.

UNTREF

UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO

Desarrollo de software de transmisión de gestos oculares a un robot colaborativo

Jorge Ignacio Fourmentel

Tutor: Mariano Sardón
Co-Tutor: Germán Ito

Trabajo de final de grado
Licenciatura en Artes Electrónicas
Universidad Nacional de Tres de Febrero
2017

Agradecimientos

A mis tutores, Mariano Sardón y Germán Ito, por toda la ayuda y la orientación para realizar este proyecto.

A Joaquín Bua-Bonaldi y Nicolás Mezzanotte de Patagonia CNC Machines, por prestar el equipamiento al proyecto para permitir su desarrollo.

A mis padres, mis hermanas y mis amigos, por todo el apoyo brindado y por estar siempre cuando los necesito.

A los chicos del Laboratorio Arte Electrónico e Inteligencia Artificial de la UNTREF, de los cuales aprendí, compartí y disfrute mucho todo este tiempo.

A la Universidad nacional de Tres de Febrero, y a la educación pública.

INDICE

Resumen

El presente proyecto consiste en el desarrollo de una interfaz de software que permite conectar un robot colaborativo industrial con un sistema de seguimiento ocular (*Eyetracking*) de alta precisión, de forma que el dispositivo de seguimiento ocular (*Eyetracker*) suministre posiciones oculares en pixeles, para luego transponerlas a posiciones en milímetros correspondientes a un plano, donde el robot realiza sus movimientos.

Este documento se divide en dos partes. En la primera se presentarán diferentes temáticas fundantes del proyecto, empezando por la visión, donde se mencionarán los conceptos necesarios para comprender el funcionamiento del ojo y la dinámica ocular durante un proceso de observación. Posteriormente se tratará el eyetracking, como la disciplina encargada de estudiar estos movimientos utilizando dispositivos de alta tecnología. Luego la robótica, partiendo desde sus nociones más generales hasta adentrarnos en la robótica colaborativa. Finalmente, se entregarán conceptos básicos de programación, centrada en el lenguaje Python, siendo éste el lenguaje utilizado para el desarrollo de la interfaz.

En la segunda parte, se describirá el desarrollo paso a paso del proyecto, donde se explicarán cada uno de los pasos realizados para llevar a cabo el desarrollo de la interfaz.

Este proyecto fue llevado a cabo en el marco del proyecto de arte y neurociencia de Mariano Sardón y Mariano Sigman.

Palabras claves: Seguimiento Ocular, Robótica colaborativa, Python, PyGaze, Visualización en Tiempo Real.

Keywords: Eye Tracking, Collaborative Robotics, Python, PyGaze, Real time Visualization.

1- De los conocimientos necesarios para entender y desarrollar el proyecto

1.1 La visión.

1.1.1 El ojo

La percepción visual es uno de los sentidos más importantes para el ser humano, dado que la información que se puede obtener sobre los objetos que nos rodean a partir de la luz que éstos reflejan es variada y muy valiosa. (1) Una porción enorme de las actividades que realizamos día a día están íntimamente asociadas con la visión: desde agarrar un objeto hasta interactuar con otros individuos.

Cuando la luz llega al ojo, inicialmente atraviesa una estructura hemisférica transparente llamada córnea, cuya finalidad es proteger al ojo (Figura 1.1). Detrás de la córnea se encuentra el humor acuoso (una cavidad rellena de líquido) y luego la pupila, que es un diafragma o apertura que regula la cantidad de luz que entra al ojo, de forma similar a la de una cámara de fotografía. La apertura de la pupila es controlada por el iris. (2)

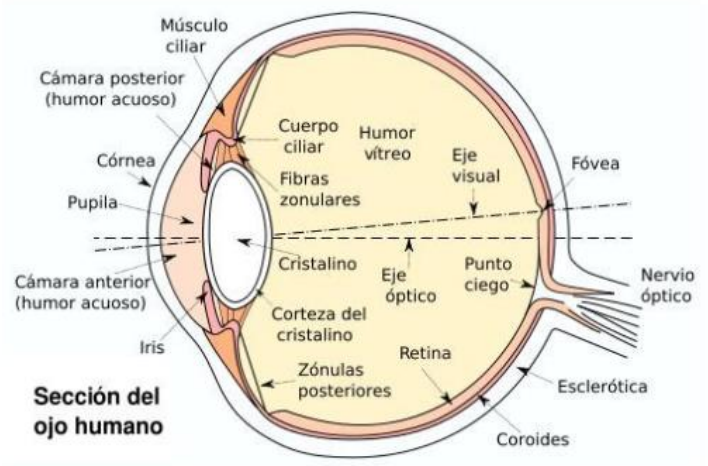


Figura 1.1 Esquema de un ojo humano con una enumeración de sus componentes.

Detrás del iris se encuentra el cristalino, el cual tiene forma de lente biconvexa y su curvatura se modifica para lograr enfocar a distintas distancias. Su índice de refracción es distinto al de los medios que lo rodean, lo que le permite refractar la luz y formar las imágenes dentro del ojo.

En la parte posterior del ojo se encuentra la retina, que es la zona en donde se hallan las células fotoreceptoras que convierten la luz en señales nerviosas. Para poder ver una imagen definida, la luz refractada por el cristalino debe formar la imagen sobre la retina. Existen allí dos tipos de fotorreceptores: los conos y los bastones. Los conos son los responsables de la visión en colores (existen tres tipos de conos, cada uno sensible a un rango de longitudes de onda distinto), mientras que los bastones detectan intensidad de luz.

Sin embargo, la forma en que estos fotorreceptores están distribuidos a lo largo de la retina no es uniforme: existe una mayor concentración de conos en el centro de la retina (Figura 1.2), mientras que los bastones se encuentran en la periferia. La zona central, compuesta únicamente por conos, se conoce como **fóvea**, y es el punto de máxima resolución de nuestra retina. La zona inmediatamente cercana a la fóvea es la **parafóvea**, en la cual aún es posible recolectar valiosa información para la lectura (comprendiendo 5° alrededor del punto de fijación) y el resto de la retina es la **perifóvea**, de donde prácticamente no es posible extraer información relevante a la lectura.

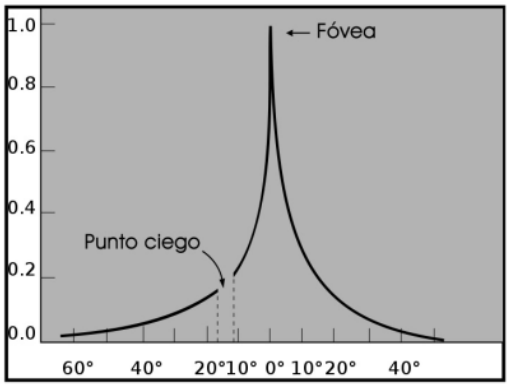


Figura 1.2 Resolución espacial del ojo humano. La zona central de máxima resolución se conoce como fóvea. La agudeza visual disminuye rápidamente al alejarse del centro.

Finalmente, la información visual recolectada en la retina es enviada al cerebro a través del nervio óptico. El punto de entrada del nervio óptico no tiene bastones ni conos, y por lo tanto no se puede ver en esta región de la retina, por lo que se la denomina *punto ciego*.

1.1.2 Movimientos oculares

El ojo es un órgano que es fuertemente selectivo por su propia estructura. El campo visual cubre una región sumamente amplia del espacio frente al observador – 180 grados horizontalmente y 130 verticalmente -- lo cual implica, una selección que no existe por ejemplo en la audición. (3)

La resolución de esta información en la retina no es homogénea. Ésta es máxima en una región central de 1 ó 2 grados de diámetro, la fóvea, y decae rápidamente hacia la periferia (1). El único modo de obtener información detallada de diferentes áreas de la escena visual es redirigiendo el ojo de modo que los objetos relevantes caigan secuencialmente en la fóvea.

Tipos de movimiento ocular

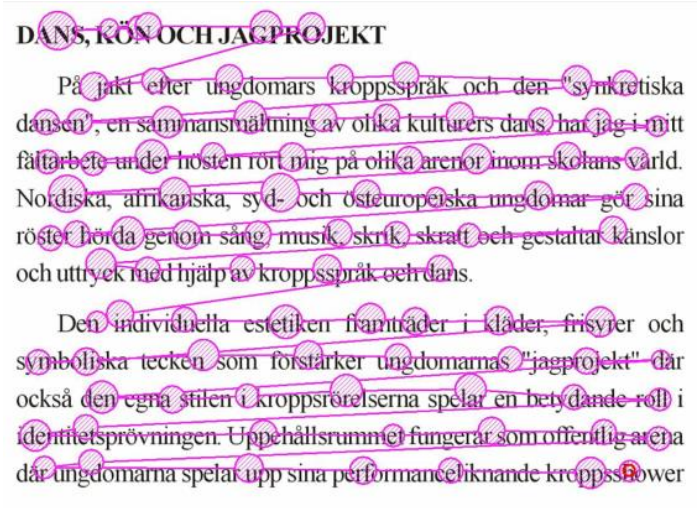
Existen distintos tipos de movimientos de los ojos, caracterizados por su fusión y su fisiología (1). Sin embargo, no todo movimiento de los ojos es cognitivamente relevante. Los movimientos que nos importan en este trabajo son los llamados movimientos sacádicos.

Las sacadas son movimientos rápidos y abruptos cuya función es traer un nuevo objeto de interés a la fóvea. Una sacada toma típicamente 150-200 ms en planearse y ejecutarse, y el movimiento en

sí mismo no toma más de 30 ms, alcanzando velocidades de hasta 900 grados/segundo. (3) Estos movimientos son interrumpidos por las fijaciones, intervalos de tiempo en los que la posición del ojo permanece relativamente estable, y durante los cuales se extrae información de la escena. El proceso de mover y fijar se realiza normalmente entre 3 o 4 veces por segundo, en donde las fijaciones tienen una duración media aproximada entre 200ms y 300ms (4).

Es durante las fijaciones que el sistema visual obtiene información sobre objetos y propiedades del ambiente. Es notable que no se tenga una experiencia consciente del movimiento de la imagen retinal durante las sacadas. De hecho, se ha mostrado que el sistema visual es ciego durante las sacadas, fenómeno conocido como supresión sacádica.

En el 1800, los estudios en relación a los movimientos oculares se realizaban mediante la observación directa. Para ese entonces, en París, Louis Emile Javal observó que la lectura no implica un suave barrido de los ojos a lo largo del texto (como se suponía), sino una serie de paradas cortas (llamadas fijaciones) y sacadas rápidas.


DANS, KÖN OCH JAGPROJEKT

På jakt efter ungdomars kroppsspråk och den "synkretiska dansen", en sammansmältning av olika kulturers dans, har jag i mitt fältarbete under hösten rört mig på olika arenor inom skolans värld. Nordiska, afrikanska, syd- och östeuropeiska ungdomar gör sina röster hörda genom sång, musik, skrik, skratt och gestaltar känslor och uttryck med hjälp av kroppsspråk och dans.

Den individuella estetiken framträder i kläder, frisyrer och symboliska tecken som förstärker ungdomarnas "jagprojekt" där också den egna stilen i kroppsrörelserna spelar en betydande roll i identitetsprövningen. Uppehållsrummet fungerar som offentlig arena där ungdomarna spelar upp sina performanceliknande kroppsshower

Figura 1.3 Ejemplo de fijaciones y sacadas en un texto. Este es el patrón típico del movimiento de los ojos en una lectura.

Yarbus fue un pionero del estudio de la exploración sacádica de imágenes, en la década de los ‘50s y ‘60s. En sus investigaciones (5), los participantes debían observar imágenes complejas con el objeto de responder a distintas preguntas. Una de las conclusiones fundamentales de su experiencia es que la secuencia de fijaciones depende fuertemente de la información que se desea extraer de la escena.

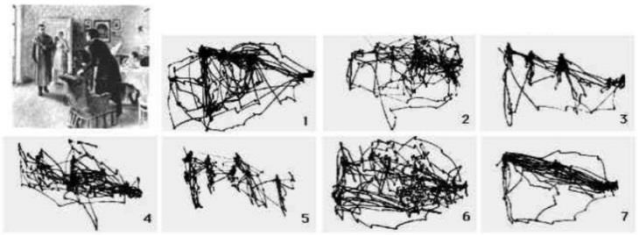


Figura 1.4 Estudio realizado por Yarbus en 1967 donde se contempla que la tarea encomendada influye en el movimiento de los ojos.

1.2 Eyetracking

El eyetracking, o seguimiento ocular, es el proceso mediante el cual se analiza tanto el punto donde se fija la mirada como el movimiento del ojo en relación a la cabeza. Un eyetracker, es el dispositivo que permite realizar estas mediciones, los mismos son usados en investigaciones sobre la visión, estudios de marketing, diseño de productos y también como dispositivos de interacción humana. Existen varios métodos para medir la posición de los ojos, las variantes más populares usan video para extraer estas posiciones.

1.2.1 Historia

A principios de 1900, Edmund Huey construyó el primer prototipo de un seguidor de ojos con una especie de lente de contacto con un agujero que se colocaba en los ojos del sujeto a seguir (6). El objetivo estaba conectado a un puntero de aluminio que se movía en respuesta al movimiento del ojo, este era un método intrusivo para analizar los movimientos oculares.

El primer seguidor de ojos no intrusivo fue construido por Guy Thomas Buswell. Éste utilizaba haces de luz que se reflejan en el ojo y luego eran grabados en fílmico. Buswell se dedicó a realizar estudios sistemáticos en los procesos involucrados en la lectura y la visualización de imágenes.

En 1980 se empezó a utilizar los dispositivos de seguimiento ocular para resolver preguntas relacionadas con la interacción persona-computadora. Además, los ordenadores permitieron a los investigadores utilizar el seguimiento de los ojos en tiempo real, principalmente para ayudar a los usuarios con discapacidades. (7)

Otra área de investigación reciente se centra en el desarrollo Web. En esta se puede incluir cómo reaccionan los usuarios a los menús desplegables o saber en lo que se centran en una página web para que el desarrollador seleccione el lugar óptimo dónde colocar un anuncio. Según Hoffman, el consenso actual es que la atención visual siempre va un poco (de 100 a 250 ms) por delante de los ojos. Pero tan pronto como se mueve la atención a una nueva posición, los ojos hacen lo mismo. (8)

1.2.2 Tipos de seguidores oculares

Dentro del eyetracking se pueden distinguir diferentes tipos de seguimientos oculares, entre los cuales se pueden encontrar:

- ***Mediante censado invasivo:*** Funcionan utilizando algo adjunto al ojo, como una lente de contacto especial con un espejo incorporado o un sensor de campo magnético (9).
- **Mediante censado no invasivo:** En este tipo de censado no existe necesidad de contacto con el ojo. El mismo es el tipo de seguimiento utilizado durante el desarrollo y las pruebas del proyecto.

- **Mediante potenciales eléctricos:** utilizan el potencial eléctrico medido con electrodos colocados alrededor de los ojos para detectar el movimiento. Este tipo de censado tiene la ventaja de que permite operar en la plena oscuridad. (10)

Censado no invasivo

Este tipo de censado utilizado no requiere contacto con el ojo, funciona a través de una luz, por lo general luz infrarroja, que es reflejada en los ojos y se capta mediante una cámara de video o algún otro sensor óptico. Se utiliza el contraste para ubicar el centro de la pupila, y luego el vector resultante entre el centro de la pupila y el reflejo de la córnea se utiliza para determinar la dirección de la mirada.

1.2.3 Eyetracker Eyelink 1000 Plus

Para el desarrollo del presente trabajo, como dispositivo de seguimiento ocular, se utilizó un *Eyelink 1000 Plus,* de SR-Research, el cual se puede utilizar en diferentes configuraciones, cada cual con sus capacidades propias, permitiendo al sistema adaptarse dinámicamente a las necesidades del experimento a desarrollar.

Este es un dispositivo de censado no invasivo, que funciona emitiendo un haz de luz infrarroja en dirección de los ojos del sujeto a ser analizado, y mediante el reflejo de dicho haz de luz sobre la córnea, determina en donde se está posando la mirada. La cámara del Eyelink 1000 Plus opera en una frecuencia de 2000hz de forma monocular, y a 1000hz operando de manera binocular. Estas altas frecuencias de muestreo permiten observar en detalle todas las características de la dinámica ocular, fijaciones, sacadas y regresiones. Con una baja frecuencia de muestreo, el recorrido del ojo entre una fijación y otra se vería como un recorrido casi lineal, en cambio con altas frecuencias de muestreo se puede observar que esto no es así. (11)

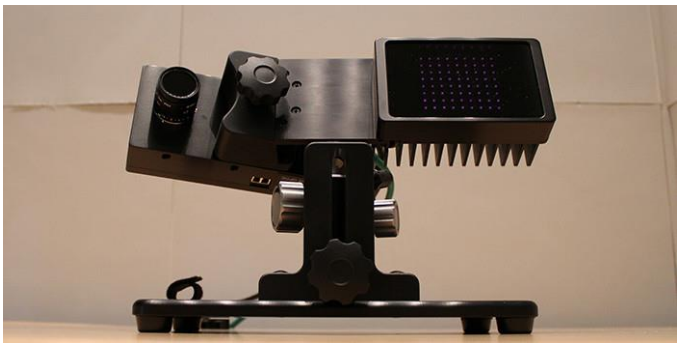


Figura 1.5 Eyetracker tipo Eyelink 1000 plus de Sr Research labs.

El EyeLink 1000 funciona a partir de dos computadoras, una que registra los datos y realiza la calibración, PC HOST, y la otra en donde se corre el experimento, PC DISPLAY, dedicada exclusivamente a la presentación de estímulos. Cuando finaliza el procedimiento, la computadora central envía los datos a la computadora de presentación en donde los analiza. En la base de datos queda guardada la información de todas las fjaciones y sacadas, su posición en pixeles y su duración en milisegundos.

1.3 Robótica

1.3.1 Introducción a la robótica.

La Robótica es una ciencia o rama de la tecnología, que estudia el diseño y construcción de máquinas capaces de desempeñar tareas realizadas por el ser humano o que requieren del uso de inteligencia.

Desde la época de los griegos se intentó crear dispositivos que tuvieran un movimiento sin fin, que no fuera controlado ni supervisado por personas. En los siglos XVII y XVIII fue la construcción de autómatas humanoides fabricados con mecanismos de relojería por Jacques de Vaucanson, como El flautista, una figura de tamaño natural de un pastor que tocaba el tambor y la flauta y tenía un repertorio de doce canciones.

Concepto de robótica

De forma general, la Robótica se define como: El conjunto de conocimientos teóricos y prácticos que permiten concebir, realizar y automatizar sistemas basados en estructuras mecánicas poli articuladas, dotados de un determinado grado de "inteligencia" y destinados a la producción industrial o a la sustitución del hombre en diversas tareas.

Un sistema Robótico puede describirse como: "Aquel que es capaz de recibir información, de comprender su entorno a través del empleo de modelos, de formular y de ejecutar planes, y de controlar o supervisar su operación". La Robótica es esencialmente pluridisciplinaria y se apoya en gran medida en los progresos de la microelectrónica y de la informática, así como en los de nuevas disciplinas tales como el reconocimiento de patrones y de inteligencia artificial. (12)

La Robótica abre una nueva y decisiva etapa en el actual proceso de mecanización y automatización creciente de los procesos de producción. Consiste esencialmente en la sustitución de máquinas o sistemas automáticos que realizan operaciones concretas, por dispositivos mecánicos que realizan operaciones concretas, por dispositivos mecánicos de uso general, dotados de varios grados de libertad en sus movimientos y capaces de adaptarse a la automatización de un número muy variado de procesos y operaciones.

Concepto de robot

La definición de un robot ha sido siempre controversial, dentro de un espectro muy amplio. A un extremo se encuentra la versión ligada a la ciencia ficción, un humanoide o un androide, con características antropomórficas. En el otro extremo se encuentran los eficientes y repetitivos robots ligados a los procesos de producción industrial. La normativa ISO 8373, define a un robot como “un manipulador multipropósito, reprogramable y automáticamente controlable diseñado para mover piezas, herramientas o dispositivos especializados a través movimientos programados para la realización de una variedad de tareas” (13).

Primera aparición de cada término.

La palabra Robot surge con la obra RUR, los "Robots Universales de Rossum" de Karel Capek (1890 – 1938). Es una palabra checoslovaca que significa trabajador, sirviente. Sin embargo podemos encontrar en casi todos los mitos de las diversas culturas una referencia a la posibilidad de crear un ente con inteligencia, desde el Popol-Vuh de nuestros antepasados mayas hasta el Golem del judaísmo. (12)

Isaac Asimov (1920 – 1992) fue un escritor de ciencia ficción que tuvo un profundo impacto en la historia de la robótica. Durante su primer periodo colaboró con el pensamiento en relación a lo que podría llegar a transformarse la robótica. Escribió una serie de cuentos cortos en relación al tema. Fue en la serie de cuentos *Yo, Robot* (1950), particularmente en "Roundaround" donde Asimov introduce las Tres Leyes de la Robótica y además aparece por primera vez la palabra Robótica, utilizada para hacer referencia a la tecnología involucrada en el diseño, construcción y operación de robots.

Las tres leyes de la robótica se definen de la siguiente manera:

- Un robot no hará daño a un ser humano o, por inacción, permitir que un ser humano sufra daño.
- Un robot debe hacer o realizar las órdenes dadas por los seres humanos, excepto si estas órdenes entrasen en conflicto con la 1ª Ley.
- Un robot debe proteger su propia existencia en la medida en que esta protección no entre en conflicto con la 1ª o la 2ª Ley.

1.3.2 Robótica Industrial.

En 1946 un inventor con el nombre de George C. Devol patentó un dispositivo de reproducción utilizado para controlar máquinas. Esta búsqueda por la automatización lo llevó a realizar otro invento en 1954, en el que aplicó a una patente escribiendo "El presente invento hace posible por primera vez una máquina multipropósito que posee una aplicación universal en una vasta diversidad de aplicaciones donde sea necesario un control cíclico". Devol llamo a este invento, Universal Automation, ó, *Unimation*. Posteriormente, en 1961, Devol junto con un equipo de ingenieros desarrollaría el *Unimate*, el primer robot industrial, entregado a General Motors. (12)

1.3.3 Robótica Colaborativa.

Hoy en día los trabajadores pueden trabajar a la par de los *cobots* (Robots Colaborativos) sin requerir vallas de seguridad, o cualquier sectorización que evite la proximidad entre el cuerpo humano y la máquina. Este es uno de los componentes principales de su naturaleza colaborativa, pero no la totalidad.

Los cobots requieren cumplir con estándares particulares de seguridad, determinados por las normas ISO. Según las normas ISO 10218, hay cuatro características de seguridad que debe poseer un cobot. (14)

1 Freno de seguridad por monitoreo:

Esta característica está presente en los cobots que trabajen en mayormente solos, con raras intervenciones por parte de los humanos, ingresando en su área de trabajo. Si fuera necesario realizar algún ajuste, mover alguna pieza del área de trabajo, el robot automáticamente se detendría ante la presencia de humano, pero solo temporalmente, para automáticamente retomar sus tareas cuando el área de trabajo se encuentre despejada.

2 Controlable a mano:

Es necesario el control a mano, a manera de guía, para enseñar los movimientos que el robot debe realizar. Esto consta de guiar al robot por una secuencia de pasos requeridas para completar la tarea deseada. Los robots poseen sensores de posición y fuerza que registran todos los movimientos que el humano le implementa, para luego poder reproducirlos.

3 Monitoreo de velocidad y distancia:

Para aplicaciones que requieren mayor intervención del hombre, un Sistema de visión laser se instala en el área de trabajo para censar la proximidad de un humano para con el robot. El robot reducirá su velocidad a medida que un trabajador se acerque, frenando por complete cuando se encuentre demasiado cerca. El robot retomará su actividad paulatinamente a medida que el trabajador se aleje.

4 Límites de fuerza:

Cobots como los de Universal Robots, poseen funciones que limitan la potencia y la fuerza del robot. Los mismos leen las fuerzas en sus juntas (presión, resistencia, impacto) mediante sensores embebidos en la mecánica. Al momento de sentir un impacto, o una fuerza no esperada el robot se detendrá por completo o revertirá su curso. Su estructura, suavidad y formas redondeadas evitan cualquier daño mediante el impacto.

Los robots colaborativos surgen como respuesta a un problema que estaba afectando las principales líneas de producción, problemas del tipo ergonómico que afectaban a los trabajadores provocando lesiones, y por ende la pérdida de tiempo de producción. Estos problemas afectaban a todas las fábricas de automóviles, pero principalmente a General Motors, quienes se pusieron el objetivo de solucionar esta problemática. (15)

Así como en el pasado, fueron los primeros en implementar un robot industrial en sus procesos productivos, el Unimate, se encargaron de reunir un equipo de expertos de diferentes universidades con el objetivo de desarrollar un tipo de robot que pueda trabajar a la par de la gente, sin que signifique un riesgo para los trabajadores.

En un principio, se buscaba que el robot realizara movimientos controlados por computadora, sin margen de error. Para garantizar seguridad, el robot solo podría sostener una carga, pero no moverla. Todo movimiento debía provenir de un trabajador.

Posteriormente cuando la interacción humano-máquina se volvió más dinámica, se introdujo el término robot colaborativo. La primer patente un robot colaborativo fue presentada en 1999.

1.3.4 Robot Colaborativo Universal Robots UR-3

El robot UR-3 de Universal Robots proporciona una elevada precisión para los entornos de producción más pequeños. Las tareas en las que el UR-3 se destaca incluyen: montaje de objetos pequeños, pegado, atornillado, manejo de herramientas, soldadura y pintura. (16)

Diseñado para los entornos de menor alcance, el UR-3 tiene un radio de alcance de 500mm y puede manejar cargas de hasta 3kg. Esto hace posible su uso en espacios limitados y añade valor a prácticamente cualquier entorno de producción.

El UR3 también cuenta con la función InfiniteSpin en su última articulación, lo que permite utilizarlo para tareas de atornillado sin necesidad de incorporar otro dispositivo. Es un robot fácil de programar y es seguro; permitiéndole trabajar sin un vallado de seguridad que lo separe de los trabajadores.

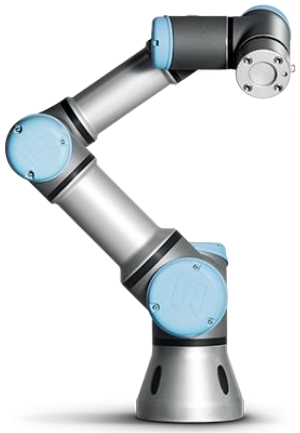


Figura 1.6 Robot Ur-3 de Universal Robots

1.3.4 Posicionamiento del robot en sus 6 ejes.

El robot UR 3 posee 6 ejes, por ende el cálculo de sus coordenadas resulta en una ecuación muy compleja, que involucra vectores de rotación. Uno podría decir simplemente, quiero que el robot tome una determinada posición de X – Y –Z, pero el robot puede llegar a esa posición de muchas maneras. Por ejemplo, el TCP o *Tool Center Point* (Centro de la herramienta), puede tomar la posición desde la izquierda, la derecha, desde arriba, desde abajo, y siempre ser la misma posición X-Y-Z. (17)

Para un mejor entendimiento nos concentraremos primero en las posiciones X, Y y Z.

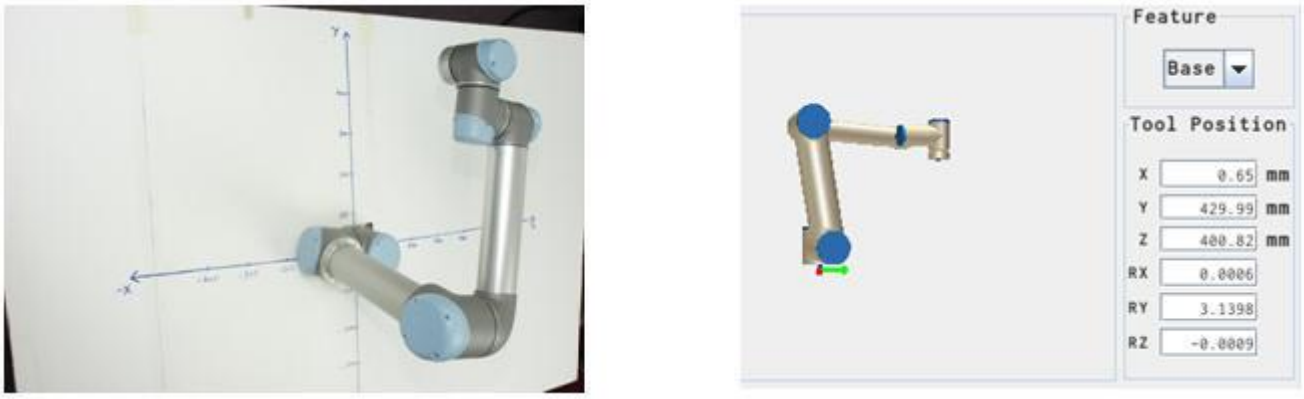


Figura 1.7 Posicionamiento del UR3 en un eje de coordenadas.

Consideremos al robot en un eje de coordenadas X – Y – Z.

El gráfico muestra al robot en la posición X = 0 mm, Y = 430 mm y Z = 400 mm. Estas posiciones se muestran usando la referencia "base", que toma como valor de referencia el centro de la herramienta (TCP).

Para facilitar la comprensión, podemos considerar nuestro propio brazo. La posición X – Y – Z de la punta de nuestro dedo índice representaría las coordenadas X-Y-Z de nuestro cuerpo, mientras que Rx, Ry y Rz se relacionaría con la rotación de nuestra muñeca.

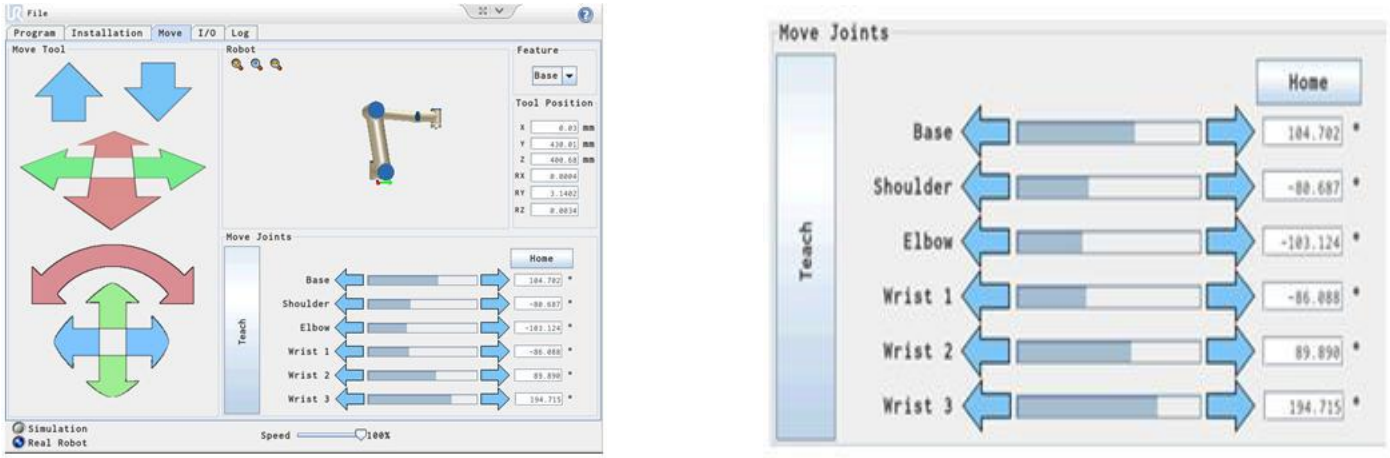


Figura 1.8 Interfaz de PolyScope para el posicionamiento del robot.

Esta parte de la pantalla muestra los ángulos de cada articulación (Joints). Esto no indica directamente donde se encuentra el TCP en el espacio. Nótese que los valores están expresados en grados. Cada articulación puede girar +360 grados y -360 grados, conformando un total de 720 grados.

Figura 1.9 Intefaz de Polyscope para el posicionamiento del TCP

Esta parte de la pantalla muestra la posición de la herramienta como un vector de 6 ejes, con la posición X-Y-Z expresadas en milímetros, y Rx-Ry-Rz expresadas en grados. Los ángulos de las juntas están representados en grados, pero también pueden estar representadas como radianes. Esto es de suma importancia, porque posteriormente cuando se programa un script en PolyScope, estos valores deben ser provistos al programa en radianes para que puedan ser procesados sin errores..

1.4 Software

1.4.1 Python.

Python es un lenguaje de programación de alto nivel, multipropósito y ampliamente usado. Fue creado a los principios de 1990, por Guido van Rossum, en *Stichting Mathematisch Centrum* en Holanda, como sucesor de un lenguaje llamado ABC. (18)

Su filosofía hace énfasis en la facilidad de lectura del código, y una sintaxis que ayude a los programadores a expresar conceptos complejos en pocas líneas de código, en comparación a otros lenguajes como C++ o Java.

Es un tipo de lenguaje multiparadigma, esto significa que en lugar de obligar al programador a elegir un estilo particular de programación, permite la utilización de varios estilos, incluyendo programación del tipo orientada a objetos, imperativa, funcional y procedural. Utiliza tipado dinámico, es decir que una variable puede tomar distintos valores de distinto tipo en distintos momentos. Es también un lenguaje de interpretado, es decir, es capaz de analizar y ejecutar otros programas. Los interpretes se diferencian de los compiladores o de los ensambladores en que mientras estos traducen un programa desde su descripción en un lenguaje de programación al código de máquina del sistema, los interpretes solo realizan la traducción a medida que sea necesaria, típicamente instrucción por instrucción, y normalmente no guardan el resultado de dicha traducción. (19)

Python administra su memoria mediante un conteo de referencias, que es una técnica para contabilizar las veces que un determinado recurso está siendo referido. Estos recursos, generalmente son bloques de memoria, y esta técnica permite establecer cuando no existe ninguna referencia a ese bloque y este puede ser liberado.

1.4.2 PyGaze

PyGaze es un paquete que incluye dentro de sí una gran variedad de otras libreras como por ejemplo PyGame, PsychoPy, pylink (Propia de los Eyetrackers de SRResearch), Tobii SDK, etc. Lo que ofrece es una librería uniforme con una sintaxis muy sencilla, y algunas funciones muy útiles para experimentos en tiempo real, como la detección de sacadas. (20)

Está dirigido tanto para usuarios sin experiencia en Python, como para programadores avanzados. Al funcionar en Python, es código abierto, lo cual es muy útil ya que permite poder ver y entender cómo funciona. El software propietario tiene el defecto que no se puede observar “bajo el capot”, por ende no entender realmente cuales son los procesos que se están realizando para obtener un resultado deseado.

Además posee una base de documentación completa con la descripción de cada función y como debe ser utilizada.

1.4.3 PolyScope

Polyscope es el lenguaje de programación propio del robot Universal Robots Ur-3. El robot puede ser programado mediante una interfaz gráfica (GUI), mediante un script y mediante C-API. Como cualquier otro lenguaje de programación, el mismo posee variables, clases, funciones, etc. Posee una lista de funciones pre-establecidas que por ejemplo sirven para monitorear entradas y salidas, temperatura, entre otras cosas. (21)

Este programa permite que usuarios con poca experiencia puedan programar el robot. En la mayoría de tareas, para programar se utiliza el panel táctil sin tener que teclear complicados comandos.

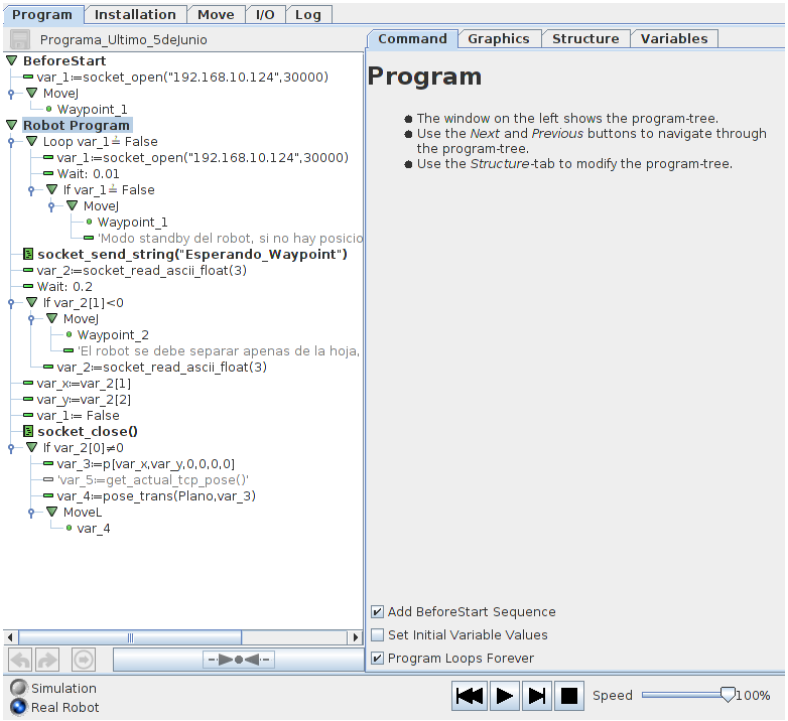


Figura 1.10 Pantalla principal de PolyScope

El movimiento es una parte importante del programa de un robot, por lo tanto resulta esencial poder enseñar al robot los movimientos que deseamos que realice. En PolyScope los movimientos de la herramienta se indican mediante puntos de paso (*Waypoints)*, es decir, puntos en el espacio de trabajo del robot. Estos puntos se pueden establecer moviendo al robot a la posición deseada, o calcularse mediante software. Se puede realizar tanto mediante el panel de movimiento, en donde uno controla la posición del robot presionando flechas en la pantalla táctil, o también presionando el botón de *Movimiento Libre*, el cual libera las juntas del robot permitiendo posicionar el robot en la posición deseada sin esfuerzo.

Además de realizar movimientos por pasos, también puede enviar señales de entrada y salidas, ejecutar comandos del tipo *while, if, loops y* muchas más.

2 - Pasos previos que nos llevaron a elegir el camino

2.1 Prueba de red.

El primer paso necesario para el desarrollo del software planteado, fue investigar acerca de las posibilidades de suministrar datos y controlar al robot UR-3 mediante una conexión de red. Se pudo obtener información y ejemplos sobre el control del robot mediante Ethernet, a partir del sitio web de un centro de soporte técnico acreditado de Universal Robots llamado "Zacobria Pte. Ltd". Entre varias aplicaciones, Zacobria utiliza el control vía red para realizar tareas de soporte a distancia.

A partir de la información obtenida se plantea realizar una prueba de conexionado Cliente-Servidor entre el Robot y una PC. Se realiza entonces un programa en la interfaz PolyScope donde se utilizará una conexión TCP vía Ethernet, en la cual el UR3 funciona como cliente solicitando posiciones a la PC host. El robot abrirá una conexión con la misma IP y PUERTO que el host, y solicitará a este un *string* con 3 valores: Indicador, X en pixeles, Y en pixeles. Una vez que recibe esos datos los procesa, y ubica en una variable que el robot puede procesar como la siguiente posición a la que se tendría que mover. Estas posiciones, o puntos de paso se denominan *Waypoints*.

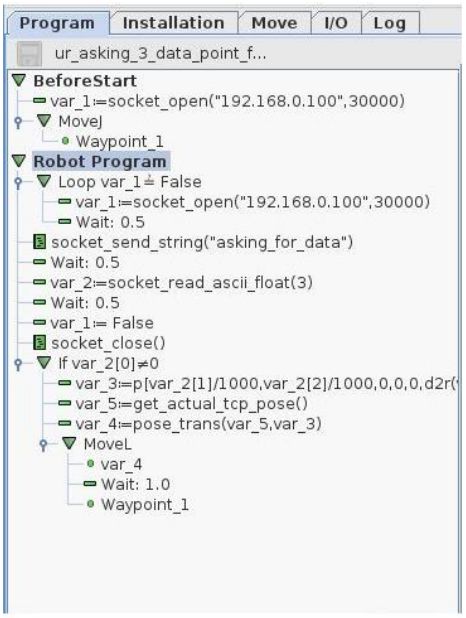


Figura 2.1 Código de PolyScope para pruebas de red.

2.2 Búsqueda de un lenguaje en común.

El próximo paso fue seleccionar un lenguaje de programación que permitiera comunicar ambos sistemas, ya que ambos tienen sus propios lenguajes. Durante el proceso de investigación acerca de las posibilidades de conexionado en red del UR3, se observó que la gran mayoría de ejemplos presentados por Zacobria involucraban un código desarrollado en el lenguaje Python, por ende se decidió investigar las posibilidades de dicho lenguaje.

Sabiendo que Python permite trabajar con el robot, restaría averiguar sobre sus posibilidades con el eyetracker. Luego de algo de investigación se pudo dar con una librería de Python llamada PyGaze, es un proyecto de código abierto que busca brindar una alternativa a MATLAb, como el lenguaje estándar utilizado en el eyetracking.

Una vez seleccionado Python como lenguaje de desarrollo, se deben realizar dos últimas pruebas. El funcionamiento de Python con el robot por un lado, y su funcionamiento con el eyetracker por otro.

Para el desarrollo de código en Python es necesario seleccionar una IDE, o interfaz de desarrollo, en este caso utilizaremos Spyder (Scientific Python Development EnviRoment). Spyder es una interfaz dinámica similar a MATLAb, que provee herramientas avanzadas de edicion, de pruebas y debugging entre otras cosas. Se encuentra incluida en el paquete WinPython-PyGaze.

Funcionamiento de Python con el robot.

Se escribirá un pequeño programa de Python en el cual se abrirá una conexión TCP con el robot y se esperará a que PolyScope le envíe un string con el mensaje ("*Esperando_Waypoint*") al que automáticamente el programa responderá con una posición X-Y, que posteriormente el robot procesará.

En el anexo se encontrará este código completo para realizar las pruebas necesarias, con el nombre de *Prueba Python-Ur3*. A continuación se explicará el funcionamiento y estructura de este programa.

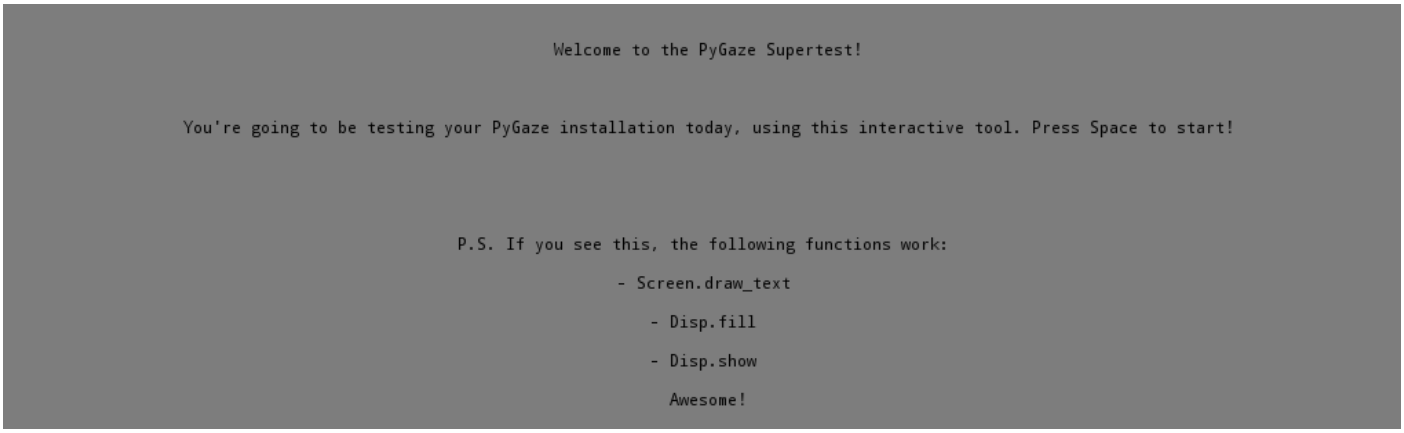
```
import socket
import time

HOST = '192.168.0.5'   # The remote host
PORT = 30000               # The same port as used by the server

print "Comenzando programa...."
count = 0

while (count < 1000):

      print "Entrando en el loop"
      s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
      s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

      s.bind((HOST, PORT))                    # Bind to the port
      s.listen(5)                             # Now wait for client connection.
      c, addr = s.accept()                    # Establish connection with client.

      try:
            msg = c.recv(1024)
            print msg
            time.sleep(1)
            if msg == "Esperando_Waypoint":
                      count = count + 1
                      #print "Contador en:", count
                      time.sleep(0.5)
                      print "MENSAJE ES" + msg
                      time.sleep(0.5)
                      #print "Posicion", xySTR
                      c.send("OK")

      except socket.error as socketerror:
            print count


c.close()
s.close()
```

Figura 2.2 Código de Python que se comunica con el robot, enviando una posición luego de recibir un string esperado.

Funcionamiento de Python con el eyetracker.

Para probar el funcionamiento de Python, específicamente de PyGaze, se hará uso de uno de los códigos de prueba incluidos en la librería. Es un código de 250 líneas que se encarga de probar todas las funciones de la librería y el eyetracker.

Este código se llama *PyGaze_trackertest*, y se encuentra en la carpeta *examples*.



Figuras 2.3 Pantalla principal del código Tracker Test, de la librería PyGaze

3- Desarrollo del software

Aclaraciones previas.

El programa resultante de este proyecto es de código libre, y en este capítulo se detallará su funcionamiento, tanto para su comprensión como para facilitar su reproducción. En el anexo se encontrará el código completo, así como también información general de los lenguajes de programación utilizados para facilitar su utilización.

3.1 Objetivo del software

El programa a desarrollar debe presentar un experimento de eyetracking en el cual se sucedan una serie de imágenes a definir, para obtener la posición donde se posa la mirada del sujeto experimental sobre dichas imágenes en pantalla, y tener la capacidad de procesar esa gran cantidad de información que suministra el eyetracker y filtrarla en lo que es de nuestro interés. Se necesita obtener posiciones X-Y de los ojos en Pixeles, para luego reescalar esos datos a milímetros en relación a un plano en el que se va a mover el robot.

Mediante el software podemos obtener fijaciones y sacadas por separado, pero para cumplir con el objetivo se optó por utilizar muestras (*samples*), es decir la posición del ojo en tiempo real al momento de ser solicitado, sin distinguir que tipo de movimiento es. Esta información la obtenemos solamente del ojo derecho.

3.2 Lo macro

Para entender el código desarrollado, podemos dividirlo en 3 grandes secciones. La primera sección se encargará de definir todas las variables necesarias para el desarrollo del experimento, directorios, ip y puerto del servidor, entre otras cosas.

La segunda parte consta de un condicional while, que ejecutará el código en su interior siempre que se cumplan las condiciones iniciales. Este código al interior del *while* es el encargado de todo el proceso de obtención de las miradas, el procesamiento de esos datos y el envío de los mismos al robot.

La tercer parte, es la que se encarga de finalizar el código. Entramos a esta sección una vez que se dejan de cumplir las condiciones iniciales del *while,* y procederemos a cerrar la instancia de eyetracker, y finalizar todos los dispositivos involucrados.

3.3 Lo micro

Inicio del software

Este programa en primera instancia lo que hace es convocar a otro archivo denominado constants donde están establecidas todas las configuraciones del experimento, siendo dichas configuraciones los tamaños de pantalla, ip y puerto por donde se comunicara el software con el robot y las medidas en milímetros de las hojas a utilizar.

Una vez cargado el archivo constants el próximo paso es que el experimento carga las librerías que necesita, por ejemplo la librería display que se encarga de la administración de la pantalla, keyboard que permite utilizar un teclado, la librería del eyetracker PyGaze. Todo esto es necesario porque por defecto Python no permite acceder a todos estos dispositivos, requieren si o si convocar estos módulos.

Inicialización del software

La segunda parte es el setup, donde una vez que se establecen las librerías que se necesitan se empiezan a utilizar, lo primero que se realiza es inicializar un display, generar la pantalla donde se va a llevar el experimento. Convocar un teclado, para leer teclas durante el experimento.

Una vez cargadas estos dispositivos necesarios, la pantalla y el teclado, empezamos a preparar el experimento. Lo primero que hacemos es cargar un archivo de instrucciones, en donde se le comenta al sujeto de que se tratará el experimento, y este archivo se cargará en el display que configuramos antes. (Python funciona asi, primero se deben crear las instancias y luego indicar en donde se van a mostrar). Luego se carga otro texto de instrucciones, y posteriormente convocar a la librería del teclado para que "escuche" si alguien presiona una tecla, para pasar a la siguiente función, que será tracker.calibrate() que es donde se convoca a la función de calibración del eyetracker.

En esta sección, se utiliza la función de calibración propia del eyetracker, de la computadora (host), es decir se convoca a otro programa en el cual no se tiene control, hasta que finalice el proceso de calibración.

Una vez finalizada la calibración, se realiza una limpieza del display. Esto se debe realizar siempre que queremos visualizar nuevo contenido puesto que de otra manera se solaparían los contenidos.

Se vuelven a indicar instrucciones y escuchar teclas, para pasar a la próxima función del programa, siendo esta tracker.drift_correction(), que se utiliza con el fin de asegurarse que el participante este mirando el centro de la pantalla antes de empezar el experimento.

Una vez asegurado esto, ingresamos al experimento. Todo lo anterior fue la preparación del mismo.

El Software

Previo a comenzar con el experimento, se deben inicializar una serie de contadores, que se encargarán de medir el tiempo transcurrido del experimento, para así poder cambiar las imágenes cada X cantidad de tiempo. Es la manera que tenemos de controlar las variables de tiempo en el experimento.

Una vez iniciados los contadores convocamos a la librería time, y se le indica la función *time.time()*, que empieza a contar el tiempo transcurrido desde que se convoca dicha función, como un cronometro. Luego se define un string donde se enumerarán los archivos de imágenes que queremos que se visualicen en el experimento. Al contar la cantidad de archivos en la carpeta "IMÁGENES" del experimento, determinara de cuantas vueltas constará el experimento.

Una vez listas todas las configuraciones, entramos en una función del tipo "*while*", que es un ciclo que se repetirá siempre y cuando se cumpla la condición inicial. La condición inicial es que el tiempo transcurrido sea menor al TRIALTIME, es decir al tiempo máximo establecido para cada imagen. (Esto está definido en el archivo constants) y también, que no se presione la tecla ESPACIO, que permite salir del experimento.

A continuación se convoca la función *tracker.start_recording()*, que recibe y guarda las posiciones de los ojos durante todo el experimento hasta que se convoque la función *tracker.stop_recording()*. También se inicializa el listen del teclado. Se cargan los archivos, y se limpia la pantalla.

Utilizamos el módulo OS de Python, que permite administrar y recorrer archivos del sistema. Cargamos las imágenes en el string que creamos previamente, y las presentamos en la pantalla. Posteriormente a esto convocamos la función *eyetracker.sample()*, que permite obtener en tiempo real la posición actual de los ojos en relación a las pantalla, son samples/muestras. Para convocar esta función es necesario utilizar si o si *eyetracker.start_recording()* previamente, por más que no nos interese guardar esta información. A esta función *eyetracker.sample()*, la asignamos a la variable "*gazepos*", la cual nos devuelve posiciones X-Y del ojo derecho.

Luego segmentamos estos valores, los separamos en X y en Y. El espacio 0 del string corresponde al valor X, y el espacio 1 al valor Y.

Luego pasamos a reescalar estos valores, porque las posiciones obtenidas en pantalla necesitamos trasladarlas a posiciones en milímetros en relación a un plano sobre el que trabajará el robot. En nuestro caso, en una hoja a4. (*En el anexo se encuentra un apartado donde se detalla todos los cálculos necesarios para el reescalado de los datos.)*

El resultado de estas conversiones de Pixel a Milimetros, los cargamos en dos variables (PosX y PosY), para luego enviarlas en un string al robot.

Etapa de Red, Envío de datos.

Empezamos creando un socket, para comunicar Python con el robot.

Los sockets se definen como un concepto abstracto en donde 2 aplicaciones interactúan entre sí a través de un protocolo para intercambiar datos. Los Sockets como tal los hay de varios tipos, pero básicamente hablamos de:

Sockets de flujo: Se caracteriza por utilizar el tipo de sockets SOCK_STREAM que usa como base el protocolo TCP (Transmission Control Protocol). En teoría asegura que los mensajes enviados a destino lleguen en el mismo orden en el que fueron enviados.

Sockets de datagrama: Este usa el tipo de sockets SOCK_DGRAM y es especial para trabajar con el protocolo UDP (User Datagram Protocol), a diferencia del anterior los mensajes pueden llegar en distinto orden en el que originalmente fueron enviados.

Una vez llamado el módulo socket, utilizamos la función bind, para asignar un puerto y una ip para la comunicación vía red (establecidas en el archivo constants.py).

Una vez establecidas la ip y puerto, utilizamos la función listen del módulo socket, que lo que va a realizar es una "escucha" sobre esa ip y puerto, esperando conexiones entrantes. Apenas se reciba una conexión, se la acepta y se avanza con el código.

Creamos una variable llamada msg, que va a llamar a la función recieve del módulo socket, de esta manera asignamos al mensaje recibido por la conexión entrante a la variable msg, para luego comparar si este mensaje recibido es el mensaje definido en común "esperando_waypoints", y de ser así, se envía la posición X-Y del ojo, luego de haberse realizado los cálculos.

Una vez enviadas las posiciones, volvemos a convocar a la función time para restar el tiempo actual del experimento con el tiempo de inicio del experimento, y comparamos si este tiempo es mayor o igual al tiempo previamente definido para cada imagen, de ser así, pasamos a la siguiente. Y cuando ya no hay más imágenes para presentar, terminamos el experimento.

Final del software.

Salimos del While y hacemos una limpieza del screen, y enviamos la función *tracker.stop_recording()*, la cual finaliza la grabación con el eyetracker, envía los archivos de información recolectados y cierra la conexión con el eyetracker.

Presentamos un texto para los participantes, donde se agradece la participación y esperamos que se presione cualquier tecla. Una vez presionada una tecla, salimos del experimento.

3.4 Descripción del código utilizado en el software.

Por razones didácticas en este capítulo se presentará el código en su versión final, segmentado, de forma que sea posible explicar tanto la estructura y diferentes niveles del código, como cada función utilizada.

El código en su versión final al día de la fecha consta de 2 archivos. El primer archivo es contants.py que posee 82 líneas de código en donde se declaran variables correspondientes a la configuración del experimento y experiment.py que posee 200 líneas en la que se reconocen 3 estructuras principales: Importación de librerías y declaración de variables; el trial donde se presentan las imágenes junto con la extracción y envío de datos; el cierre del experimento donde se guardan todos los datos y se cierran las instancias de eyetracking.

En el anexo se encontrará el código completo, con comentarios para facilitar su comprensión.

Comenzaremos explicando el primer archivo, constants.py.

```
import os.path


DIR = os.path.dirname(__file__)
DATADIR = os.path.join(DIR, 'data')
IMGDIR = os.path.join(DIR, 'images/track')
INSTFILE = os.path.join(DIR, 'instructions.txt')
LOGFILENAME = "Pruebas" #input("Nombre del sujeto:  ")
LOGFILE = os.path.join(DATADIR, LOGFILENAME)
```

En esta sección se definen todos los directorios en relación al experimento.

- **DIR**: Con tiene el directorio donde están todos los archivos relacionados al experimento. Es el directorio RAIZ.
- **DATADIR**: Es el directorio donde se guardaran los archivos .EDF que se extraigan al finalizar el experimento.
- **IMGDIR**: Es el directorio donde se encuentran las imágenes a utilizar en el experimento, en formato JPEG.

- **INSTFILE**: Es un archivo .TXT donde se puede introducir un pequeño texto introductorio explicando en qué consistirá el experimento.
- **LOGFILENAME**: Determinará el nombre de archivo con el que se guardarán los resultados del experimento. Es una función que abre un cuadro de texto, donde el sujeto experimental debe ingresar su nombre.
- **LOGFILE**: Crea el archivo de logeo, con el nombre resultante de LOGFILENAME en el directorio DATADIR.

```
DISPTYPE = 'psychopy'
DISPSIZE = (1024,768)
SCREENSIZE = (532.0,299.0)
SCREENDIST = 60.0
FULLSCREEN = False
BGC = (0,0,0)
FGC = (255,255,255)
TEXTSIZE = 24
```

Configuración del display:

- **DISPTYPE**: En esta variable se elige el motor gráfico que va a renderizar el experimento. Existen dos opciones: Psychopy y Pygame. Psychopy tiene mejor resolución de tiempo en ms, y por otro lado, pygame utiliza menos recursos.
- **DISPSIZE**: Resolución del experimento. Por ejemplo, 1920x1080.
- **SCREENSIZE**: Tamaño físico de la pantalla donde correrá el experimento, en centímetros.
- **SCREENDIST**: Distancia en centímetros entre la pantalla y el sujeto experimental.
- **FULLSCREEN**: Determina si el experimento se ejecutará en pantalla completa o ventana. True = Pantalla Completa False = en Ventana.
- **BGC**: Background Color. Determina el color de fondo del experimento.
- **FGC**: Foreground Color. Determina el color frontal del experimento.
- **TEXTSIZE**: Determina el tamaño de texto a utilizar en el experimento.

```
TRIALTIME = 5 # segundos
ITI = 2000 # ms
```

Variables de tiempo:

- **TRIALTIME**: Tiempo de duración por imagen en el experimento.
- **ITI**: Tiempo de intervalo, de uso general.

```
TRACKERTYPE = 'dummy'
EYELINKCALBEEP = True
DUMMYMODE = True
```

Configuración del eyetracker:

- **TRACKERTYPE**: Selecciona el tipo de eye tracker que se utilizará en el experimento. Puede ser 'eyelink','tobii', etc. Si no se está utilizando un eye tracker, se debe escribir 'dummy'.
- **EYELINKCALBEEP**: Activa o desactiva el beep al momento de calibrar. True= Activa False= Desactiva.
- **DUMMYMODE**: Tiene que activarse si no se está usando un eyetracker. True=Activa el modo dummy. False = Si se está usando un eyetracker.

```
HOST = '192.168.10.124'
PORT = 30000
```

Configuración de IP y PUERTO:

- **HOST**: La IP de la PC HOST, en la que se corre el experimento.
- **PORT**: El puerto a utilizar, puede ser cualquiera, pero debe coincidir tanto en el robot como en la pc que corra el experimento.

```
X_pantalla_mm = 405.0
Y_pantalla_mm = 230.0
X_a4_mm = 210.0
Y_a4_mm = 245.0
```

- Valores en mm de las pantallas:
- **X_pantalla_mm**: Medida en mm de la pantalla, eje X.
- **Y_pantalla_mm**: Medida en mm de la pantalla, eje Y.
- **X_a4_mm**: Medida en mm de una hoja tipo Carta, en eje X, donde el Robot Ur3 dibujará.
- **Y_a4_mm**: Medida en mm de una hoja tipo Carta, en eje Y, donde el Robot Ur3 dibujará.
- A continuación se explicara línea por línea el segundo archivo, ***experiment.py***, que contiene todas las funciones y variables que hacen funcionar el experimento.

Primer parte del experimento:

```
import os
import socket
import time

from constants import *

from pygaze.libscreen import Display, Screen
from pygaze.libinput import Keyboard
from pygaze.eyetracker import EyeTracker
from pygaze.liblog import Logfile
import pygaze.libtime as timer
```

En esta sección se importan todas las librerías necesarias para el funcionamiento del experimento.

- **Import OS, SOCKET y TIME: OS** es una librería que permite administrar directorios y rutas de archivo, para incluirlos en el experimento. **SOCKET** es la librería que permite establecer una conexión de red con un servidor externo. **TIME** es una librería que permite administrar el tiempo en el experimento, por ejemplo utilizando contadores.
- **from constants import *:** Aquí se importa dentro del experimento, mediante el símbolo "*", todo el contenido dentro del archivo constants.py.
- **from pygaze.xxx import.xxxx:** Se encarga de importer de la librería PYGaze, funciones específicas.

```
disp = Display()
scr = Screen()

kb = Keyboard()
kb = Keyboard(keylist=['space'],timeout=None)
tracker = EyeTracker(disp)
```

- **disp= Display():** Vincula la variable disp con la función Display(), encargada de la configuración de la pantalla en la que se desarrollará el experimento.
- **scr=Screen():** Vincula la variable scr con la función Screen(), encargada de presentar una pantalla en la cual se hará un "fill", es decir se la llenará de contenido. Imágenes, Textos,etc.
- **kb=Keyboard():** Vincula la variable kb con la función Keyboard(), la cual permite tener control absoluto del teclado mientras se ejecuta en experimento.

- **kb = Keyboard(keylist=['space'],timeout=None):** Aquí definimos que teclas habilitaremos para el experimento, en este caso solo 'space', mediante la definición "keylist". En "timeout=None", establecemos que no devuelva ningún valor a excepción de que se aprete la tecla establecida.

- **tracker= EyeTracker(disp):** Vincula la variable tracker con la función Eyetracker(). Esta función permite establecer la conexión con el eye tracker y definir su configuración básica.

Estructura de la función Eye Tracker:

EyeTracker(***display***[Requiere ser definido, en nuestro caso es la variable **disp**], ***trackertype***=TRACKERTYPE, ***resolution***=DISPSIZE, ***eyedatafile***=LOGFILENAME, ***logfile***=LOGFILE, ***fgc***=FGC, ***bgc***=BGC, ***saccvelthresh***=SACCVELTHRESH, ***saccaccthresh***=SACCACCTHRESH, ***ip***=SMIIP, ***sendport***=SMISENDPORT, ***receiveport***=SMIRECEIVEPORT)

```
instfile = open(INSTFILE)
instructions = instfile.read()
instfile.close()
```

- **instfile=open(INSTFILE):** Vincula la variable instfile con la función "open", para abrir un archivo. En este caso, el archivo de texto que contiene las instrucciones.

- **Instructions=instfile.read():** Vincula la variable Instructions con instfile, vinculado a la función read(). **Instfile.close()**, cierra el archivo.

```
images = os.listdir(IMGDIR)
```

- **Images=os.listdir(IMGDIR):** Vincula la variable Images con la función os.listdir(IMGDIR), que se encarga de leer el directorio indicado "IMGDIR" y devolver la cantidad de archivos que se encuentren en ese directorio.

```
scr.draw_text(text="Presiona cualquier tecla para comenzar la calibracion.", fontsize=TEXTSIZE)
disp.fill(scr)
disp.show()
```

- **Scr.draw_text(text="Presiona cualquier Tecla para comenzar la calibracion", fontsize=TEXTSIZE):** Dibuja un texto en la screen seleccionada "scr", el texto se indica en el parámetro "text", y fontsize permite determinar el tamaño de texto.

- **Disp.fill(scr):** La función fill se encarga de rellenar el display seleccionado "disp" con el contenido que indiquemos entre paréntesis "scr"

- **Disp.show():** Esta función se encarga de visualizar el display seleccionado.

```
kb.get_key(keylist=None, timeout=None, flush=True)
```

- **Kb.get_key(keylist=None, timeout=None, flush=True):** Aquí permitimos la lectura de cualquier tecla.

```
tracker.calibrate()
```

- **Tracker.calibrate()=** Enviamos la función calibrate() al eyetracker, haciendo que el eyetracker en el modo de calibración.

```
scr.clear()
scr.draw_text(text=instructions, fontsize=TEXTSIZE)
disp.fill(scr)
disp.show()
```

- **scr.clear():** La función clear(), limpia la pantalla que se le indique. En este caso la pantalla ligada a la variable scr.

```
kb.get_key(keylist=None, timeout=None, flush=True)
tracker.drift_correction()
```

- **tracker.drift_correction():** Se le indica al Eyetracker que ejecute la función drift_correction(), la cual realiza un checkeo de correlación entre la posición del ojo y la fijación en pantalla previo a ejecutar el experimento.

```
key = None
trialnr = 0
count = 0
vueltas = 0
```

- **key = None , trailnr = 0, count = 0, vueltas = 0:** Declaramos estas variables, y le asignamos el valor 0. Son contadores que luego vamos a utilizar para saber en qué etapa se encuentra el experimento.

```
start = time.time()
time.clock()
elapsed = 0
```

- **start = time.time():** Start llama a la función time de la librería time. Se encarga de consultar el tiempo al momento de convocar la función, para luego usarla como comparador.
- **time.clock():** Devuelve el tiempo en segundos, desde que se llama a la función. Es el tiempo del procesador.

```
Imagenes = [2,4,2]
ntrials = len(Imagenes)
X = 0
```

- **Imagenes = [2,4,2]:** Es un string en donde elegimos que archivos de imagen utilizaremos en el experimento
- **ntrials = len(Imágenes):** Determina la cantidad de imágenes que veremos en el experimento, en función de los archivos que elegimos mostrar.
- **X = 0:** Es un puntero encargado de recorrer el string de imágenes. Avanzará un valor con cada vuelta del trial.

Segunda parte del código:

Esta es la sección donde comenzamos a visualizar las imágenes elegidas para el experimento, es una función while que se repetirá siempre y cuando se cumplan las condiciones definidas previamente. En este caso, que el tiempo transcurrido del experimento sea menor al tiempo total de experimento que definimos previamente.

```
while elapsed < TRIALTIME and not key == 'space':
```

Todo el código que sigue, se ejecutará siempre y cuando se cumplan las condiciones establecidas en el while.

```
tracker.start_recording()
key, presstime = kb.get_key(timeout=1)
```

- **Tracker.start_recording():** Indicamos al eyetracker, que comenzaremos a grabar las posiciones de los ojos. Es necesario ejecutarlo para luego utilizar la función ***tracker.sample().***

```
Carga = Imagenes[X]
scr.clear()
scr.draw_image(os.path.join(IMGDIR,images[Carga]))
disp.show()
```

- **Carga = Imágenes[X] =** Con carga elegimos que archivo de video ejecutar en cada vuelta.
- **Scr.draw_image(os.path.join(IMGDIR,images[Carga])):** Presentamos la imagen seleccionada. Convocamos la función **path.join** de la librería **os**. Le indicamos que cargue archivos de la carpeta IMGDIR, el archivo lo elegirá en función del valor de la variable Carga.

```
gazepos = tracker.sample()
```

- **Gazepos = tracker.sample():** Vinculamos la variable gazepos a la función sample(), del módulo eyetracker. Esta función nos devuelve posiciones X-Y del ojo derecho en tiempo real, son muestras.

```
scr.draw_fixation(fixtype='dot', pos=gazepos, pw=5, diameter=15)
```

- **Scr.draw_fixation(fixtype='dot', pos=gazepos, pw=5, diameter=15):** Función que se encarga de graficar las posiciones obtenidas por el eyetracker. "fixtype" indica que lo graficamos como un punto, "pos" es la variable a la cual se le debe suministrar una posición X-Y para graficar la fijación. "pw y diameter", permiten a justar el tamaño de la fijación.

```
disp.fill(scr)
gazeposSTR = str(gazepos)
```

- **gazeposSTR = str(gazepos):** Variable con la cual transformamos los valores de gazepos en un string.

```
gazeposX = gazepos[0]
gazeposY = gazepos[1]
FactorX = (X_a4_mm / X_pantalla_mm)
FactorY = (Y_a4_mm / Y_pantalla_mm)
PX_mm_X = (X_pantalla_mm / DISPSIZE[0])
PX_mm_Y = (Y_pantalla_mm / DISPSIZE[1])
PosX = (PX_mm_X * (gazeposX * FactorX))
PosY = (PX_mm_Y * (gazeposY * FactorY))
PosFinalTuple = (PosX/1000,PosY/1000)
PosFinalSTR = str(PosFinalTuple)
```

- En la sección de arriba se encuentran todos los calculos necesarios para el reescalado de Pixeles a Milimetros.

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

- Creamos el socket, y le asignamos que pertenece a la familia AF_INET y del tipo SOCK_STREAM.

```
s.bind((HOST, PORT))
```

- **s.bind((HOST,PORT)):** Asigna al servidor que estamos creando, la IP y PUERTO establecidas en el archivo constants.py.

```
s.listen(1)
```

- **s.listen(1):** Asigna cuantas conexiones el servidor puede permitir en espera.

```
c, addr = s.accept()
```

- **c, addr = s.accept():** Esta función se encarga de aceptar las conexiones entrantes.

```
msg = c.recv(1024)
```

- **msg = c.recv(1024):** Recibe data del socket creado. El valor obtenido se representa como un string, y la máxima cantidad de data a ser recibida se especifica en el bufzsize, en este caso 1024. Para obtener buenas respuestas con la mayoría de los hardware y redes, el valor del buffer debería ser una potencia de 2, por ejemplo 4096.

```
time.sleep(0.001):
```

- **time.sleep(0.001):** Actúa como un pequeño delay, en el orden de las milésimas de segundo, para no saturar la red.

```
if msg == "Esperando_Waypoint":
```

- **if msg == "Esperando_Waypoint":** Condicional que fine el mensaje que espera el servidor, a partir del cual, puede responder con las posiciones.

```
c.send(PosFinalSTR);
```

- **c.send(PosFinalSTR);:** Al cumplirse el condicional, es decir se obtiene el mensaje esperado, se responde al robot con las posiciones ya escaladas.

```
elapsed = time.time() - start
if elapsed >= TRIALTIME:
X = X + 1
TRIALTIME = TRIALTIME + 10
timer.pause(ITI)
```

- En esta sección se encuentran los contadores que permiten saber cuándo hay que pasar a la siguiente imagen. Contabilizamos el tiempo transcurrido, y si es mayor al tiempo que establecimos por imagen, avanzamos a la siguiente.

```
if X == ntrials:
break
```

- Si ya cargamos todas las imagenes disponibles, interrumpimos y salimos del while.

```
tracker.stop_recording()
```

- Frenamos la obtención de muestras por parte del eyetracker.

```
scr.clear()
scr.draw_text(text="Gracias por participar.", fontsize=TEXTSIZE)
disp.fill(scr)
disp.show()
timer.pause(1000)
scr.clear()
scr.draw_text(text="Transfiriendo data files... Espere.", fontsize=TEXTSIZE)
disp.fill(scr)
disp.show()
timer.pause(1)
```

- Limpiamos la pantalla y enviamos un mensaje al usuario.

```
tracker.close()
```

- Finalmente cerramos la conexión con el eyetracker.

```
scr.clear()
scr.draw_text(text="Este es el final de el experimento. Gracias por participar! !\n\n(Presione c
ualquier tecla para salir)", fontsize=TEXTSIZE)
disp.fill(scr)
disp.show()
kb.get_key(keylist=None, timeout=None, flush=True)
disp.close()
```

- Enviamos un mensaje final, y esperamos que se presione cualquier tecla para cerrar el experimento.

Fin del código.

Apéndice

Códigos del software.

Ambos códigos son de uso y se pueden encontrar actualizados en el repositorio de github correspondiente al proyecto, que se encuentra en: https://github.com/nachofourmentel/Python_Eyetracker_UR3.

En dicho repositorio encontraremos las siguientes carpetas:

- ***Eyetracker_Ur3_Realtime_Final_CALIBRACION:*** La cual contiene el software con el que podemos calibrar los pixeles de pantalla a los mm correspondientes al plano sobre el que trabajará el robot.
- ***Eyetracker_Ur3_Realtime_Final_Imagenes***: Software en el cual se presentarán una serie de imágenes, para ser observadas por el participante.
- ***Eyetracker_Ur3_Realtime_Final_Texto:*** Software en el cual se presenta un texto para ser leído por el participante.

Última versión del código impresa en la fecha del 22/11/2017

EXPERIMENT.PY

```
# Experimento de analisis de la gestualidad del ojo humano en relacion a un robot industrial colaborativo.
# Ignacio Fourmentel, Universidad Nacional de Tres de Febrero.
# contact: ignacio.fourmentel@gmail.com
# ULTIMA Version 8 de Noviembre 2017


import os
import socket
import time


from constants import *


from pygaze.libscreen import Display, Screen
from pygaze.libinput import Keyboard
from pygaze.eyetracker import EyeTracker
from pygaze.liblog import Logfile
import pygaze.libtime as timer


# # # # #
# SETUP
# visuals
disp = Display()
scr = Screen()
```

```
# input
kb = Keyboard()
kb = Keyboard(keylist=['space'],timeout=None)
tracker = EyeTracker(disp)

# # # # #
# Preparacion del expermiento
# Cargamos las instrucciones desde un archivo.
instfile = open(INSTFILE)
instructions = instfile.read()
instfile.close()

# Leemos el directorio de las imagenes.
images = os.listdir(IMGDIR)

# Instrucciones.
scr.draw_text(text="Presiona cualquier tecla para comenzar la calibracion.", fontsize=TEXTSIZE)
disp.fill(scr)
disp.show()

# Esperamos la tecla.
kb.get_key(keylist=None, timeout=None, flush=True)

# Etapa de Calibracion
tracker.calibrate()

# # # # # # # # # # # # #
# Comienza el experimento.
# Display de instrucciones.
scr.clear()
scr.draw_text(text=instructions, fontsize=TEXTSIZE)
disp.fill(scr)
disp.show()

# Esperamos tecla puede ser CUALQUIERA
kb.get_key(keylist=None, timeout=None, flush=True)

# drift check se puede sacar a posteiori.
tracker.drift_correction()



#Variables para controlar los trials
key = None
```

```
trialnr = 0
count = 0
vueltas = 0

# CORRE EL TRIAL
# Comenzamos el recording

#SETEO EL TIEMPO DEL EXPERIMENTO
#timepo transcurrido desde que se llama la variable start
start = time.time()
time.clock()
elapsed = 0

# PREPARAMOS IMAGENES
#ELEGIMOS LAS IMAGENES
Imagenes = [2,4,2]
# Numero de TRIALS en funcion de cantidad de imagenes.
ntrials = len(Imagenes)
#PUNTERO QUE RECORRE LA LISTA
X = 0

while elapsed < TRIALTIME and not key == 'space':
#tracker.start_recording()
key, presstime = kb.get_key(timeout=1)
#Cargamos archivos
Carga = Imagenes[X]
# presentamos imagen.
scr.clear()
scr.draw_image(os.path.join(IMGDIR,images[Carga]))
disp.show()
#SAMPLES
gazepos = tracker.sample()
# Presentamos fijaciones
scr.draw_fixation(fixtype='dot', pos=gazepos, pw=5, diameter=15)
disp.fill(scr)
gazeposSTR = str(gazepos)
gazeposX = gazepos[0]
gazeposY = gazepos[1]
FactorX = (X_a4_mm / X_pantalla_mm)
FactorY = (Y_a4_mm / Y_pantalla_mm)
PX_mm_X = (X_pantalla_mm / DISPSIZE[0])
PX_mm_Y = (Y_pantalla_mm / DISPSIZE[1])
#CUENTA FINAL
PosX = (PX_mm_X * (gazeposX * FactorX))
```

```
PosY = (PX_mm_Y * (gazeposY * FactorY))
PosFinalTuple = (PosX/1000,PosY/1000)
PosFinalSTR = str(PosFinalTuple)
print PosFinalSTR
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #definir el socket
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1) #lo mismo
s.bind((HOST, PORT))
s.listen(1) # Cuantas conexiones puede dejar en espera.
c, addr = s.accept() # Aceptar conexion
msg = c.recv(1024) # Buffer recibir mensajes 1024
time.sleep(0.001) #0.001
if msg == "Esperando_Waypoint":
c.send(PosFinalSTR); #ENVIO la posicion en string
elapsed = time.time() - start
if elapsed >= TRIALTIME:
X = X + 1
TRIALTIME = TRIALTIME + 10
timer.pause(ITI)

if X == ntrials:
break
# Limpiamos el screen
disp.fill()
t1 = disp.show()
#tracker.log("image offline at %d" % t1)

# Frenamos el recording del eyetracker.
#tracker.stop_recording()
# FIN TRIAL

# # # # #
# CIERRE
# MENSAJE..
scr.clear()
scr.draw_text(text="Gracias por participar.", fontsize=TEXTSIZE)
disp.fill(scr)
disp.show()
timer.pause(1000)
scr.clear()
scr.draw_text(text="Transfiriendo data files... Espere.", fontsize=TEXTSIZE)
disp.fill(scr)
disp.show()
timer.pause(1)
```

```
# cerramos conexion con el tracker

# enviamos los datos a la pc.

tracker.close()


# Mensaje de salida.

scr.clear()

scr.draw_text(text="Este es el final de el experimento. Gracias por participar! !\n\n(Presione cualquier tecl
a para salir)", fontsize=TEXTSIZE)

disp.fill(scr)

disp.show()


# Esperamos tecla..

kb.get_key(keylist=None, timeout=None, flush=True)


# Cierra display

disp.close()
```

CONSTANTS.PY

```
# Experimento de analisis de la gestualidad del ojo humano en relacion a un robot industrial colaborativo.

# Ignacio Fourmentel, Universidad Nacional de Tres de Febrero.

# contact: ignacio.fourmentel@gmail.com

# ULTIMA Version 8 de Noviembre 2017


import os.path


#DEFINICION DE DIRECTORIOS

#DIR contiene el path del directorio donde estan todos los archivos.

DIR = os.path.dirname(__file__)

#Directorio de archivos de DATA

DATADIR = os.path.join(DIR, 'data')

#IMGDIR directorio de archivos de imagenees

IMGDIR = os.path.join(DIR, 'images/track')

# INSTFILE archivo de instrucciones.

INSTFILE = os.path.join(DIR, 'instructions.txt')

# NOMBRE DE ARCHIVO = NOMBRE DE PARTICIPANTE.

LOGFILENAME = "Pruebas" #input("Nombre del sujeto: ")

# Crea el archivo de log con ese nombre.

LOGFILE = os.path.join(DATADIR, LOGFILENAME)


#DISPLAY

#Display 'pygame' o 'psychopy';

#psychopy tiene mejor resolucion de tiempo en ms, pygame usa menos recursos.

DISPTYPE = 'psychopy'
```

```
#Resolucion LABO (1366,768) BENQ (1080,1920) PORTRAIT
DISPSIZE = (1024,768)

#Tamano fisico de la pantalla. Centimetros.
SCREENSIZE = (532.0,299.0)

#Distancia en cm entre pantalla y el sujeto experimental.
#SCREENDIST = 60.0

#FULLSCREEN
FULLSCREEN = False

#COLORES background y foreground. RGB.
BGC = (0,0,0)
FGC = (255,255,255)

#Tamano de texto.
TEXTSIZE = 24

#VARIABLES DE TIEMPO
#Tiempo por imagen
TRIALTIME = 5 # segundos
#Tiempo entre imagenes. intervalo
ITI = 2000 # ms

#EYE TRACKING
#Tipo de eyetracker. 'eyelink' 'dummy'
TRACKERTYPE = 'dummy'
#Beep durante calibracion.
EYELINKCALBEEP = True
#Dummymode cuando no hay eyetracker conectado
DUMMYMODE = True

#SETUP HOST
#IP y Puerto de la PC HOST donde se corre el experimento.
HOST = '192.168.0.5'
PORT = 30000

#Datos de pantalla
#Datos del display del eyetracker.

#Pantalla del labo
X_pantalla_mm = 405.0
Y_pantalla_mm = 230.0
```

```
#Pantalla BENQ
X_pantalla_mm = 299.0
Y_pantalla_mm = 532.0

#Datos de Hoja A4
X_a4_mm = 210.0
Y_a4_mm = 245.0

#Datos de Hoja A3
X_a3_mm = 297.0
Y_a3_mm = 420.0
```

Anexo

Programación en Python

En este apartado se explicará nociones básicas acerca de cómo programar en Python, la siguiente información fue extraída y compilada de la documentación oficial de Python.

Modo interactivo

El intérprete de Python estándar incluye un modo interactivo en el cual se escriben las instrucciones en una especie de intérprete de comandos: las expresiones pueden ser introducidas una a una, pudiendo verse el resultado de su evaluación inmediatamente, lo que da la posibilidad de probar porciones de código en el modo interactivo antes de integrarlo como parte de un programa. Esto resulta útil tanto para las personas que se están familiarizando con el lenguaje como para los programadores más avanzados.

```
>>> 1 + 1
2
>>> a = range(10)
>>> print a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Para entender el funcionamiento de Python empezaremos con lo más básico, cálculos matemáticos en el interpretador, y seguiremos con cádenas de caracteres y listas.

Números

El intérprete actúa como una simple calculadora, se puede ingresar una expresión y el interpreté responderá con los valores. La sintaxis es sencilla: los operadores `+`, `-`, `*` y `/` funcionan como en la mayoría de los lenguajes (por ejemplo, Pascal o C); los paréntesis (`()`) pueden ser usados para agrupar. Por ejemplo:

```
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
>>> 8 / 5  # la división siempre retorna un número de punto flotante
1.6
```

Los números enteros (por ejemplo `2`, `4`, `20`) son de tipo **int**, aquellos con una parte fraccional (por ejemplo `5.0`, `1.6`) son de tipo **float**.

La división (`/`) siempre retorna un punto flotante. Para hacer ***floor division*** y obtener un resultado entero (descartando cualquier resultado fraccional) podés usar el operador `//`; para calcular el resto se puede usar `%`:

```
>>> 17 / 3  # la división clásica retorna un punto flotante
5.666666666666667
>>>
>>> 17 // 3  # la división entera descarta la parte fraccional
5
>>> 17 % 3  # el operado % retorna el resto de la división
2
>>> 5 * 3 + 2  # resultado * divisor + resto
17
```

Con Python, es posible usar el operador `**` para calcular potencias:

```
>>> 5 ** 2  # 5 al cuadrado
25
>>> 2 ** 7  # 2 a la potencia de 7
128
```

El signo igual (`=`) es usado para asignar un valor a una variable:

```
>>> ancho = 20
>>> largo = 5 * 9
>>> ancho * largo
900
```

Si una variable no está “definida” (con un valor asignado), intentar usarla producirá un error:

```
>>> n  # tratamos de acceder a una variable no definida
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'n' is not defined
```

En el modo interactivo, la última expresión impresa es asignada a la variable `_`. Esto significa que cuando estés usando Python como una calculadora de escritorio, es más fácil seguir calculando, por ejemplo:

```
>>> impuesto = 12.5 / 100
>>> precio = 100.50
>>> precio * impuesto
12.5625
>>> precio + _
113.0625
>>> round(_, 2)
113.06
```

Esta variable debería ser tratada como de sólo lectura por el usuario. No le asignes explícitamente un valor; crearás una variable local independiente con el mismo nombre enmascarando la variable con el comportamiento mágico.

Además de **int** y **float**, Python soporta otros tipos de números, como ser **Decimal** y **Fraction**. Python también tiene soporte integrado para *números complejos*, y usa el sufijo `j` o `J` para indicar la parte imaginaria (por ejemplo `3+5j`).

Cadenas de caracteres

Además de números, Python puede manipular cadenas de texto, las cuales pueden ser expresadas de distintas formas. Pueden estar encerradas en comillas simples (`'...'`) o dobles (`"..."`) con el mismo resultado . `\` puede ser usado para escapar comillas:

```
>>> 'huevos y pan'  # comillas simples
'huevos y pan'
>>> 'doesn\'t'  # usa \' para escapar comillas simples...
"doesn't"
>>> "doesn't"  # ...o de lo contrario usa comillas doblas
"doesn't"
>>> '"Si," le dijo.'
'"Si," le dijo.'
>>> "\"Si,\" le dijo."
'"Si," le dijo.'
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'
```

En el intérprete interactivo, la salida de cadenas está encerrada en comillas y los caracteres especiales son escapados con barras invertidas. Aunque esto a veces luzca diferente de la entrada (las comillas que encierran pueden cambiar), las dos cadenas son equivalentes. La cadena se encierra en comillas dobles si la cadena contiene una comilla simple y ninguna doble, de lo contrario es encerrada en comillas simples. La función **print()** produce una salida más legible,

omitiendo las comillas que la encierran e imprimiendo caracteres especiales y escapados:

```
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'
>>> print('"Isn\'t," she said.')
"Isn't," she said.
>>> s = 'Primerea línea.\nSegunda línea.'  # \n significa nueva línea
>>> s  # sin print(), \n es incluído en la salida
'Primera línea.\nSegunda línea.'
>>> print(s)  # con print(), \n produce una nueva línea
Primera línea.
Segunda línea.
```

Si no queremos que los caracteres antepuestos por `\` sean interpretados como caracteres especiales, podés usar *cadenas crudas* agregando una `r` antes de la primera comilla:

```
>>> print('C:\algun\nombre')  # aquí \n significa nueva línea!
C:\algun
ombre
>>> print(r'C:\algun\nombre')  # nota la r antes de la comilla
C:\algun\nombre
```

Las cadenas de texto pueden ser concatenadas (pegadas juntas) con el operador `+` y repetidas con `*`:

```
>>> # 3 veces 'un', seguido de 'ium'
>>> 3 * 'un' + 'ium'
'unununium'
```

Dos o más *cadenas literales* (aquellas encerradas entre comillas) una al lado de la otra son automáticamente concatenadas:

```
>>> 'Py' 'thon'
'Python'
```

Esto solo funciona con dos literales, no con variables ni expresiones:

```
>>> prefix = 'Py'
>>> prefix 'thon'  # no se puede concatenar una variable y una cadena literal
  ...
SyntaxError: invalid syntax
```

```
>>> ('un' * 3) 'ium'
  ...
SyntaxError: invalid syntax
```

Si queremos concatenar variables o una variable con un literal, usá `+`:

```
>>> prefix + 'thon'
'Python'
```

Esta característica es particularmente útil cuando queremos separar cadenadas largas:

```
>>> texto = ('Poné muchas cadenas dentro de paréntesis '
...          'para que ellas sean unidas juntas.')
>>> texto
'Poné muchas cadenas dentro de paréntesis para que ellas sean unidas juntas.'
```

Las cadenas de texto se pueden *indexar* (subíndices), el primer carácter de la cadena tiene el índice 0. No hay un tipo de dato para los caracteres; un carácter es simplemente una cadena de longitud uno:

```
>>> palabra = 'Python'
>>> palabra[0]  # caracter en la posición 0
'P'
>>> palabra[5]  # caracter en la posición 5
'n'
```

Los índices quizás sean números negativos, para empezar a contar desde la derecha:

```
>>> palabra[-1]  # último caracter
'n'
>>> palabra[-2]  # ante último caracter
'o'
>>> palabra[-6]
'P'
```

Nota que -0 es lo mismo que 0, los índices negativos comienzan desde -1.

Además de los índices, las *rebanadas* también están soportadas. Mientras que los índices son usados para obtener caracteres individuales, las *rebanadas* te permiten obtener sub-cadenas:

```
>>> palabra[0:2]  # caracteres desde la posición 0 (incluída) hasta la 2 (excluída)
'Py'
```

```
>>> palabra[2:5]  # caracteres desde la posición 2 (incluída) hasta la 5 (excluída)
'tho'
```

Nota como el primero es siempre incluído, y que el último es siempre excluído. Esto asegura que `s[:i] + s[i:]` siempre sea igual a `s`:

```
>>> palabra[:2] + palabra[2:]
'Python'
>>> palabra[:4] + palabra[4:]
'Python'
```

Los índices de las rebanadas tienen valores por defecto útiles; el valor por defecto para el primer índice es cero, el valor por defecto para el segundo índice es la longitud de la cadena a rebanar.

```
>>> palabra[:2]  # caracteres desde el principio hasta la posición 2 (excluída)
'Py'
>>> palabra[4:]  # caracterrs desde la posición 4 (incluída) hasta el final
'on'
>>> palabra[-2:] # caracteres desde la ante-última (incluída) hasta el final
'on'
```

Una forma de recordar cómo funcionan las rebanadas es pensar en los índices como puntos *entre* caracteres, con el punto a la izquierda del primer carácter numerado en 0. Luego, el punto a la derecha del último carácter de una cadena de *n* caracteres tiene índice *n*, por ejemplo:

```
 +---+---+---+---+---+---+
 | P | y | t | h | o | n |
 +---+---+---+---+---+---+
 0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

La primer fila de números da la posición de los índices 0...6 en la cadena; la segunda fila da los correspondientes índices negativos. La rebanada de *i* a *j* consiste en todos los caracteres entre los puntos etiquetados *i* y *j*, respectivamente.

Listas

Python tiene varios tipos de datos *compuestos*, usados para agrupar otros valores. El más versátil es la *lista*, la cual puede ser escrita como una lista de valores separados por coma (ítems) entre corchetes. Las listas pueden contener ítems de diferentes tipos, pero usualmente los ítems son del mismo tipo:

```
>>> cuadrados = [1, 4, 9, 16, 25]
>>> cuadrados
[1, 4, 9, 16, 25]
```

Como las cadenas de caracteres, las listas pueden ser indexadas y rebanadas:

```
>>> cuadrados[0]  # índices retornan un ítem
1
>>> cuadrados[-1]
25
>>> cuadrados[-3:]  # rebanadas retornan una nueva lista
[9, 16, 25]
```

Todas las operaciones de rebanado devuelven una nueva lista conteniendo los elementos pedidos. Esto significa que la siguiente rebanada devuelve una copia superficial de la lista:

```
>>> cuadrados[:]
[1, 4, 9, 16, 25]
```

Las listas también soportan operaciones como concatenación:

```
>>> cuadrados + [36, 49, 64, 81, 100]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

A diferencia de las cadenas de texto, que son *inmutable*s, las listas son un tipo *mutable*, es posible cambiar un su contenido:

```
>>> cubos = [1, 8, 27, 65, 125]  # hay algo mal aquí
>>> 4 ** 3  # el cubo de 4 es 64, no 65!
64
>>> cubos[3] = 64  # reemplazar el valor incorrecto
>>> cubos
[1, 8, 27, 64, 125]
```

También podemos agregar nuevos ítems al final de la lista, usando el *método* **append()** (vamos a ver más sobre los métodos luego):

```
>>> cubos.append(216)  # agregar el cubo de 6
>>> cubos.append(7 ** 3)  # y el cubo de 7
>>> cubos
[1, 8, 27, 64, 125, 216, 343]
```

También es posible asignar a una rebanada, y esto incluso puede cambiar la longitud de la lista o vaciarla totalmente:

```
>>> letras = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> letras
['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> # reemplazar algunos valores
>>> letras[2:5] = ['C', 'D', 'E']
>>> letras
['a', 'b', 'C', 'D', 'E', 'f', 'g']
>>> # ahora borrarlas
>>> letras[2:5] = []
>>> letras
['a', 'b', 'f', 'g']
>>> # borrar la lista reemplzando todos los elementos por una lista vacía
>>> letras[:] = []
>>> letras
[]
```

La función predefinida **len()** también sirve para las listas:

```
>>> letras = ['a', 'b', 'c', 'd']
>>> len(letras)
4
```

Es posible anidar listas (crear listas que contengan otras listas), por ejemplo:

```
>>> a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> x
[['a', 'b', 'c'], [1, 2, 3]]
>>> x[0]
['a', 'b', 'c']
>>> x[0][1]
'b'
```

La sentencia if

Tal vez el tipo más conocido de sentencia sea el **if**. Por ejemplo:

```
>>> x = int(input("Ingresa un entero, por favor: "))
Ingresa un entero, por favor: 42
>>> if x < 0:
...      x = 0
...      print('Negativo cambiado a cero')
... elif x == 0:
...      print('Cero')
... elif x == 1:
...      print('Simple')
... else:
...      print('Más')
...
'Mas'
```

Puede haber cero o más bloques elif, y el bloque else es opcional. La palabra reservada 'elif' es una abreviación de 'else if', y es útil para evitar un sangrado excesivo. Una secuencia if ... elif ... elif ... sustituye las sentencias switch o case encontradas en otros lenguajes.

La sentencia for

La sentencia for en Python difiere un poco de lo que uno puede estar acostumbrado en lenguajes como C o Pascal. En lugar de siempre iterar sobre una progresión aritmética de números (como en Pascal) o darle al usuario la posibilidad de definir tanto el paso de la iteración como la condición de fin (como en C), la sentencia for de Python itera sobre los ítems de cualquier secuencia (una lista o una cadena de texto), en el orden que aparecen en la secuencia. Por ejemplo:

```
>>> # Midiendo cadenas de texto
... palabras = ['gato', 'ventana', 'defenestrado']
>>> for p in palabras:
...     print(p, len(p))
...
gato 4
ventana 7
defenestrado 12
```

Si necesitamos modificar la secuencia sobre la que estás iterando mientras estás adentro del ciclo (por ejemplo para borrar algunos ítems), se recomienda que hagas primero una copia. Iterar sobre una secuencia no hace implícitamente una copia. La notación de rebanada es especialmente conveniente para esto:

```
>>> for p in palabras[:]:  # hace una copia por rebanada de toda la lista
```

```
...     if len(p) > 6:
...         palabras.insert(0, p)
...
>>> palabras
['defenestrado', 'ventana', 'gato', 'ventana', 'defenestrado']
```

Con for w in words: el ejemplo intentaría crear una lista infinita, insertando defenestrado una y otra vez.

Las sentencias break, continue, y else.

La sentencia break, como en C, termina el lazo for o while más anidado.

Las sentencias de lazo pueden tener una cláusula else que es ejecutada cuando el lazo termina, luego de agotar la lista (con for) o cuando la condición se hace falsa (con while), pero no cuando el lazo es terminado con la sentencia break. Se ejemplifica en el siguiente lazo, que busca números primos:

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(n, 'es igual a', x, '*', n/x)
...             break
...     else:
...         # sigue el bucle sin encontrar un factor
...         print(n, 'es un numero primo')
...
2 es un numero primo
3 es un numero primo
4 es igual a 2 * 2
5 es un numero primo
6 es igual a 2 * 3
7 es un numero primo
8 es igual a 2 * 4
9 es igual a 3 * 3
```

Cuando se usa con un ciclo, el else tiene más en común con el else de una declaración try que con el de un if: el else de un try se ejecuta cuando no se genera ninguna excepción, y el else de un ciclo se ejecuta cuando no hay ningún break. Para más sobre la declaración try y excepciones, mirá *Manejando excepciones*.

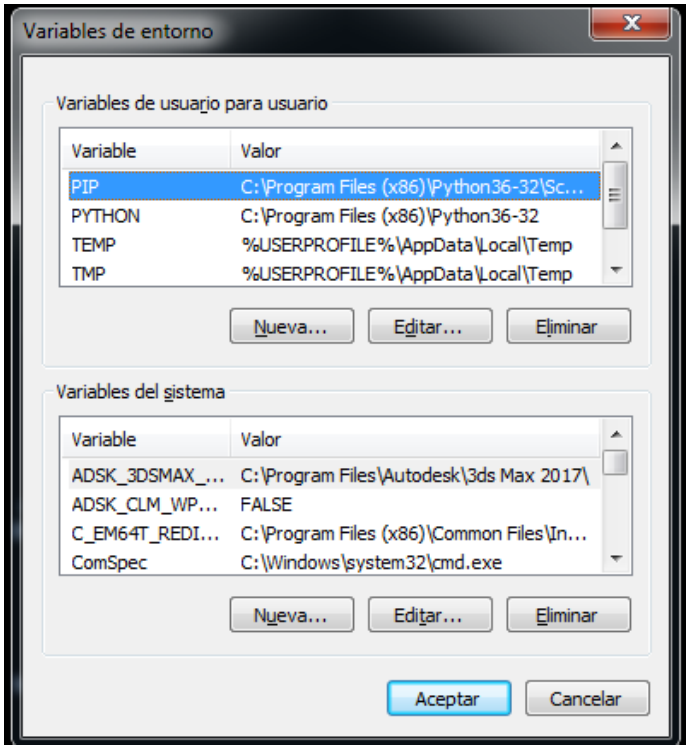
La declaración *continue*, también tomada de C, continúa con la siguiente iteración del ciclo:

```
>>> for num in range(2, 10):
...     if num % 2 == 0:
...         print("Encontré un número par", num)
...         continue
...     print("Encontré un número", num)
Encontré un número par 2
Encontré un número 3
Encontré un número par 4
Encontré un número 5
Encontré un número par 6
Encontré un número 7
Encontré un número par 8
Encontré un número 9
```

Instalación de Python y librerías.

Para el desarrollo del software se optó utilizar "WinPython-PyGaze-0.5.1", una versión especial de Python 2.7.3 que ya incluye la librería PyGaze, junto con una serie de aplicaciones, como la interfaz Spyder, la misma está disponible para su descarga en el repositorio de ***github*** del proyecto.

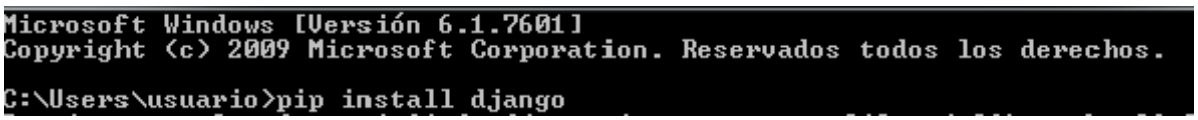
Aun así, se explicará cómo realizar una instalación desde cero, de Python y sus Librerías. En Windows basta con descargar el instalador de Python de la página oficial y ejecutar el instalaor. El paso siguiente sería añadir a Python dentro de las variables de entorno de Windows, para asi poder ejecutarlo desde el ***command prompt***, el símbolo del sistema, no es más que un interpretador de líneas de comando, tal como Python. Para hacer esto hay que editar las variables de entorno (Inicio > Editar variables de entorno del sistema). Se abrirán las propiedades del sistema, y hay que hacer click en la opción "Variables de entorno." Luego en la sección "Variables del Sistema", buscar la variable "Path" y editarla. Aquí, separadas por punto y coma, debemos agregar las rutas tanto de la carpeta Raiz donde se encuentra el ejecutable de Python, como la carpeta Scripts.



Para testear que se instaló correctamente abriremos un command prompt***, (Inicio > CMD)*** y escribimos Python. Si se instaló correctamente debería entregarnos la Versión de Python con fecha y horario. Y finalmente, para corroborar que el instalador de librerías funcione correctamente, Abriremos nuevamente un command prompt y escribimos **pip install**. Debería devolvernos un aviso, sobre que es necesario nombrar la librería que deseamos instalar.

La ejecución del comando *pip install* en el command propt es el método a utilizar para instalar cualquier librería en Python. A manera de ejemplo, instalaremos la reconocida librería Django, la cual provee un framework para el desarrollo web con Python.

Para instalar Django debemos iniciar un command prompt (Inicio > CMD), y tipear el comando **pip install Django==1.11.3.** El comando pip install, le indica al módulo pip, encargado de la descarga e instalación de librerías que ejecute el comando install, al cual le debe seguir por escrito el nombre del módulo.



Una vez terminado el proceso de descarga e instalación la librería debemos comprobar que se instalo correctamente, esto lo hacemos al abrir nuevamente un command prompt, y ejecutar Python. Una vez cargado Python en el command prompt, debemos tipear el comando **import django**. Si no devuelve ningún aviso de error, es porque reconoce el módulo y esta correctamente instalado y listo para usarse.

Re escalamiento de valores – conversión PX a MM

A continuación se presentan las fórmulas necesarias para el reescalado de los pixeles de pantalla a

los milímetros que el robot recorrerá en el plano.

Debemos en primer lugar calcular un factor de relación entre hoja sobre la que trabajará el robot y la pantalla sobre la cual se visualiza el experimento. Para cada eje del plano, un factor de X y un factor de Y.

El cálculo de estos factores se obtiene de la división del tamaño de la hoja (en este caso a4) en mm por el tamaño de la pantalla en mm, ambos cálculos realizados para cada eje individualmente.

```
FactorX = (X_a4_mm / X_pantalla_mm)
FactorY = (Y_a4_mm / Y_pantalla_mm)
```

Luego debemos obtener el valor en mm del tamaño de pantalla, siendo esta la resolución en pixels en la cual se visualizará el experimento. De nuevo, el cálculo se realiza para cada eje por separado.

```
PX_mm_X = (X_pantalla_mm / DISPSIZE[0])
PX_mm_Y = (Y_pantalla_mm / DISPSIZE[1])
```

Finalmente, obtenemos la posición de X e Y que enviaremos al robot mediante la multiplicación de los valores obtenidos por el eyetracker por el factor obtenido en el primer cálculo, para luego multiplicarlo por los valores en mm de la pantalla de visualización obtenidos en el segundo cálculo.

```
PosX = (PX_mm_X * (gazeposX * FactorX))
PosY = (PX_mm_Y * (gazeposY * FactorY))
```

BILIOGRAFÍA

1. **Palmer, S. E.** *Vision Science: Photons to Phenomenology.* s.l. : MIT Press, 1999.

2. **Carbajal, María Julia.** *Organización de los procesos cognitivos en lectura natural.* s.l. : Tesis de Lic en Ciencias Fisicas, Univerisdad de Buenos Aires, 2013.

3. **Polosecki, Pablo Ignacio.** *De la vida mental sobre un espacio de formas: Aprendizaje, reglas y decisiones.* 2008.

4. **Rayner, K.** *Eye movements in reading and information processing: 20 years of research.* s.l. : Psychological Bulletin, 1998.

5. **Yarbus, A. L.** *Eye movements and Vision.* s.l. : Plenum Press., 1967.

6. **Huey, Edmund.** *The Psychology and Pedagogy of Reading .* s.l. : MIT Press, 1968.

7. **Robert J.K Jacob, Keith S. Karn.** *Eye Tracking in Human–Computer Interaction and Usability Research: Ready to deliver promises.* 2003.

8. **Hoffman, J.E.** *Visual attention and eye movements.* s.l. : H. Pashler, 1998.

9. **Robinson, David.** *A method of measuring eye movement using a scleral search coil in a magnetic field, IEEE Transactions on Bio-Medical Electronics.* 1963.

10. **Keren, A.S., Yuval-Greenberg, S. y Deouell, L.Y.** *Saccadic spike potentials in gamma-band EEG: Characterization, detection and suppression.* s.l. : NeuroImage, 2010.

11. **Ltd, SR Research.** *Eye-Link User Manual for EyeLink 1000, 2000 & Remote.* 2005.

12. **Kurfess, Thomas R.** *Robotics and automation handbook.* s.l. : CRC Press., 2005.

13. **8373:2012:, ISO.**

14. **International Organization for Standarization.** Industrial Robots. Manipulators. [En línea] https://www.iso.org/ics/25.040.30/x/.

15. **Pittman, Kagan.** Engineering.com. [En línea] 28 de Octubre de 2016. https://www.engineering.com/AdvancedManufacturing/ArticleID/13540/A-History-of-Collaborative-Robots-From-Intelligent-Lift-Assists-to-Cobots.aspx.

16. **Universal Robots.** *Universal Robots Manual.* 2016.

17. **Lars Skovsgaard, Zacobria.** X-Y-Z Positions and angle positions. [En línea] http://www.zacobria.com/universal-robots-zacobria-forum-hints-tips-how-to/x-y-and-z-position/.

18. **Python Software Fundation.** *History and License.* [En línea] https://docs.python.org/3/license.html.

19. **Dwoney, Allen.** *How to think like a compute scientist: Learn with Python.* 2002.

20. **Dalmaijer, E. S.** PyGaze. [En línea] 2013-2016. http://www.pygaze.org/docs/.

21. **Universal Robots.** *The URScript Programming Lenguaje.* 2013.