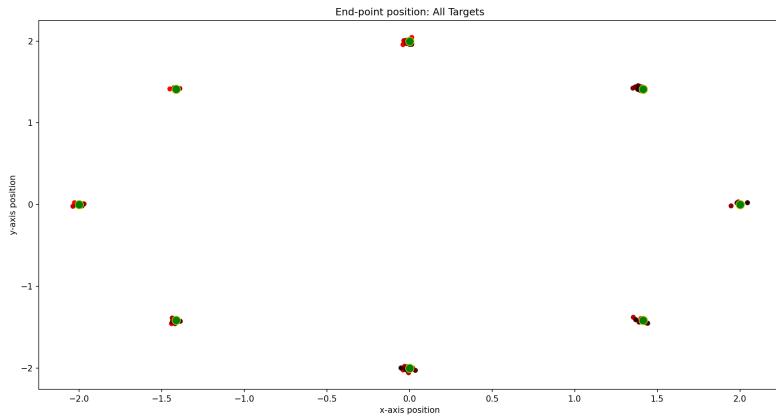


Figura 3.28: Sujeto 2, con fuerza

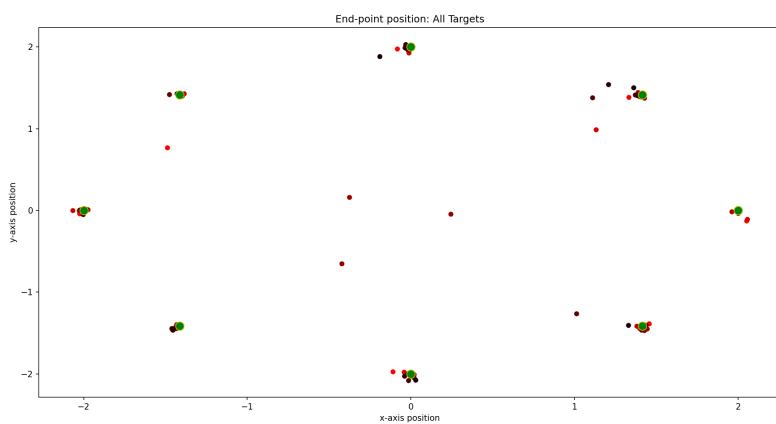


Figura 3.29: Sujeto 3, con fuerza

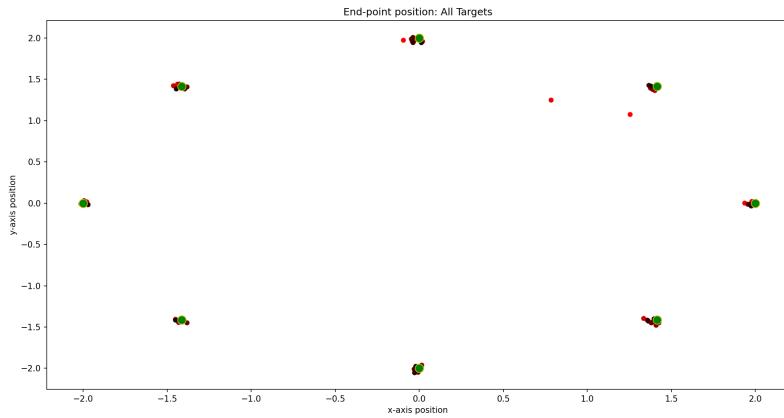


Figura 3.30: Sujeto 4, con fuerza

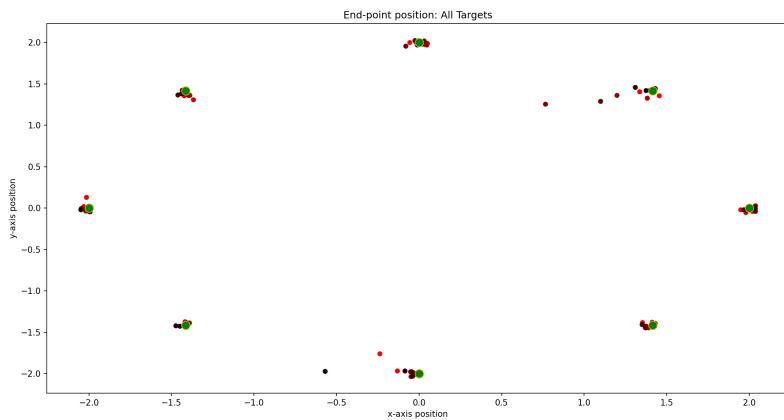


Figura 3.31: Sujeto 5, con fuerza

En cuanto a la evolución de los errores por target respecto al número de iteraciones podemos ver que los sujetos 1,2 y 4 (figuras 3.32, 3.33 y 3.34) han extrapolado el aprendizaje del movimiento sin fuerza al movimiento con fuerza, y el error es cero o muy cercano a cero desde el primer momento. El sujeto 5 (figura 3.37), a pesar de realizar trayectorias muy largas, como hemos comentado antes, también presenta una precisión bastante acertada, con unos errores, salvo en determinadas ocasiones, bastante bajos. El sujeto 3 (figura 3.34), sin embargo, presenta una distribución de errores más errática, en la que no podemos ver aprendizaje. Si estudiamos la evolución de los errores de manera global, en vez de por target, tampoco podemos ver aprendizaje (figura 3.35).

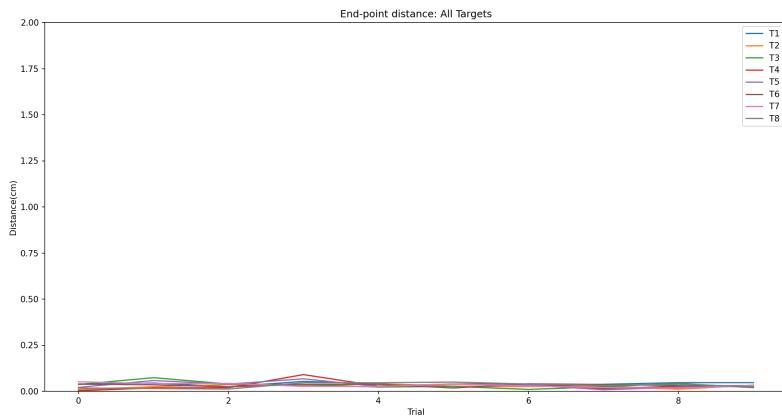


Figura 3.32: Sujeto 1, con fuerza

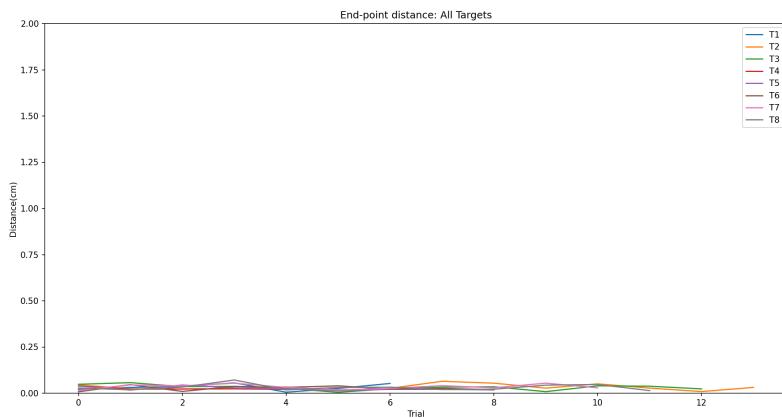


Figura 3.33: Sujeto 2, con fuerza

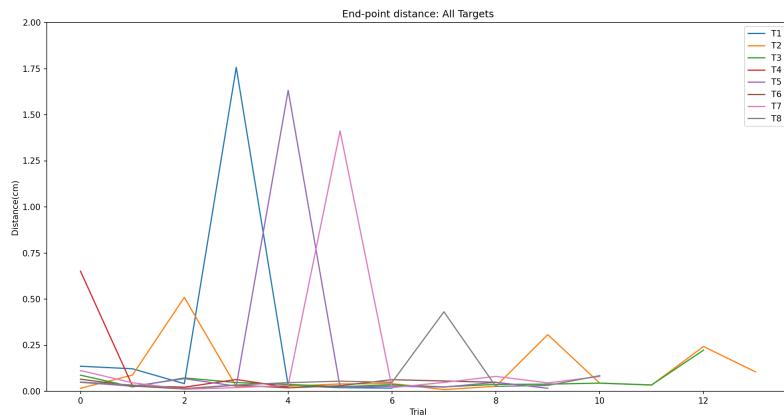


Figura 3.34: Sujeto 3, con fuerza

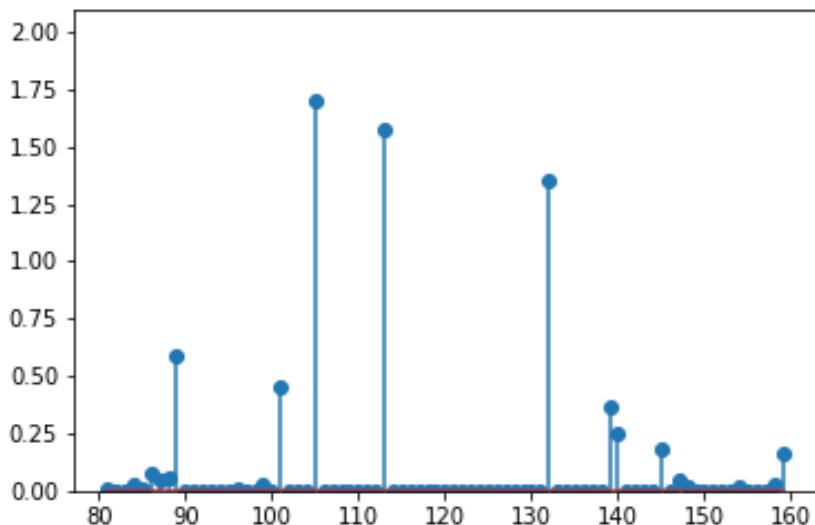


Figura 3.35: Sujeto 3, evolución de los errores totales

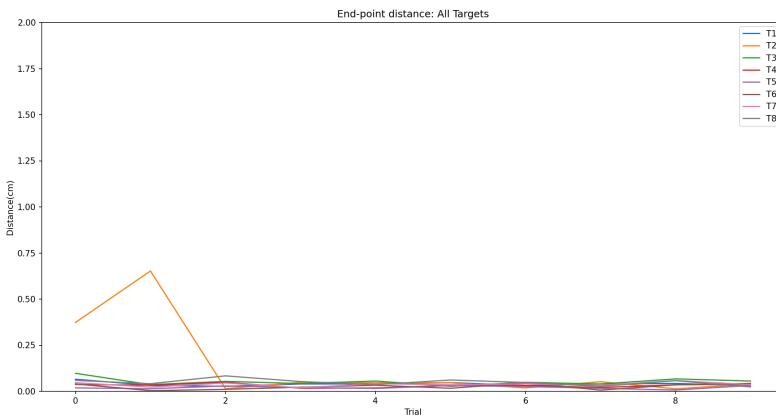


Figura 3.36: Sujeto 4, con fuerza

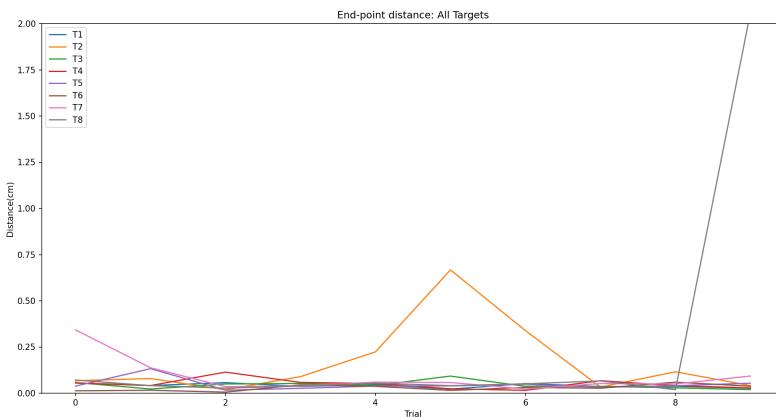


Figura 3.37: Sujeto 5, con fuerza

Respecto a la rapidez en la ejecución de los movimientos podemos observar resultados bastante similares entre los sujetos, siendo el sujeto 3 (figura 3.40) algo más lento que los demás. Sorprenden quizás los resultados del sujeto 5 (figura 3.42), quien, a pesar de hacer las trayectorias más largas, tiene unos tiempos de ejecución similares a los demás y mejores que los del otro sujeto mayor de 60 años. La mejor curva de aprendizaje se puede ver en el sujeto 4 (figura 3.41).

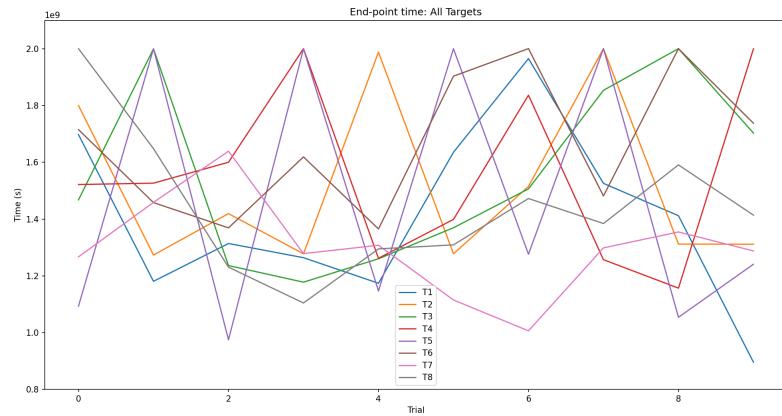


Figura 3.38: Sujeto 1, con fuerza

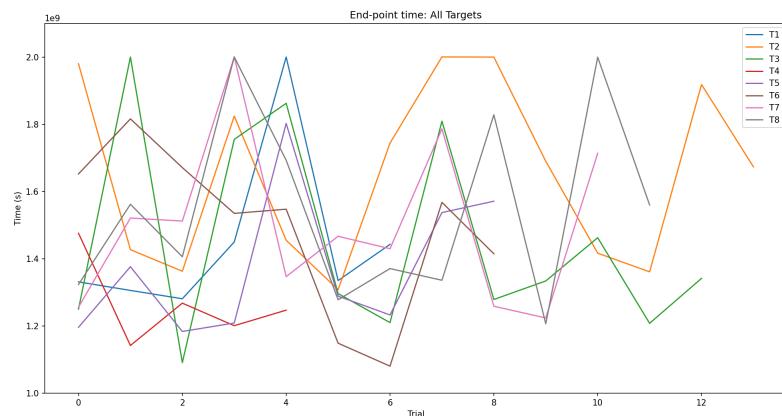


Figura 3.39: Sujeto 2, con fuerza

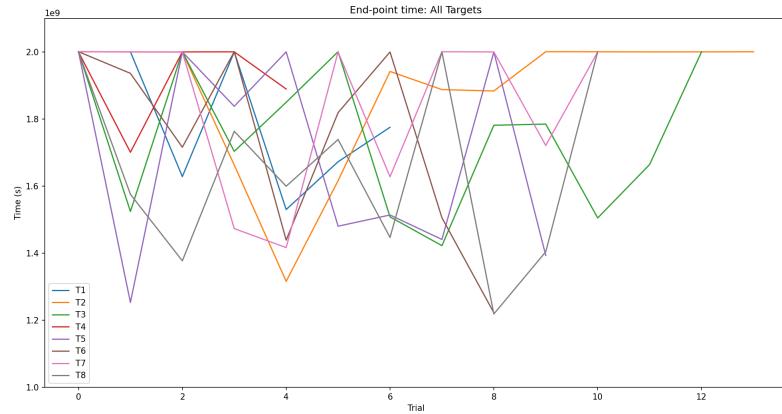


Figura 3.40: Sujeto 3, con fuerza

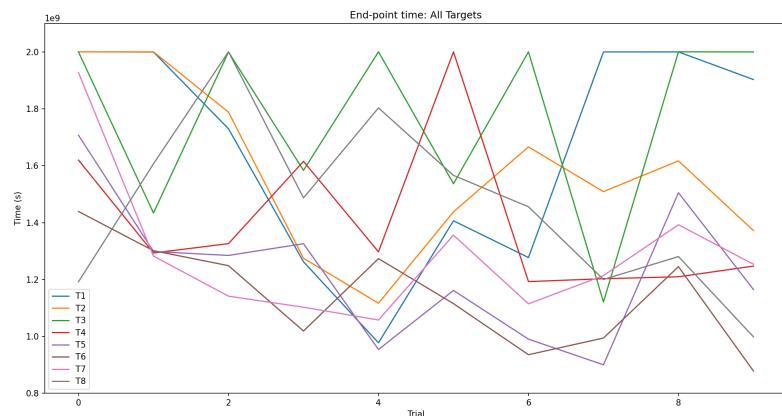


Figura 3.41: Sujeto 4, con fuerza

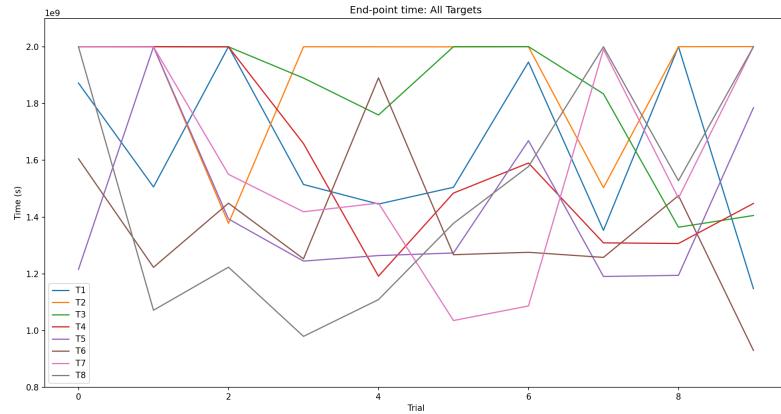


Figura 3.42: Sujeto 5, con fuerza

Fase sin fuerza y sin cursor

En esta fase del experimento no se aplica fuerza en ningún momento al dispositivo, pero en su lugar se ocultará el cursor cuando se inicie el movimiento, es decir, cuando el sujeto vaya desde el punto de referencia al target.

Estudiando las trayectorias que realiza cada sujeto, podemos ver que es el sujeto 5 (figura 3.47) el que realiza las trayectorias menos definidas, mientras que las trayectorias de los demás sujetos son similares.

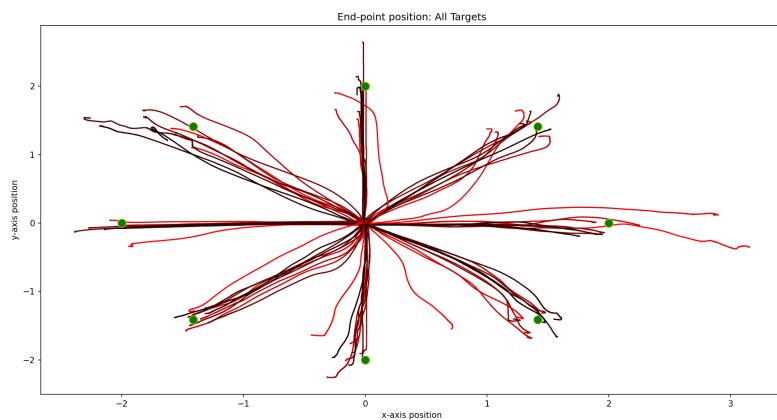


Figura 3.43: Sujeto 1, sin fuerza y sin cursor

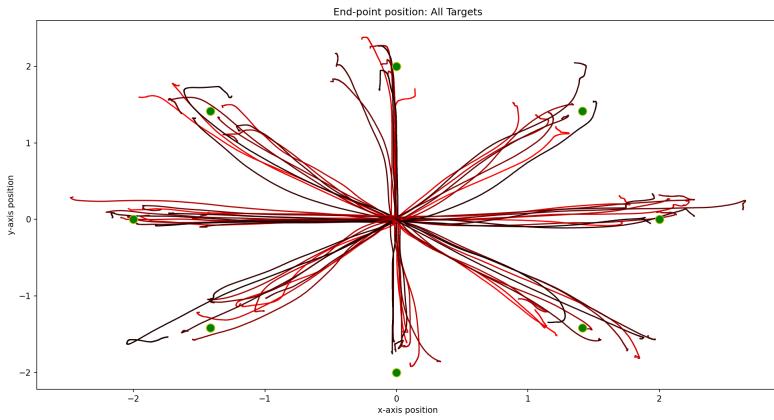


Figura 3.44: Sujeto 2, sin fuerza y sin cursor

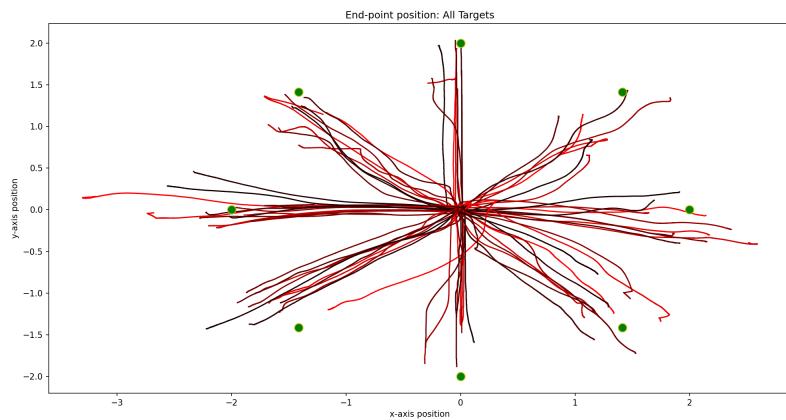


Figura 3.45: Sujeto 3, sin fuerza y sin cursor

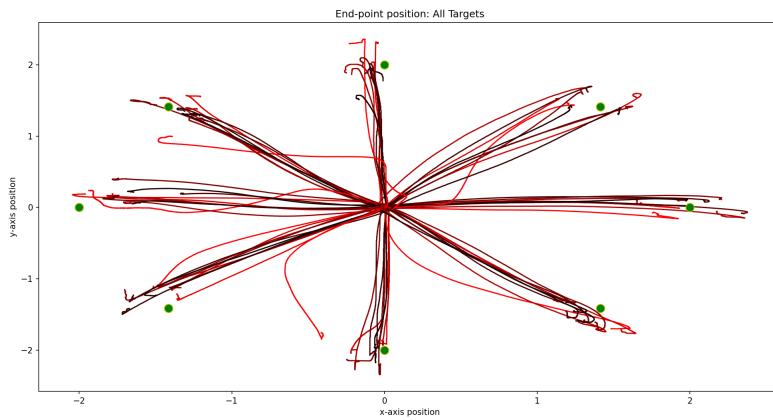


Figura 3.46: Sujeto 4, sin fuerza y sin cursor

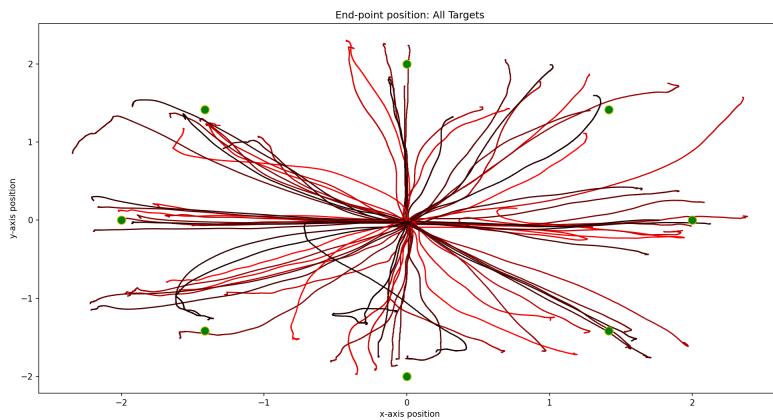


Figura 3.47: Sujeto 5, sin fuerza y sin cursor

Respecto a la distribución de puntos finales los resultados son similares entre todos los sujetos, aunque la precisión de los sujetos jóvenes (figuras 3.48, 3.49 y 3.51) es mejor que la de los sujetos mayores (figuras 3.50 y 3.52).

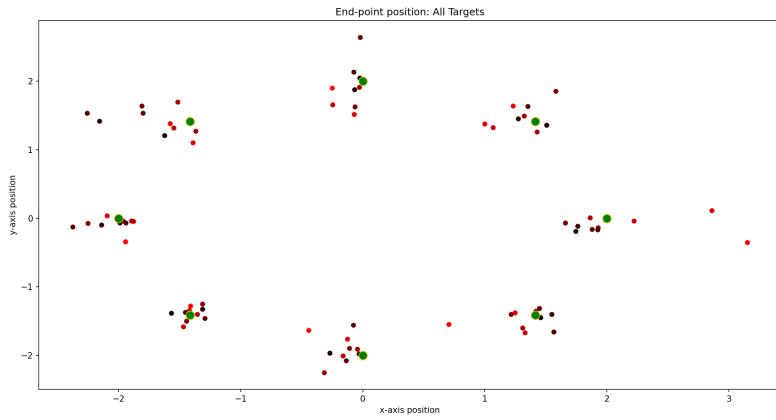


Figura 3.48: Sujeto 1, sin fuerza y sin cursor

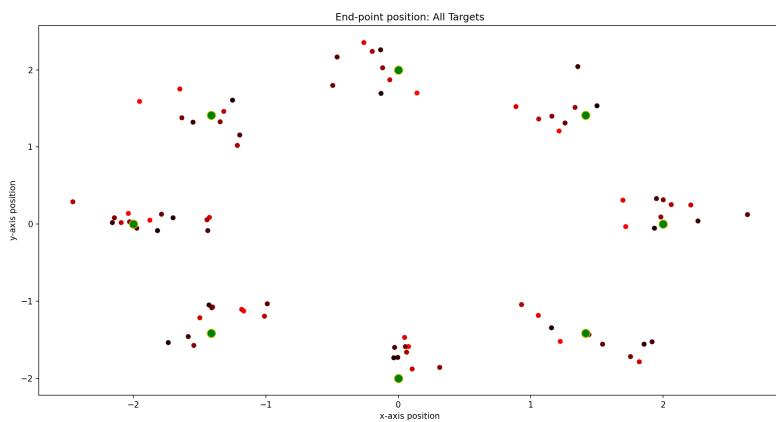


Figura 3.49: Sujeto 2, sin fuerza y sin cursor

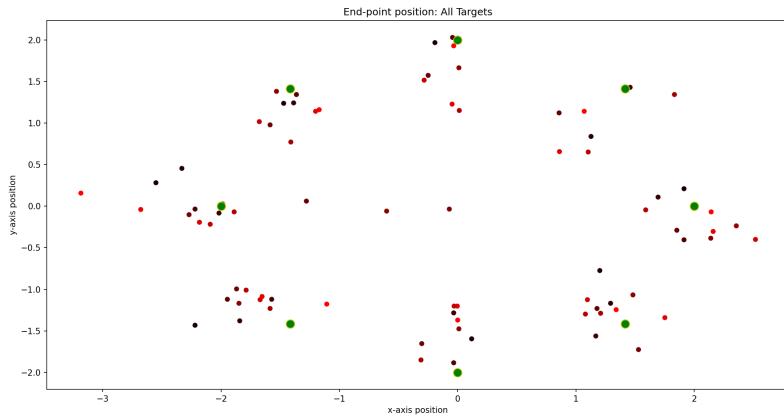


Figura 3.50: Sujeto 3, sin fuerza y sin cursor

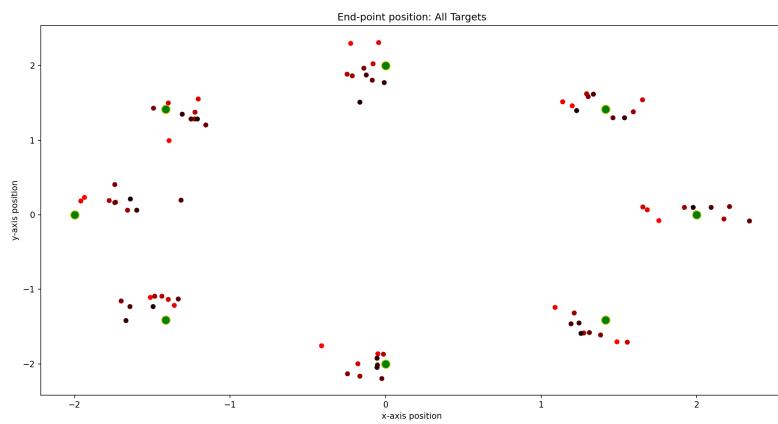


Figura 3.51: Sujeto 4, sin fuerza y sin cursor

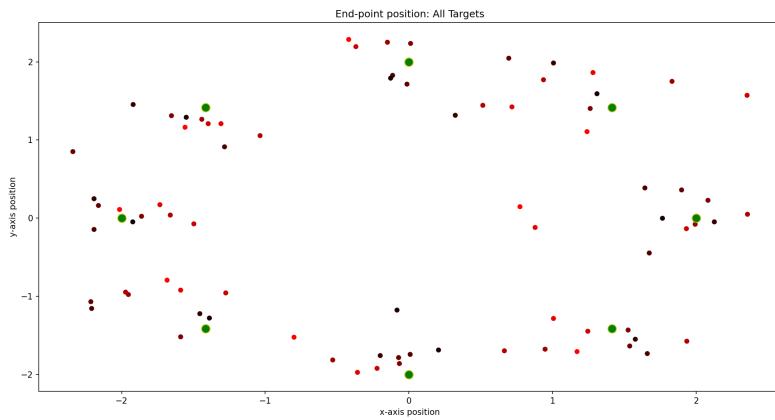


Figura 3.52: Sujeto 5, sin fuerza y sin cursor

Si estudiamos ahora la evolución de los errores en el tiempo podemos ver que en los sujetos jóvenes los errores se mantienen principalmente por debajo de 0.5, mientras que en los sujetos mayores están por debajo de 0.75 ó 1. Respecto a las curvas de aprendizaje podemos ver que en los sujetos 1 (figura 3.53) y 4 (figura 3.56) -jóvenes, en orden- los resultados al final son algo mejores que los resultados al principio. Es interesante remarcar que en el otro sujeto joven -en orden aleatorio- (figura 3.54) no se ve ese aprendizaje, ni siquiera en los targets en los que hay más iteraciones (14). En los sujetos mayores observamos algo parecido: los resultados del sujeto 5 (figura 3.57) son algo mejores al final que al principio, salvo los dos últimos targets (tal vez por el cansancio); mientras que en el sujeto 3 (figura 3.55)) podemos observar que los mejores resultados los tienen los targets con menos iteraciones, y que los targets con más iteraciones no mejoran los resultados.

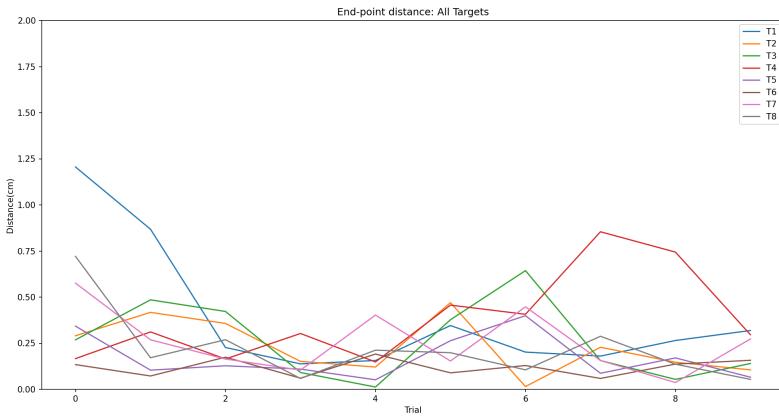


Figura 3.53: Sujeto 1, sin fuerza y sin cursor

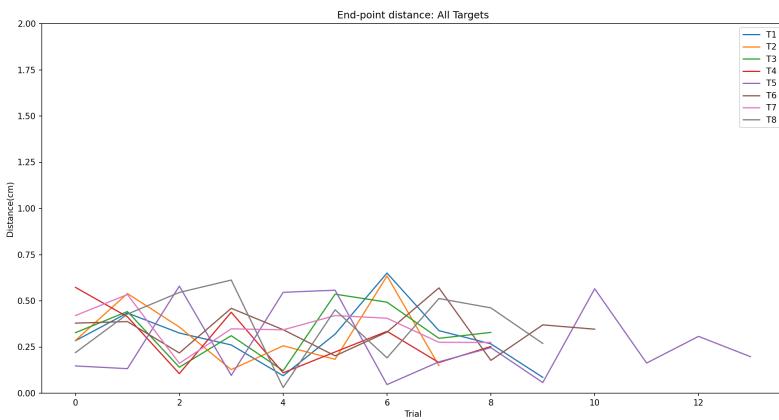


Figura 3.54: Sujeto 2, sin fuerza y sin cursor

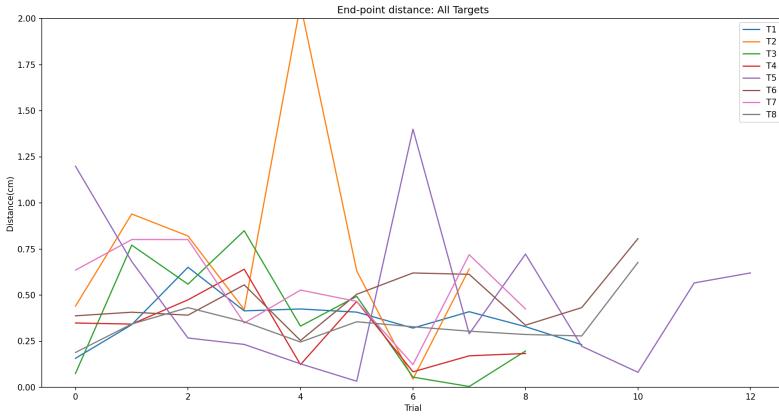


Figura 3.55: Sujeto 3, sin fuerza y sin cursor

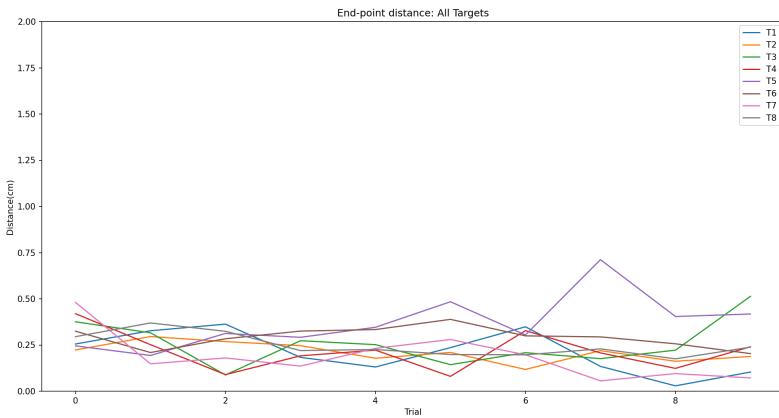


Figura 3.56: Sujeto 4, sin fuerza y sin cursor

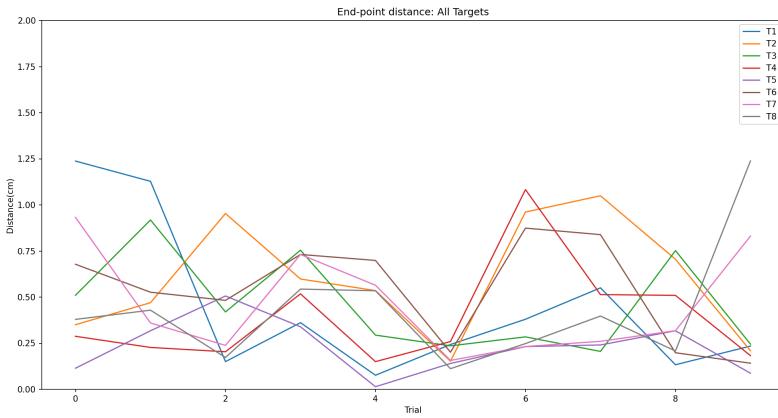


Figura 3.57: Sujeto 5, sin fuerza y sin cursor

Estudiando ahora la evolución respecto al tiempo (figuras 3.58, 3.59, 3.60, 3.61 y 3.62) vemos que en este caso no podemos obtener mucha información de las gráficas, pues en casi ningún punto los sujetos han podido terminar el movimiento a tiempo.

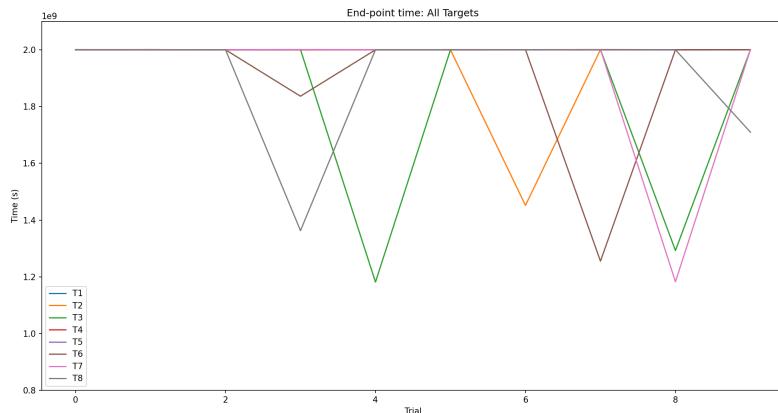


Figura 3.58: Sujeto 1, sin fuerza y sin cursor

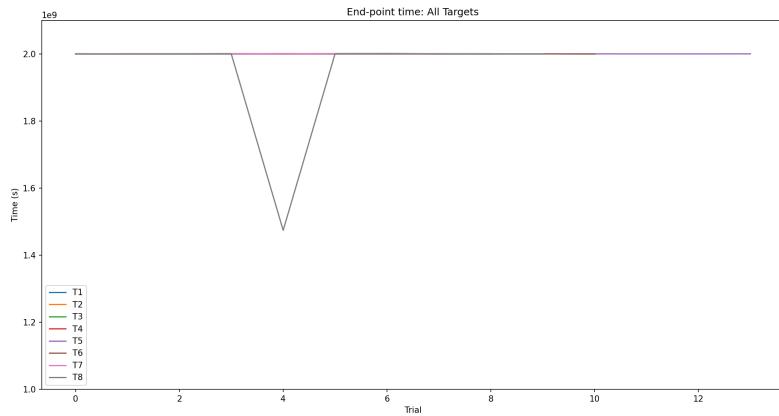


Figura 3.59: Sujeto 2, sin fuerza y sin cursor

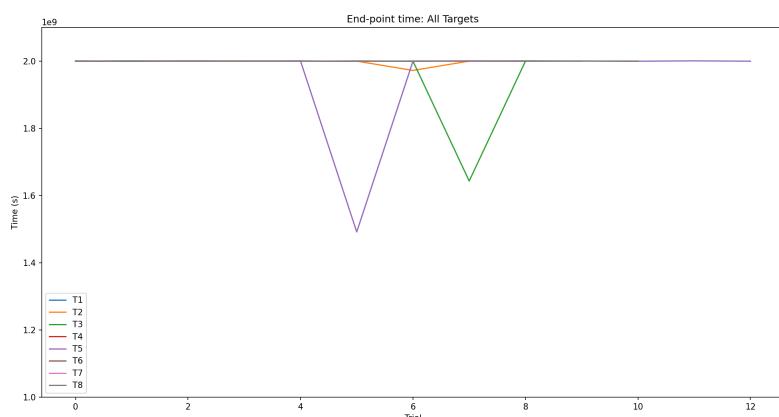


Figura 3.60: Sujeto 3, sin fuerza y sin cursor

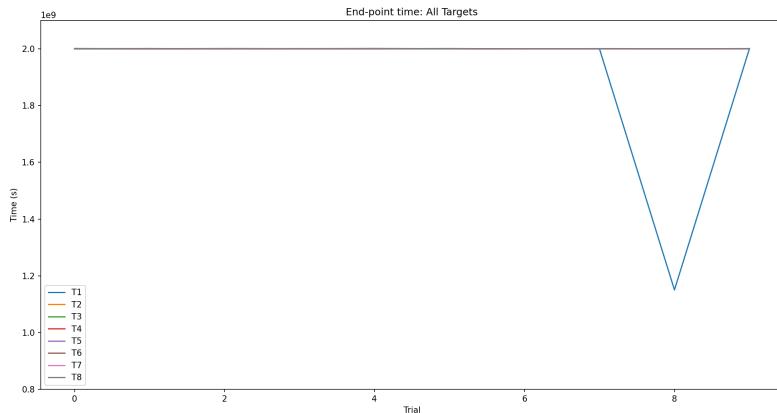


Figura 3.61: Sujeto 4, sin fuerza y sin cursor

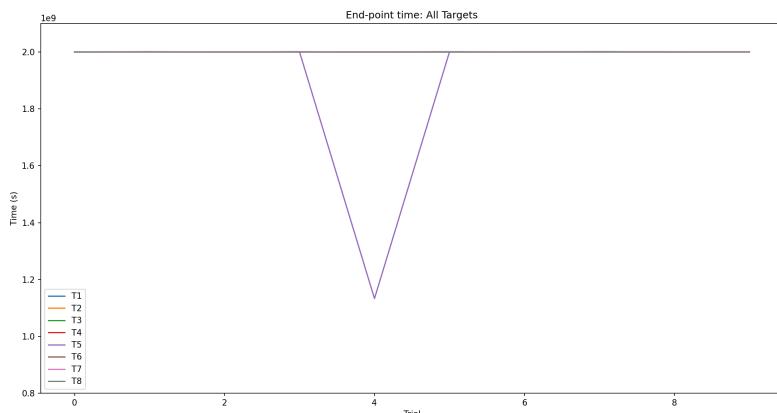


Figura 3.62: Sujeto 5, sin fuerza y sin cursor

Fase con fuerza y sin cursor

En esta fase vamos a aplicar la fuerza de la fase 2, mientras que a la vez ocultamos el cursor como hemos hecho en la fase 3. De esta forma vamos a estudiar cómo los sujetos reaccionan a la fuerza que le hemos aplicado cuando no tienen retroalimentación visual.

Respecto a las trayectorias podemos ver que, esta vez, los que realizan las trayectorias más largas son el sujeto 1 (figura 3.63) y el 5 (figura 3.67) (en la fase sin fuerza y sin cursor eran el sujeto 3 (figura 3.24) y el 5 (figura 3.26)). Aún así podemos observar cómo las trayectorias del sujeto 5 no son tan largas como en la fase 2. En la gráfica del sujeto 4 (figura 3.66) podemos

ver claramente cómo se realiza el aprendizaje del movimiento, pues en las primeras trayectorias en cada punto se puede observar el efecto de la fuerza, que en las iteraciones siguientes se corrige para dar lugar a trayectorias más rectas.

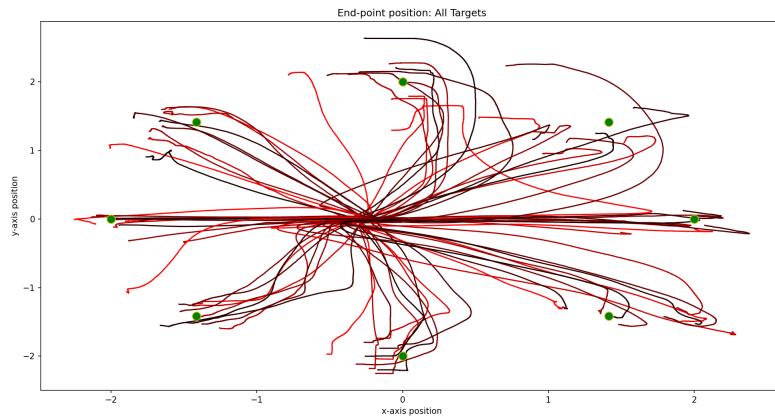


Figura 3.63: Sujeto 1, con fuerza y sin cursor

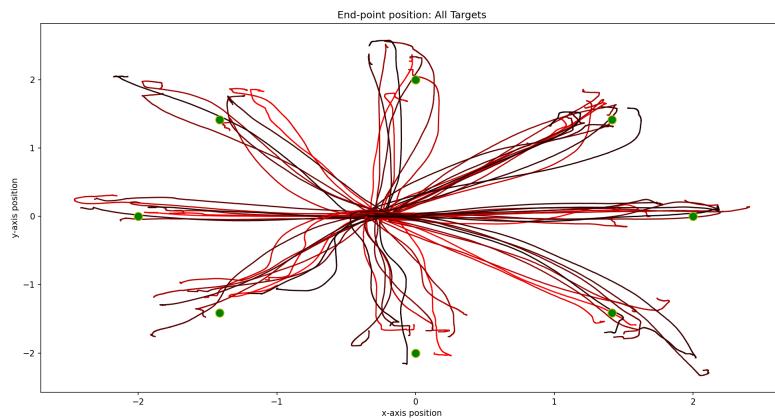


Figura 3.64: Sujeto 2, con fuerza y sin cursor

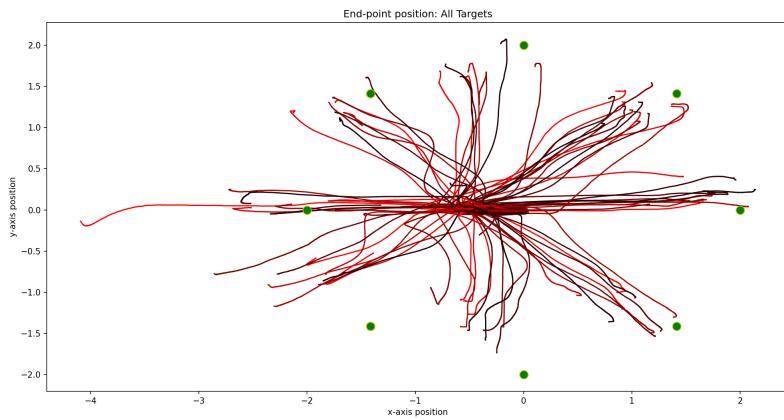


Figura 3.65: Sujeto 3, con fuerza y sin cursor

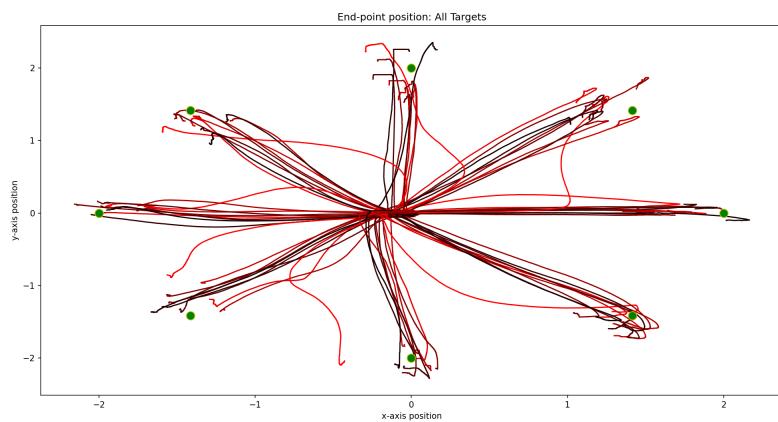


Figura 3.66: Sujeto 4, con fuerza y sin cursor

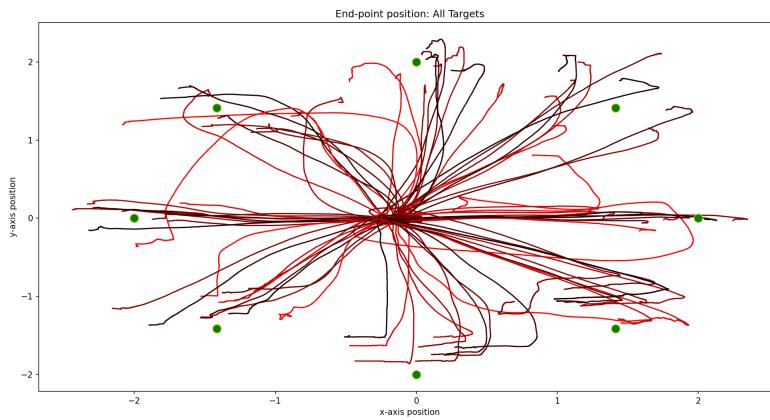


Figura 3.67: Sujeto 5, con fuerza y sin cursor

En la distribución de puntos finales podemos ver que el más preciso de todos los sujetos es el sujeto 4 (figura 3.71), y el menos el sujeto 3 (figura 3.70). Los sujetos 1 (figura 3.68) y 2 (figura 3.69) tienen un desempeño similar, mientras que el del 5 (figura 3.72) es algo peor.

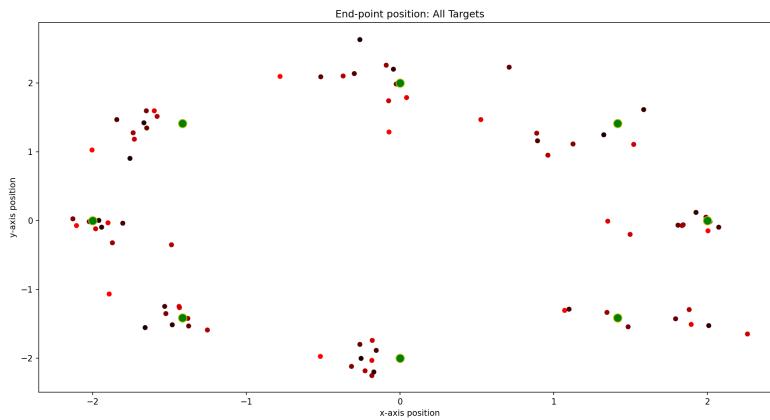


Figura 3.68: Sujeto 1, con fuerza y sin cursor

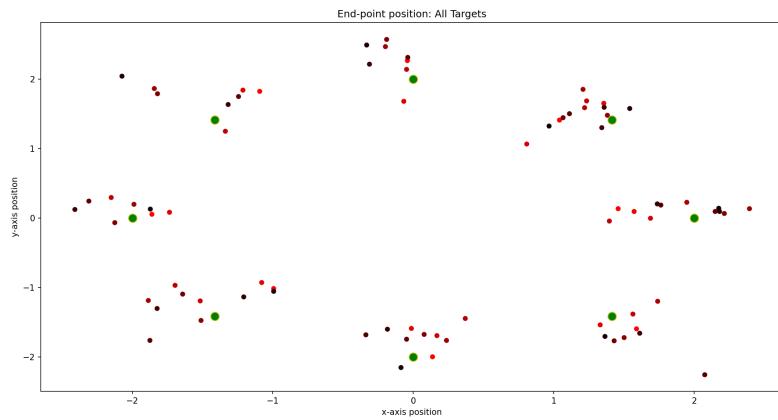


Figura 3.69: Sujeto 2, con fuerza y sin cursor

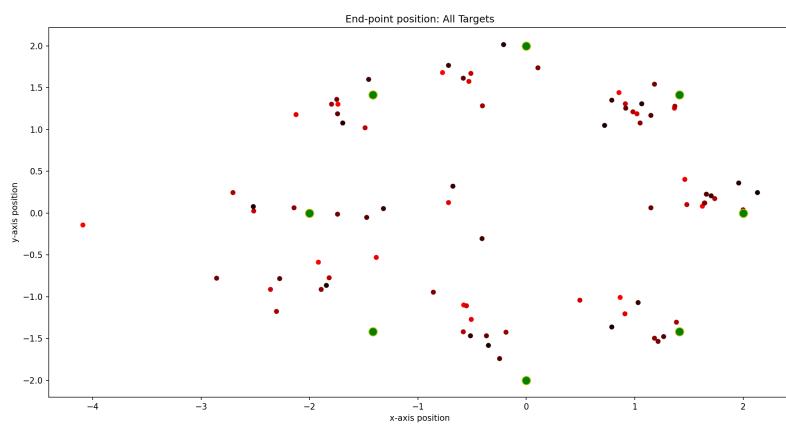


Figura 3.70: Sujeto 3, con fuerza y sin cursor

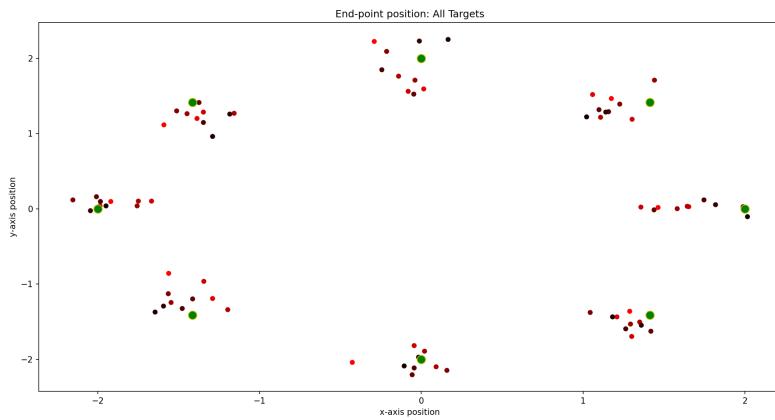


Figura 3.71: Sujeto 4, con fuerza y sin cursor

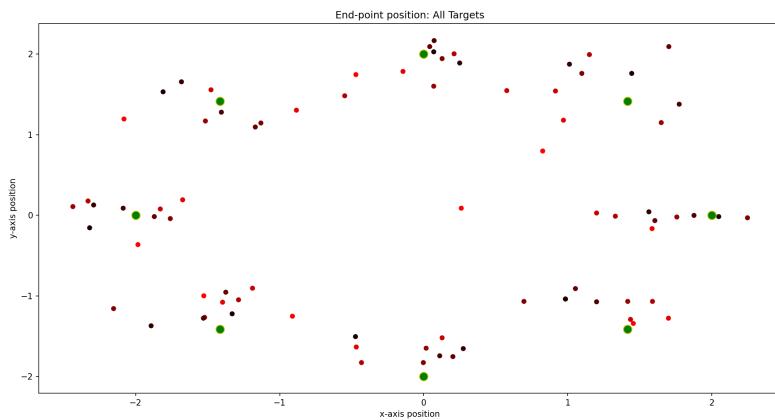


Figura 3.72: Sujeto 5, con fuerza y sin cursor

Vamos ahora a analizar la evolución de los errores en cada target. Podemos ver que el que menos errores tiene es el sujeto 4 (figura 3.76), que sigue manteniéndose por debajo de 0.5, y experimenta un ligero pero no muy claro aprendizaje. Las gráficas del sujeto 1 (figura 3.73) y del sujeto 2 (figura 3.74) son parecidas: en el sujeto 1 podemos observar que los errores en las últimas iteraciones son algo menores, pero en el sujeto 2 vemos otra vez que los targets con más iteraciones (14) no tienen mejor desempeño.

Respecto a los sujetos mayores vemos que aquí sí hay más diferencia entre el sujeto 5 (figura 3.77) y el sujeto 3 (figura 3.75). Mientras que el sujeto 5 muestra cierto aprendizaje (salvo en T8) el sujeto 3 tiene un comportamiento más errático y con mayores errores.

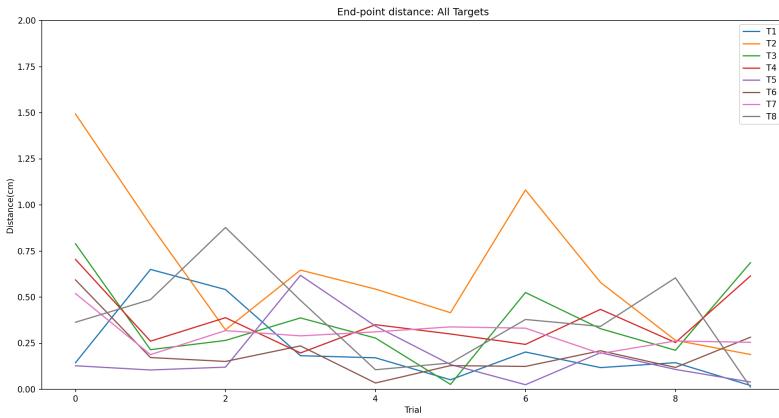


Figura 3.73: Sujeto 1, con fuerza y sin cursor

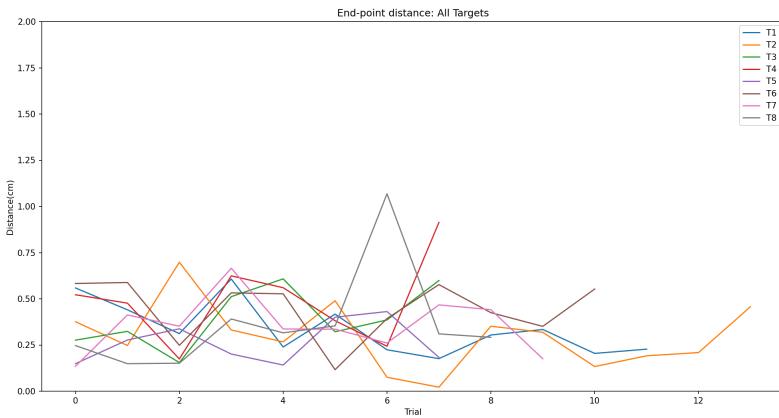


Figura 3.74: Sujeto 2, con fuerza y sin cursor

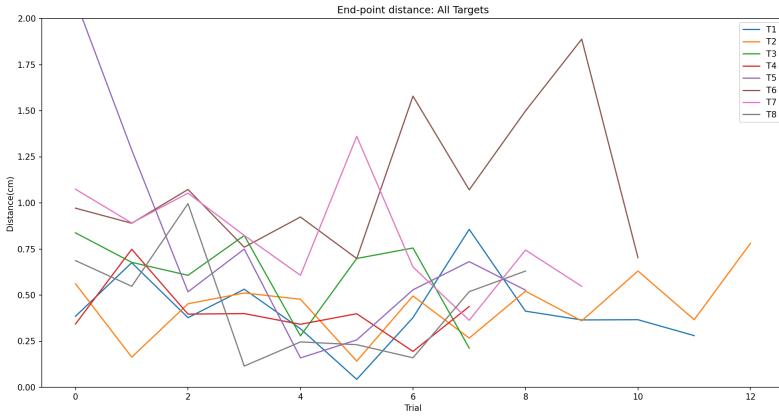


Figura 3.75: Sujeto 3, con fuerza y sin cursor

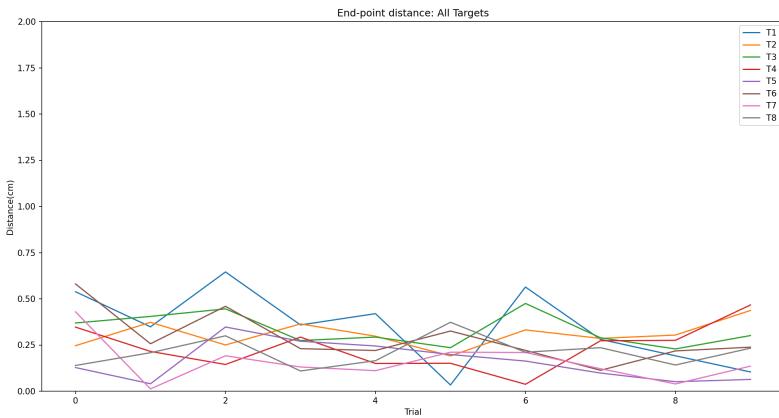


Figura 3.76: Sujeto 4, con fuerza y sin cursor

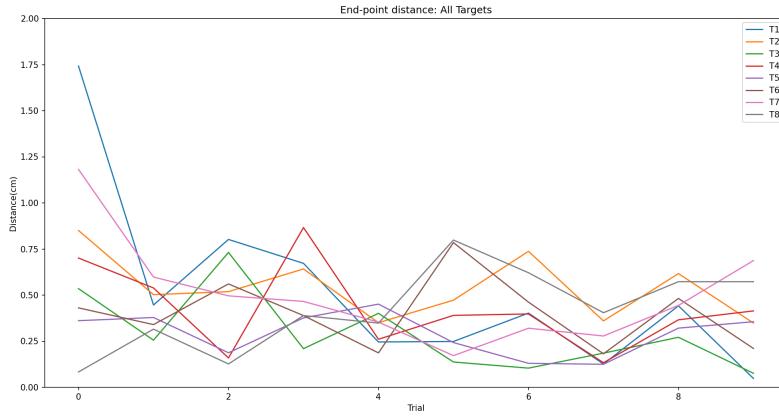


Figura 3.77: Sujeto 5, con fuerza y sin cursor

Respecto a la evolución de la velocidad en este caso volvemos a no poder extraer mucha información, pues en casi ningún punto los sujetos han podido realizar el movimiento a tiempo. En todo caso vuelven a ser los sujetos jóvenes (figuras 3.78, 3.79 y 3.81) los que tienen mejores tiempos.

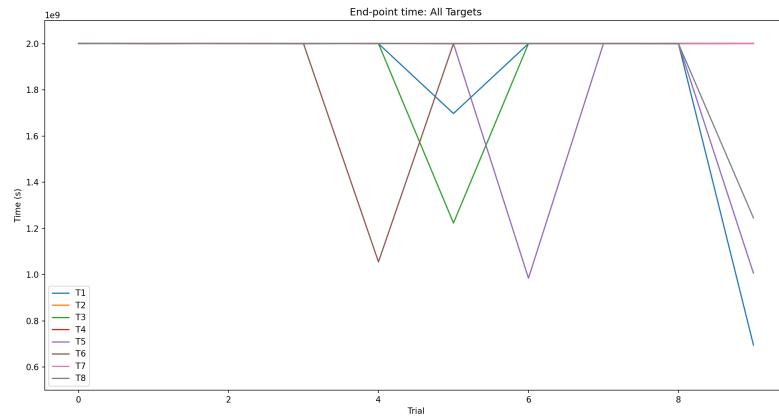


Figura 3.78: Sujeto 1, con fuerza y sin cursor

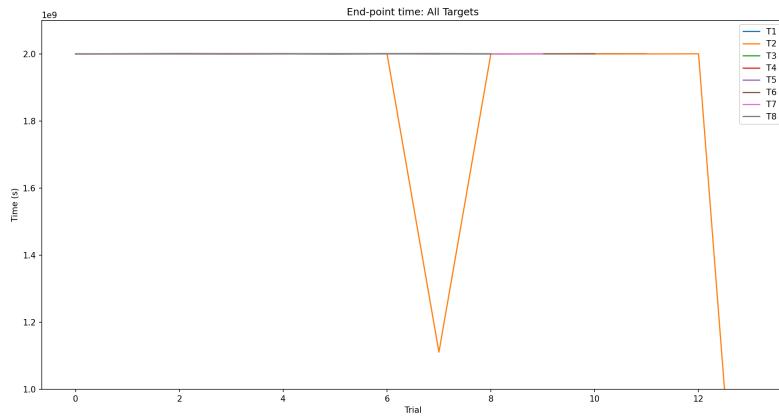


Figura 3.79: Sujeto 2, con fuerza y sin cursor

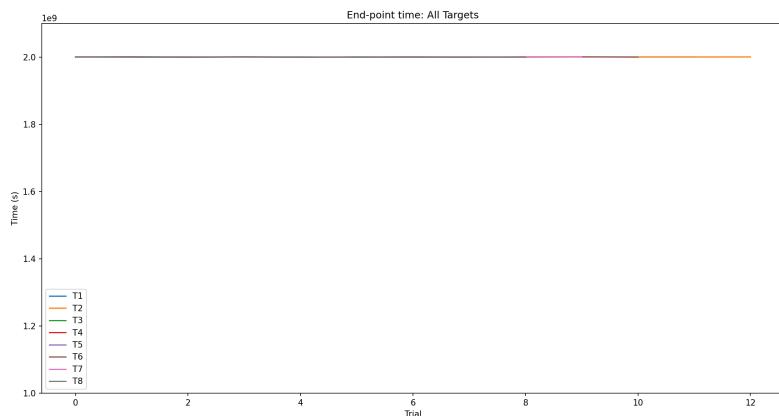


Figura 3.80: Sujeto 3, con fuerza y sin cursor

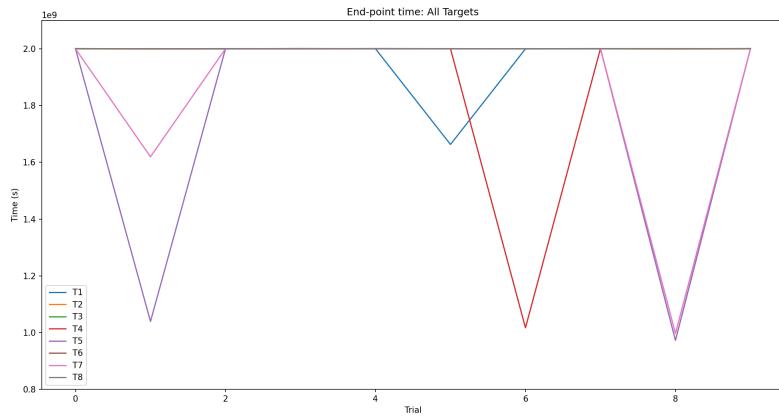


Figura 3.81: Sujeto 4, con fuerza y sin cursor

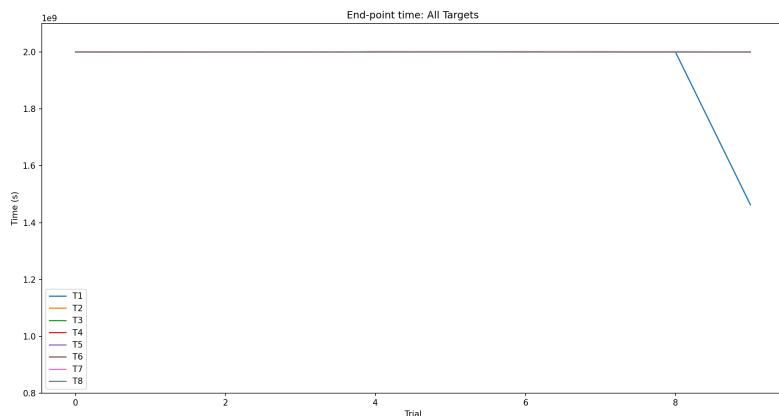


Figura 3.82: Sujeto 5, con fuerza y sin cursor

3.4.2. Análisis de diferentes fases entre el mismo individuo

Vamos ahora a estudiar la evolución de cada sujeto a través de las diferentes fases. De esta forma queremos analizar cómo ha afectado la fuerza que hemos aplicado al dispositivo, y si el sujeto extrae la extrapolación realizada en una fase a otra.

Para ello vamos a estudiar principalmente las trayectorias y los puntos finales, para ver los movimientos realizados y la precisión y exactitud de esos movimientos.

Sujeto 1

En la primera fase podemos ver que los movimientos son muy precisos y exactos (a partir de T3 no hay errores), y las trayectorias, salvo las primeras en cada target, son líneas rectas, especialmente las de los targets T1, T3, T5, T7, que a priori son las más fáciles.

En la segunda fase vemos que se extrapola completamente el aprendizaje realizado en la primera fase, pues no hay errores desde el principio, por lo que la precisión es incluso mayor. Las trayectorias sin embargo son algo más largas, y los movimientos más cortos se producen en los targets T1 y T5, lo que a priori también tiene sentido pues es la dirección en la que estamos aplicando la fuerza. Respecto al tiempo de ejecución las dos fases tienen resultados parecidos.

En la tercera fase, ahora sin retroalimentación del cursor, podemos ver que baja la precisión de los movimientos pero las trayectorias, salvo las primeras de cada target, son líneas rectas.

En la cuarta fase vemos que la precisión baja algo más y tiende a desplazarse en el sentido de la fuerza, especialmente en los targets en los que el movimiento es en el sentido de la fuerza. En las trayectorias se observa ahora más el efecto de la fuerza, y no podemos apreciar claramente que se efecto se contrarreste cuando realizamos más iteraciones.

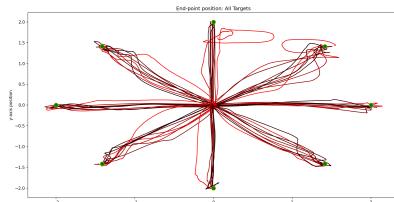


Figura 3.83: Sujeto 1, sin fuerza

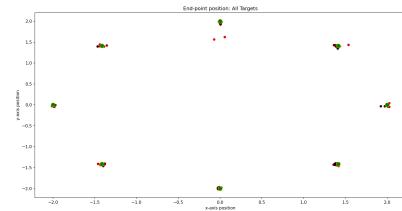


Figura 3.84: Sujeto 1, sin fuerza

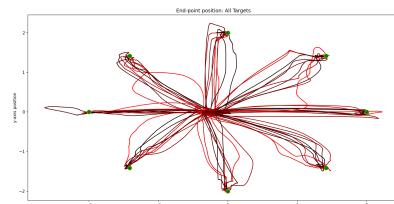


Figura 3.85: Sujeto 1, con fuerza

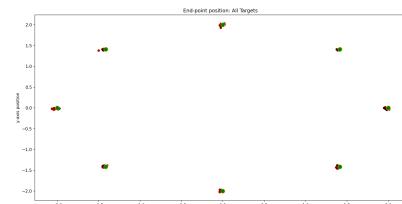


Figura 3.86: Sujeto 1, con fuerza

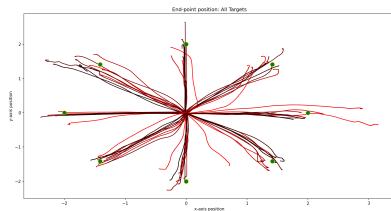


Figura 3.87: Sujeto 1, sin fuerza y sin cursor

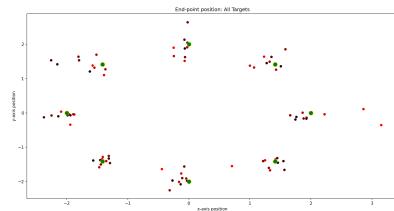


Figura 3.88: Sujeto 1, sin fuerza y sin cursor

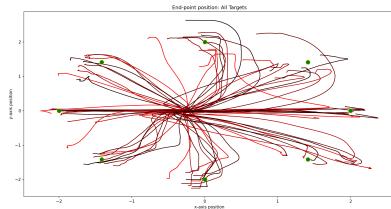


Figura 3.89: Sujeto 1, con fuerza y sin cursor

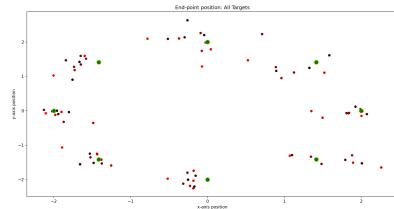


Figura 3.90: Sujeto 1, con fuerza y sin cursor

Sujeto 2

En la primera fase podemos ver que el sujeto 1 realiza correctamente los movimientos desde la primera iteración. Además las trayectorias son bastante rectas.

En la segunda fase vemos que sigue manteniendo un 100 por 100 de precisión, aunque las trayectorias ahora son un poco más largas, especialmente en T6 y T7.

En la tercera fase podemos observar que baja la precisión, pero las trayectorias siguen siendo rectas.

En la cuarta fase los resultados son parecidos a la tercera, por lo que podemos ver que la fuerza que hemos aplicado no ha afectado a la precisión de los movimientos, aunque sí afecta ligeramente a las trayectorias realizadas, que se vuelven algo menos definidas. Las trayectorias más definidas vuelven a ser las que se realizan en la dirección de la fuerza.

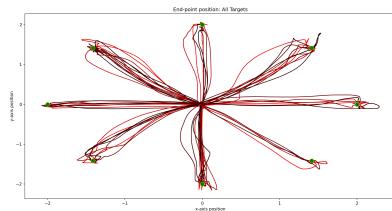


Figura 3.91: Sujeto 2, sin fuerza

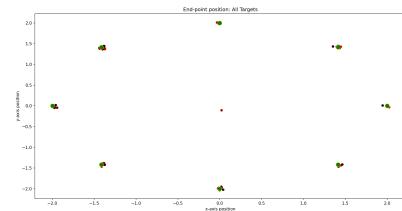


Figura 3.92: Sujeto 2, sin fuerza

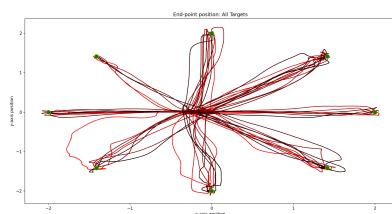


Figura 3.93: Sujeto 2, con fuerza

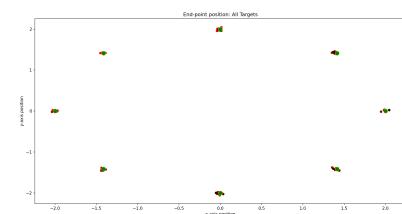


Figura 3.94: Sujeto 2, con fuerza

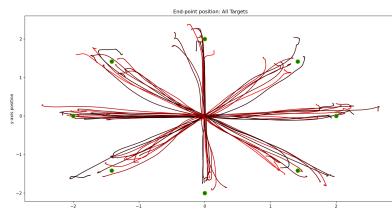


Figura 3.95: Sujeto 2, sin fuerza y sin cursor

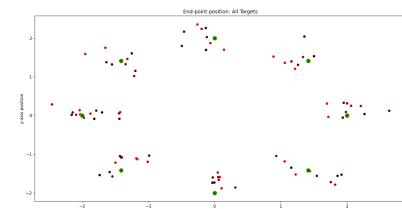


Figura 3.96: Sujeto 2, sin fuerza y sin cursor

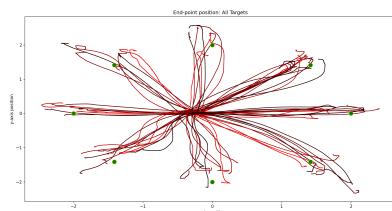


Figura 3.97: Sujeto 2, con fuerza y sin cursor

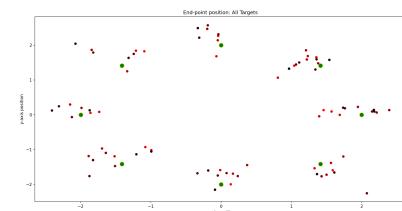


Figura 3.98: Sujeto 2, con fuerza y sin cursor

Sujeto 3

En la primera y la segunda fase vemos unos resultados muy parecidos en cuanto a precisión, aunque respecto a las trayectorias podemos ver que en la fase en la que aplicamos la fuerza están desviadas en el sentido de la fuerza. Aún así esto no afecta a la precisión ni a la velocidad del movimiento.

Respecto a las fases en las que no se muestra el cursor podemos ver que los puntos finales cuando aplicamos fuerza están algo desplazados en la dirección de la fuerza, mientras que al no aplicar fuerza se reparten aleatoriamente alrededor del punto objetivo. Las trayectorias también son algo menos claras al aplicar fuerza. Aún así los resultados son parecidos en términos de exactitud y tiempo.

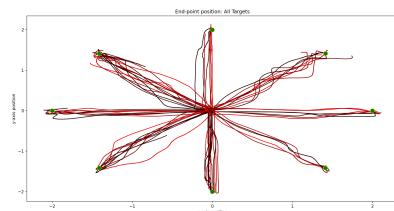


Figura 3.99: Sujeto 3, sin fuerza

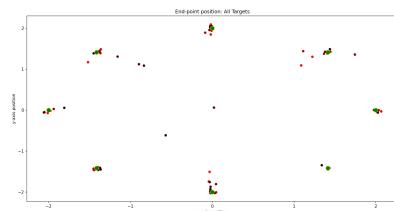


Figura 3.100: Sujeto 3, sin fuerza

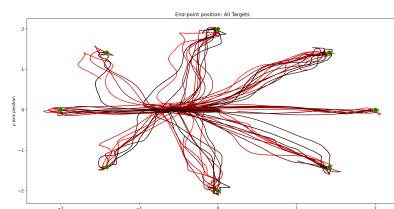


Figura 3.101: Sujeto 3, con fuerza

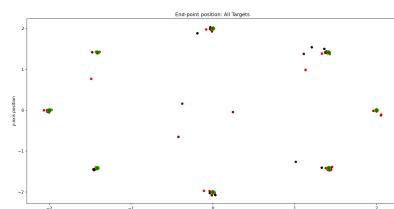


Figura 3.102: Sujeto 3, con fuerza

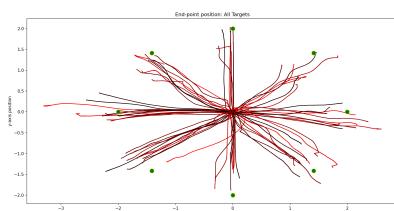


Figura 3.103: Sujeto 3, sin fuerza y sin cursor

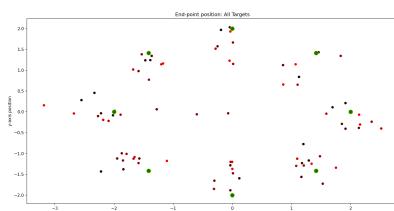


Figura 3.104: Sujeto 3, sin fuerza y sin cursor

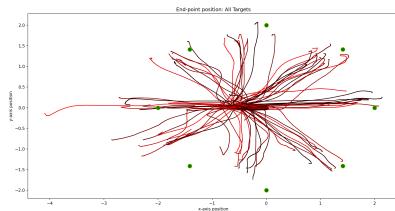


Figura 3.105: Sujeto 3, con fuerza y sin cursor

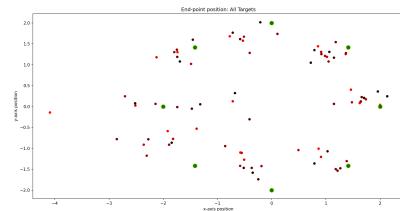


Figura 3.106: Sujeto 3, con fuerza y sin cursor

Sujeto 4

El sujeto 4 es el que presenta los resultados más parecidos entre fases.

Tanto en la fase 1 como en la 2 los movimientos son casi 100 por 100 precisos y las trayectorias son líneas rectas, exceptuando la primera iteración en cada target.

En las fases en las que no hay realimentación del cursor los resultados son muy parecidos, con una precisión casi similar en la fase sin fuerza y con fuerza. En las trayectorias tampoco podemos observar una influencia muy clara de la fuerza.

Por lo tanto en el sujeto 4 no hemos obtenido resultados diferentes que indiquen que tiene sentido diferencias estos dos tipos de movimientos (con fuerza y sin fuerza).

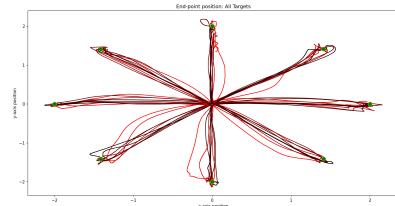


Figura 3.107: Sujeto 4, sin fuerza

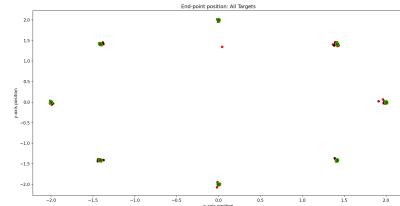


Figura 3.108: Sujeto 4, sin fuerza

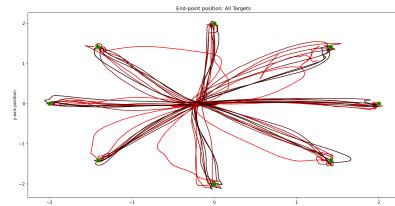


Figura 3.109: Sujeto 4, con fuerza

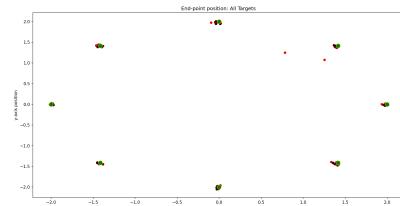


Figura 3.110: Sujeto 4, con fuerza

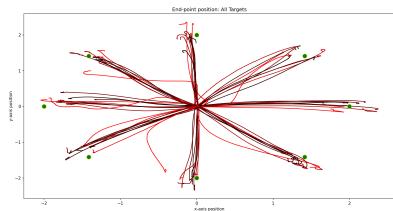


Figura 3.111: Sujeto 4, sin fuerza y sin cursor

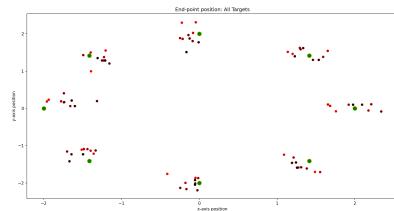


Figura 3.112: Sujeto 4, sin fuerza y sin cursor

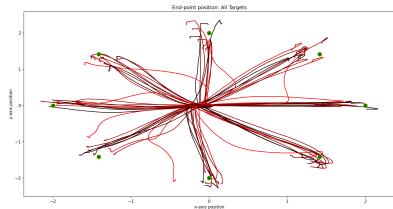


Figura 3.113: Sujeto 4, con fuerza y sin cursor

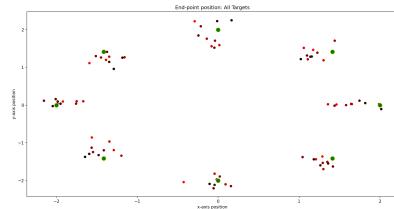


Figura 3.114: Sujeto 4, con fuerza y sin cursor

Sujeto 5

En el sujeto 5 es en el que podemos observar mejor los efectos de la fuerza, aunque solo en las trayectorias.

En la fase 1 podemos observar que se produce un aprendizaje en los dos primeros targets, que luego se extrapolá a los targets siguientes. En la fase con fuerza ese aprendizaje se extrapolá en el primer target, en el segundo target vuelven a aparecer errores y luego se vuelve a conseguir una precisión bastante buena. Por tanto los resultados al final son muy parecidos entre fases.

En cuanto a las trayectorias se puede apreciar con claridad los efectos de la fuerza, pues los movimientos siguen el sentido de la fuerza y no se corrigen cuando se repite el movimiento más veces.

Respecto a las fases 3 y 4 los resultados vuelven a ser muy parecidos, con una precisión casi igual en las dos fases. Es interesante remarcar que las trayectorias curvas que se realizan en la fase 2 se suavizan bastante en la fase 4.

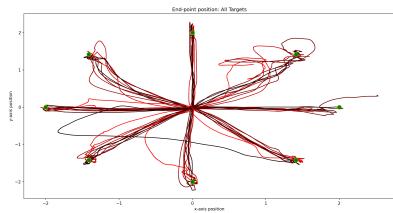


Figura 3.115: Sujeto 5, sin fuerza

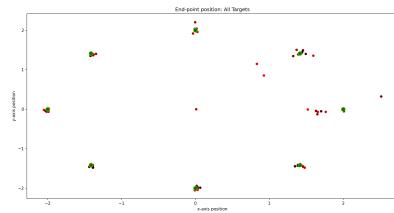


Figura 3.116: Sujeto 5, sin fuerza

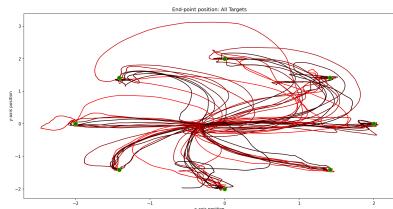


Figura 3.117: Sujeto 5, con fuerza

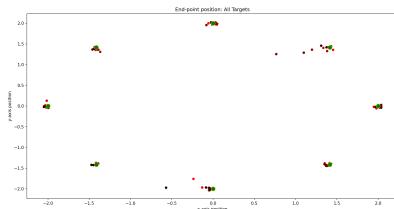


Figura 3.118: Sujeto 5, con fuerza

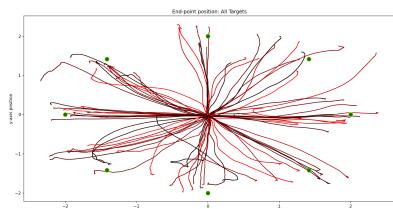


Figura 3.119: Sujeto 5, sin fuerza y sin cursor

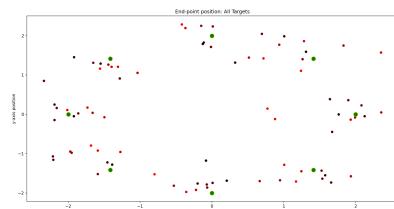


Figura 3.120: Sujeto 5, sin fuerza y sin cursor

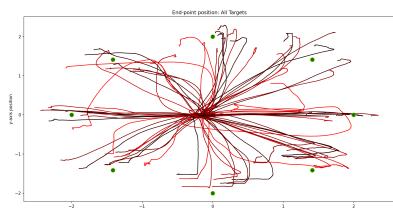


Figura 3.121: Sujeto 5, con fuerza y sin cursor

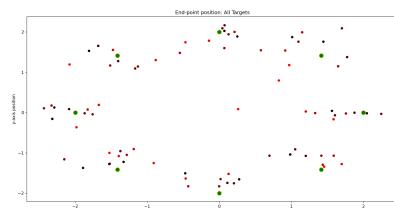


Figura 3.122: Sujeto 5, con fuerza y sin cursor

Capítulo 4

Discusion y conclusiones

Resumen comparación entre sujetos

sin fuerza:

los sujetos jóvenes lo hacen mejor que los mayores los sujetos jóvenes tienen un aprendizaje casi instantaneo, mientras que de los mayores solo muestra aprendizaje el que tiene los puntos en orden. Los tiempos de ejecución del sujeto mayor en orden también son mejores.

con fuerza:

los sujetos jóvenes controlan mejor el efecto de la fuerza (no se dejan llevar e intentan hacer trayectorias más rectas)

los sujetos jóvenes lo hacen bien desde el principio, y el mayor en orden lo hace mucho mejor que el aleatorio

las trayectorias largas no tienen por que afectar a los tiempos de ejecución sin fuerza y sin cursor:

no hay tanta diferencia, algo menos de errores los jóvenes, y algo mejor la curva de aprendizaje los en orden, pero no mucho. Los targets aleatorios que se repiten más no mejoran los resultados, quizá no estamos repitiendo el número de veces necesario.

ninguno termina los movimientos a tiempo

Con fuerza y sin cursor:

no se nota tanto el efecto de la fuerza en las trayectorias

entre los jóvenes no se nota diferencias si es en orden o no, entre los mayores algo más.

mas iteraciones en aleatorio no significa hacerlo mejor

respecto al tiempo no podemos decir nada, salvo que los jóvenes lo hacen algo mejor

Bibliografía

- [1] 3D Systems, 2023. URL <https://es.3dsystems.com/>.
- [2] A. Romero. Álgebra lineal y geometría, 1991.
- [3] Deivi Luzardo, Alirio J. Pena P. Historia del Álgebra lineal hasta los albores del siglo xx, 2006.
- [4] Eric Temple Bell. Historia de las matemáticas, 1945.
- [5] Francisco J. López. Geometría III.
- [6] Khronos Group. Opengl, 2023. URL <https://www.opengl.org/>.
- [7] HTC. Vive pro eye, 2019. URL https://www.vive.com/us/support/vive-pro-eye/category_howto/about-the-headset.html.
- [8] Reza Shadmehr Jun Izawa, Sarah E. Criscimagna-Hemminger. Cerebellar contributions to reach adaptation and learning sensory consequences of action, 2012.
- [9] Celia Arias Martínez. Repositorio de GitHub, 2023. URL <https://github.com/ariasmartinez/tfg>.
- [10] OpenGL. Coordinate systems, 2014. URL <https://learnopengl.com/Getting-started/Coordinate-Systems>.
- [11] OpenGL. GLUT. The OpenGL Utility Toolkit, 2023. URL <https://www.opengl.org/resources/libraries/glut/>.
- [12] 3D Systems. Openhaptics, 2023. URL <https://es.3dsystems.com/haptics-devices/openhaptics>.
- [13] 3D Systems. Dispositivo touch, 2023. URL <https://es.3dsystems.com/haptics-devices/touch>.
- [14] 3D Systems. OpenHaptics for Windows Developer Edition v3.5, 2023. URL https://support.3dsystems.com/s/article/OpenHaptics-for-Windows-Developer-Edition-v35?language=en_US.

- [15] I. Siegler I. Israël A. Berthoz Y.P. Ivanenko, I. Viaud-Delmon. The vestibulo-ocular reflex and angular displacement perception in darkness in humans: adaptation to a virtual environment, 1997.
- [16] Eling D. de Bruin Yu Imaoka, Andri Flury. Assessing saccadic eye movements with head-mounted display virtual reality technology, 2020.