Universidad Nacional de Ingeniería

Área de Conocimiento de Tecnología de la Información y Comunicación

“PROPUESTA DE SISTEMA COMO APLICACIÓN POR MEDIO DE UNA CÁMARA CON SENSOR DE PROFUNDIDAD UTILIZANDO GESTOS”

TRABAJO MONOGRÁFICO PARA OPTAR AL TÍTULO DE INGENIERO EN COMPUTACIÓN

Elaborado por:		Tutor:
Br. Mitchel Oswaldo Flores Cantor Carnet: 2016-0515I	Br. Jeremy Antoine Sobalvarro West Carnet: 2016-0112I	MSC. Ing. Cedrick Dalla-Torre Parrales

viernes, 14 de junio 2024

Agradecimiento

Quiero expresar mi profundo agradecimiento a todas las personas que han contribuido de manera significativa a la realización de esta tesis. Este proyecto ha sido posible gracias al apoyo y la colaboración de diversas personas e instituciones.

A nuestros padres: Luis Sobalvarro, Joan West, Mitchel Flores Lacayo y Amparo Cantor, un agradecimiento especial por su amor y apoyo constante. Su influencia ha sido la fuerza presente que ha guiado cada paso del camino.

Agradecemos sinceramente a nuestro tutor de tesis, Cedrik Dalla Torres, por su orientación experta y constante apoyo a lo largo de este proceso de investigación. Su guía ha sido fundamental para dar forma a este trabajo.

También agradecemos a nuestros profesores de la carrera por brindarnos los conocimientos necesarios para esta tesis. La riqueza de conocimientos y recursos disponibles ha sido crucial para el éxito de este proyecto.

Nuestra gratitud se extiende a nuestros compañeros de clase y amigos de la carrera Felipe, Alejandra, Kevin y Jader, por sus valiosas sugerencias y comentarios constructivos que han mejorado la calidad de este trabajo.

Dedicamos este logro también a nosotros mismos, reconociendo el esfuerzo y la dedicación personal invertidos en este proyecto.

INDICE

INDICE DE FIGURAS

INDICE DE TABLAS

I. INTRODUCCIÓN

La sistemización y automatización de los hogares de forma inteligente ya es cada vez una necesidad en el día a día actual. El poseer y hacer procesos de forma automática atrae cada vez más y más gente que desea conseguir ese tipo de tecnología para su seguridad en cualquier tipo de ambiente como negocios, hogares, institutos, etc.

La domótica y la interacción humano-computadora han emergido como áreas de gran interés, buscando mejorar la eficiencia y la comodidad en nuestros hogares mediante la integración de dispositivos electrónicos inteligentes.

Al mismo tiempo, por la seguridad no solo de los hogares o edificios, sino también de la vida humana presente, especialmente de personas convalecientes o con capacidades diferentes, se busca hacer la tecnología más accesible para todos.

Por lo tanto, este trabajo de tesis presenta una solución a los contratiempos al utilizar tecnología compleja, consistente en una aplicación para Windows 10 que utiliza el Lenguaje C# y el dispositivo de cámara de profundidad Kinect. Esta aplicación permite registrar y utilizar comandos hechos con la mano para realizar diversas tareas que pueden aplicarse a la vida cotidiana.

La aplicación funciona como una base de comandos que recibe entradas por medio de un sensor que registra la mano derecha. Se accede a las diferentes opciones mediante gestos o formas realizadas con la mano alzada, incluyendo una combinación de gestos utilizados en el lenguaje de señas americano y números binarios contados con una sola mano.

Con la intención de darle una nueva vida a este dispositivo por medio de un programa, nos enfocamos en su utilización para dispositivos a distancia, aplicaciones de domótica e incluso en el desarrollo de habilidades motrices para la comunicación.

Existen diversos dispositivos que ayudan a personas con movilidad limitada o discapacidades, como sensores, micrófonos y cámaras. Estos dispositivos son programables para adaptarse a diferentes necesidades según el entorno o la circunstancia.

Kinect tiene múltiples sensores que se pueden utilizar de diversas formas. Al asignarles un comando común de salida para un computador intérprete, proporciona la flexibilidad necesaria para ser más que una herramienta multiusos capaz en diversas situaciones, dificultades o modos de vida.

En resumen, este proyecto representa un avance significativo en el campo de la domótica y la interacción humano-computadora, con el potencial de mejorar la calidad de vida de las personas y contribuir al desarrollo de soluciones tecnológicas innovadoras en el ámbito del entretenimiento y la automatización del hogar.

II. JUSTIFICACIÓN

En la actualidad, la interacción humano-computadora ha evolucionado significativamente, pasando de los tradicionales teclados y ratones a interfaces más intuitivas y naturales. Sin embargo, muchas de estas tecnologías aún presentan limitaciones en términos de accesibilidad y adaptación a las necesidades específicas de ciertos usuarios, como personas con movilidad reducida o discapacidades. En este sentido, el desarrollo de soluciones innovadoras que permitan una interacción más fluida y accesible se vuelve crucial.

La tecnología Kinect del Xbox One ofrece una plataforma única para explorar estas posibilidades, ya que permite el seguimiento preciso de los gestos y movimientos del cuerpo, ofreciendo una forma natural y efectiva de interactuar con los dispositivos y sistemas. Al utilizar esta tecnología para controlar la domótica y las funciones del computador mediante gestos con la mano, se puede proporcionar una experiencia más intuitiva y adaptable a las necesidades de cada usuario

Proyectamos que esta aplicación podría mejorar la vida diaria de personas con discapacidades o movilidad reducida, así como beneficiar a cualquier usuario que busque simplificar y optimizar su experiencia en el hogar. La capacidad de controlar dispositivos y funciones del hogar mediante gestos con la mano ofrece una forma más intuitiva de interactuar con la tecnología. Aunque no pudimos medir directamente estos cambios, creemos que esta solución puede aumentar la eficiencia y la comodidad en el hogar. La opción de personalizar los gestos según las preferencias y necesidades individuales añade un nivel de adaptabilidad y flexibilidad que promete mejorar significativamente la experiencia del usuario.

Nuestro objetivo fue utilizar los recursos disponibles y aprovechar nuestra experiencia previa, lo que nos llevó a elegir el Kinect. Aunque esta tecnología no es la más moderna, su capacidad para seguir gestos y movimientos con precisión nos permitió ofrecer una experiencia de usuario fluida y satisfactoria. La aplicación, además, se integra con una variedad de dispositivos y sistemas en el hogar, como luces, electrodomésticos y sistemas de entretenimiento, ofreciendo así una notable versatilidad y funcionalidad.

En resumen, se espera que con los módulos desarrollados para esta aplicación, haya un grado de control mayor sobre varios elementos del computador e incluso del hogar para una gran cantidad de usuarios y así facilitar la vida diaria de muchas maneras que ni siquiera están previstas.

III. OBJETIVOS

3.1. Objetivo General

Desarrollar una aplicación de escritorio utilizando un sensor de profundidad, que permita controlar diversas funciones del hogar y del computador mediante gestos realizados con la mano derecha, con el fin de ofrecer una fácil y amplia interacción en el ámbito doméstico y en otros entornos.

3.2. Objetivos específicos

- Investigar y analizar las tecnologías disponibles y las metodologías de desarrollo más adecuadas para la implementación de la aplicación, con un enfoque en la utilización del sensor Kinect del Xbox One y el lenguaje de programación C#.

- Diseñar e implementar una interfaz de usuario intuitiva y adaptable, que permita a los usuarios controlar de forma efectiva y sencilla diversos dispositivos y funciones del hogar, como la reproducción de música y videos, el control de la iluminación y la gestión de la domótica.

- Implementar algoritmos de reconocimiento de gestos con la mano derecha, con el fin de asegurar una interacción fluida y eficiente con los dispositivos y funciones del hogar.

IV. MARCO TEÓRICO

1. Interfaz

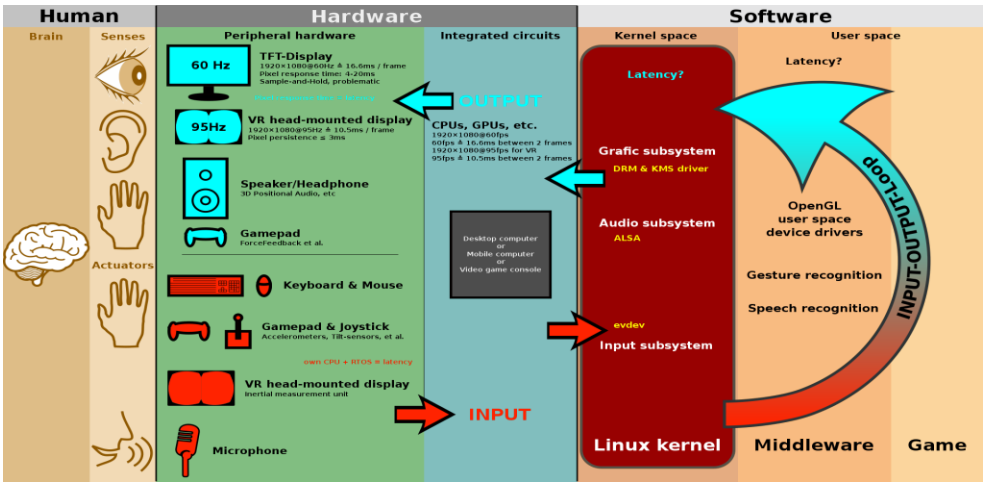
1.1 Reconocimiento de gestos

El reconocimiento de gestos en las aplicaciones de computadora ha experimentado un crecimiento significativo en los últimos años, permitiendo a los usuarios interactuar de forma natural con gestos de mano y cuerpo en lugar de dispositivos tradicionales como teclados y ratones. Se basa en el análisis de movimientos y posiciones de manos y cuerpo capturados por cámaras y sensores para detectar y reconocer gestos específicos.

Una de las ventajas principales es su capacidad para mejorar la experiencia de usuario al proporcionar una forma intuitiva e interactiva de interactuar con las aplicaciones, especialmente en entornos donde el uso de dispositivos tradicionales puede ser limitado o incómodo. Además, tiene un amplio potencial en áreas como la salud, la educación y el entretenimiento.

Por ejemplo, en la rehabilitación física, se pueden desarrollar aplicaciones que utilicen gestos para realizar ejercicios y terapias de forma más precisa y controlada. En educación, el reconocimiento de gestos puede promover la participación activa de los estudiantes y mejorar la comprensión de conceptos complejos. En el entretenimiento, permite a los usuarios interactuar con juegos y aplicaciones de manera más envolvente y divertida.

Figura 1 - El reconocimiento de gestos se procesa en middleware, los resultados se transmiten a las aplicaciones del usuario. (Human-Machine Interface, Otto 2014)



Sin embargo, el reconocimiento de gestos en las aplicaciones de computadora también presenta desafíos técnicos. La precisión y la robustez del sistema son aspectos clave a considerar, ya que los gestos pueden ser difíciles de detectar y reconocer con alta

precisión debido a variaciones en la iluminación, el fondo y la posición del usuario. Además, la creación de una biblioteca de gestos que sea amplia y flexible requiere una cuidadosa planificación y diseño.

Entre las funciones de reconocimiento de gestos tenemos:

- Más preciso
- Alta estabilidad
- Ahorro de tiempo para desbloquear un dispositivo

Las principales áreas de aplicación del reconocimiento de gestos en el escenario actual son:

- Sector automotriz
- Sector de la electrónica de consumo
- Sector de tránsito
- Sector del juego
- Para desbloquear teléfonos inteligentes
- Defensa
- Automatización del hogar (Domótica)
- Traducción automática de lengua de signos

1.1.1 Dispositivos de entrada

La capacidad de rastrear los movimientos de una persona y determinar qué gestos pueden estar realizando se puede lograr a través de varias herramientas. Las interfaces de usuario cinéticas (KUI) son un tipo emergente de interfaces de usuario que permiten a los usuarios interactuar con dispositivos informáticos a través del movimiento de objetos y cuerpos.

Los ejemplos de KUI incluyen interfaces de usuario tangibles y juegos con reconocimiento de movimiento, como Wii y Kinect de Microsoft, y otros proyectos interactivos. [1]

Aunque se ha realizado una gran cantidad de investigación sobre el reconocimiento de gestos basado en imágenes/vídeos, existe cierta variación en las herramientas y los entornos utilizados entre las implementaciones.

Guantes con cable: Estos pueden proporcionar información a la computadora sobre la posición y la rotación de las manos utilizando dispositivos de seguimiento magnéticos o de inercia.

Cámaras con reconocimiento de profundidad: Usando cámaras especializadas, como cámaras de luz estructurada o de tiempo de vuelo, se puede generar un mapa de profundidad de lo que se ve a través de la cámara a corta distancia y usar estos datos para aproximar una representación en 3D de lo que se ve. Estos pueden ser efectivos para la detección de gestos con las manos debido a sus capacidades de corto alcance.

Cámaras estéreo: Utilizando dos cámaras cuyas relaciones entre sí se conocen, la salida de las cámaras puede aproximar una representación 3D.

Controladores basados en gestos: Estos controladores actúan como una extensión del cuerpo, de modo que cuando se realizan gestos, parte de su movimiento puede ser capturado convenientemente por software. Un ejemplo de captura de movimiento emergente basada en gestos es a través del seguimiento de la mano esquelética, que se está desarrollando para aplicaciones de realidad virtual y realidad aumentada.

1.2 Cámara de Profundidad

La cámara de profundidad desempeña un papel fundamental como el principal medio a través del cual el computador interpreta los gestos realizados por los usuarios como método de entrada. Esta cámara captura información detallada sobre la geometría y la posición de las manos y el cuerpo de los usuarios en relación con el entorno.

Las cámaras de profundidad permiten capturar la presencia y posición del cuerpo. Junto con el actor de Isadora OpenNITracker, nos permiten reconstruir un esqueleto de cada cuerpo mostrando con precisión la posición en el espacio (x,y,z) de cada parte del cuerpo (manos, cabeza, pies, tronco, codos, rodillas, etc.).

Al utilizar una cámara de profundidad, el sistema puede medir con precisión la distancia entre la cámara y los objetos en la escena, permitiendo la detección y el seguimiento preciso de los gestos de los usuarios. La información de profundidad capturada se procesa mediante algoritmos de visión por computadora para identificar los gestos específicos.

Estos gestos se convierten en comandos y acciones que el computador utiliza para interactuar con la aplicación o sistema en uso, incluyendo movimientos como deslizamientos, giros y gestos de pellizco. La cámara de profundidad permite una interacción intuitiva y natural, sustituyendo dispositivos tradicionales como teclados o ratones, y abre posibilidades en aplicaciones como el control de interfaces gráficas y la interacción en entornos de realidad virtual o aumentada.

Este tipo de tecnología comenzó usándose en la creación de videojuegos, para animar personajes 3D a partir de movimientos reales de un actor. Así mismo son una tecnología muy asequible para la creación de instalaciones interactivas que trabajan con la interacción de personas.[2]

1.2.1 Microsoft Kinect

Kinect para Xbox 360, o simplemente Kinect (originalmente conocido por el nombre en clave «Project Natal»), fue «un controlador de juego libre y entretenimiento» creado por Alex Kipman, desarrollado por Microsoft para la consola de videojuegos Xbox 360, y desde junio de 2011 para PC a través de Windows 7 y Windows 8. Kinect permite a los usuarios controlar e interactuar con la consola sin necesidad de tener contacto físico con un controlador de videojuegos tradicional, mediante una interfaz natural de usuario que reconoce gestos, comandos de voz, y objetos e imágenes.

Kinect se desarrolló originalmente como un periférico de controlador de movimiento para las consolas de videojuegos Xbox, que se distingue de los competidores (como el Wii Remote de Nintendo y el PlayStation Move de Sony) al no requerir controladores físicos. El Kinect de primera generación se basó en la tecnología de la empresa israelí PrimeSense y se presentó en el E3 2009 como un periférico para Xbox 360 con el nombre en código "Proyecto Natal".

La mayoría de los juegos desarrollados para Kinect eran títulos casuales orientados a la familia, lo que ayudó a atraer nuevas audiencias a Xbox 360, pero no resultó en una adopción generalizada por parte de la base de usuarios general existente de la consola.

Kinect también se ha utilizado como parte de aplicaciones que no son de juegos en entornos académicos y comerciales, ya que era más barato y más robusto en comparación con otras tecnologías de detección de profundidad en ese momento. Si bien Microsoft inicialmente se opuso a tales aplicaciones, luego lanzó kits de desarrollo de software (SDK) para el desarrollo de aplicaciones de Microsoft Windows que usan Kinect.

1.3 Microsoft Windows

Windows es un sistema operativo creado por Microsoft. Consiste en un conjunto de programas que permiten la ejecución de los recursos que tiene un ordenador. El significado del término (windows, ventanas) hace alusión a su interfaz gráfica, que presenta un modelo basado en tareas y compartimentos independientes, con sus propios menús y controles.

El año 1975 fue el inicio del fenómeno pronto responsable de la expansión del ordenador para uso personal a nivel mundial. En esa fecha, dos chicos llamados Bill Gates y Paul Allen fundaron la compañía Microsoft, unidos por el interés de crear un lenguaje Basic para el primer ordenador personal, el Altair 8800.[3]

2. Domótica e Inmótica

2.1 Domótica:

La domótica es el conjunto de tecnologías aplicadas al control y la automatización inteligente de la vivienda, que permite una gestión eficiente del uso de la energía, que aporta seguridad y confort, además de comunicación entre el usuario y el sistema.

Por lo tanto, son los sistemas capaces de automatizar una vivienda, aportando servicios de gestión energética, seguridad, bienestar y comunicación, y que pueden estar integrados por medio de redes interiores y exteriores de comunicación, cableadas o inalámbricas, y cuyo control goza de cierta ubicuidad, desde dentro y fuera del hogar. Se podría definir como la integración de la tecnología en el diseño inteligente de un recinto cerrado.

Al estar más orientada hacia los productos propios de un hogar, como electrodomésticos,

persianas, sistemas de climatización, bombillas, etc, se inclinaría a buscar una interfaz más sencilla, con sensores muy pequeños que no supongan una ruptura brusca en el ambiente familiar. Su objetivo es, sobre todo, generar el máximo confort en la casa, la comunicación entre el sistema y las personas que la habitan y, por supuesto, optimizar y reducir el consumo de energía, con el consiguiente ahorro en las facturas. [4]

2.2 Inmótica:

La inmótica es casi lo mismo que la domótica, salvo que no va destinada a viviendas residenciales sino a hoteles, centros comerciales, escuelas, hospitales, universidades y edificios terciarios. Es decir, la principal diferencia entre domótica e inmótica es la superficie en la que se implantan, ya que la tecnología que se emplea en cada una de ellas es parecida.

La inmótica, entre tanto, centra su objetivo en la optimización del consumo energético a gran escala, lo que requiere la participación de un supervisor capacitado que compruebe el funcionamiento del sistema y que integre la domótica interna dentro de una estructura en red. El objetivo es conseguir un máximo ajuste, ahorro de energía y dinero, y la eficacia en la gestión de los edificios.

3. Comunicación

3.1 Gesto

Un gesto es una forma de comunicación no verbal o comunicación no vocal en la que las acciones corporales visibles comunican mensajes particulares, ya sea en lugar o junto con el habla. Los gestos incluyen el movimiento de las manos, la cara u otras partes del cuerpo. Los gestos difieren de la comunicación física no verbal que no comunica mensajes específicos, como demostraciones puramente expresivas, proxémicas o demostraciones de atención conjunta.

Los gestos permiten a las personas comunicar una variedad de sentimientos y pensamientos, desde el desprecio y la hostilidad hasta la aprobación y el afecto, a menudo junto con el lenguaje corporal además de las palabras cuando hablan. La gesticulación y el habla funcionan de forma independiente, pero se unen para proporcionar énfasis y significado.

En las interfaces de computadora, se distinguen dos tipos de gestos: Consideramos gestos en línea, que también pueden considerarse manipulaciones directas como escalar y rotar. Por el contrario, los gestos fuera de línea generalmente se procesan después de que finaliza la interacción; por ejemplo, se dibuja un círculo para activar un menú contextual.[5]

Gestos offline: Aquellos gestos que se procesan tras la interacción del usuario con el objeto. Un ejemplo es el gesto para activar un menú.

Gestos online: Gestos de manipulación directa. Se utilizan para escalar o rotar un objeto

tangible.

3.3 Binario

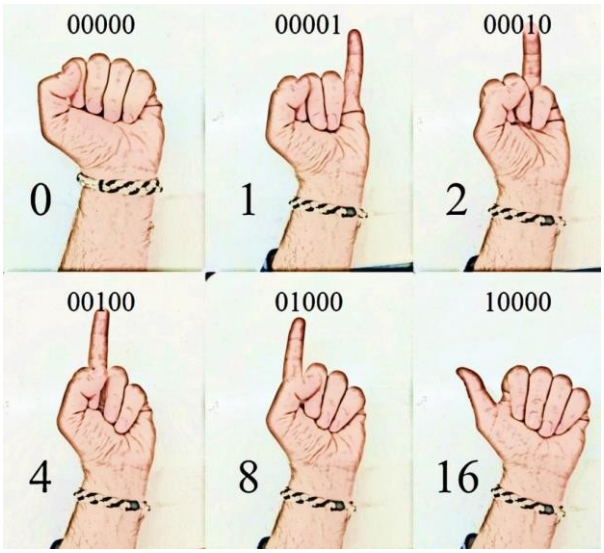
El sistema binario, también llamado sistema diádico1 en ciencias de la computación, es un sistema de numeración en el que los números son representados utilizando únicamente dos cifras: 0 (cero) y 1 (uno). Es uno de los sistemas que se utilizan en las computadoras, debido a que estas trabajan internamente con dos niveles de voltaje, por lo cual su sistema de numeración natural es el sistema binario.

3.3.1 Contar en Binario con las manos

El sistema de numeración binario es el utilizado en informática y en toda la tecnología digital para codificar la información (música, imágenes o datos). La ventaja de un sistema con tan solo dos cifras, 1 y 0, es que es fácil de representar con dispositivos físicos simples, como el paso o no de corriente en un circuito de un ordenador, con hendiduras o llanuras en un Compact Disc, con una bombilla encendida o apagada, o con un dedo abierto o cerrado.

Podemos utilizar nuestras manos para realizar las expresiones binarias de los números, teniendo en cuenta que un dedo levantado simbolizará un 1 y cerrado un 0. Si utilizamos solo una mano, podremos representar los números del 0 al 31 mediante las expresiones binarias con cinco dígitos, cada uno de los cuales será un 1 o un 0, es decir, un dedo abierto o cerrado. Así, las potencias de 2 se expresarán digitalmente como: la mano cerrada 0 (en expresión binaria 00000), solo el meñique abierto 1 (00001), solo el anular abierto 2 (00010), solo el corazón 4 (00100), solo el índice 8 (01000) y el pulgar 16 (10000), como se muestra en la siguiente imagen.

Figura 2 - Contando con los dedos de la mano al estilo binario (Ibánez, 2019)



De esta manera podemos utilizar una mano para poder contar con los dedos hasta el 31. Y al utilizar las dos podemos llegar hasta el 1023.[6]

4. Desarrollo del programa

4.1 Metodología del desarrollo del software:

En la ingeniería de software, se define metodología de desarrollo cómo:
Actividades, procedimientos, técnicas, herramientas y documentos, en su conjunto, normados y comprendidos en un marco de trabajo. Sirven de soporte en la estructuración, planificación y control requeridos para lograr la conversión de una necesidad o un grupo de necesidades a un sistema de información de manera eficiente [7].

4.1.1 Metodología DRA (Desarrollo rápido de aplicaciones):

Profundizando en qué es una metodología RAD, tenemos que tener claros cuales son sus premisas básicas. La primera persona que habló del método RAD fue James Martin a finales de los 80 y, actualmente, estamos ante uno de los métodos de desarrollo más populares, dentro de las técnicas de desarrollo ágil. James Martin consideró que para aplicar de forma correcta la metodología RAD tenemos que tener en cuenta 4 componentes: Personas, herramientas, metodología y gestión.

La idea principal es entregar sistemas de alta calidad, en poco tiempo y con un coste bajo de inversión. Para conseguir esto, hay que seguir determinadas pautas que harán que estemos ante una auténtica metodología DRA (las siglas en castellano: Desarrollo Rápido de Aplicaciones) [8]

4.1.2 Programación modular:

La programación modular consiste en la descomposición de un programa en trozos más pequeños denominados módulos o subprogramas, en el que cada uno de ellos se encargara de llevar a cabo una tarea concreta y bien definida, y se agrupara según su funcionalidad. Cada uno de estos módulos se analizara y codificara por separado.

La estructura de un programa modular constara de un módulo principal desde el que se llamara al resto de los módulos. El módulo principal recibe el control al inicio de la ejecución del programa. Cuando se invoca un módulo concreto (a través de su nombre y parámetros), el control del programa se pasara al módulo. Este módulo mantendrá el control hasta que no se finalice su ejecución en cuyo momento devolverá el control a la instrucción siguiente a la que realizo la llamada.

Este es el tipo de programación y estructura que se utilizó en la aplicación para poder desarrollar de forma correcta las diferentes ideas y posibilidades de la aplicación.

4.2 Microsoft Visual Studio

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para Windows y macOS. Es compatible con múltiples lenguajes de programación, tales como C++, C#, Visual Basic, .NET, F#, Java, Python, Ruby y PHP; al igual que

entornos de desarrollo web, como ASP.NET MVC, Django, etc., a lo cual hay que sumarle las nuevas capacidades en línea bajo Windows Azure en forma del editor Mónaco.

Visual Studio permite a los desarrolladores crear sitios y aplicaciones web, así como servicios web en cualquier entorno compatible con la plataforma .NET (a partir de la versión .NET 2002). Así, se pueden crear aplicaciones que se comuniquen entre estaciones de trabajo, páginas web, dispositivos móviles, dispositivos embebidos y videoconsolas, entre otros.

Constituye un IDE (entorno de desarrollo integrado o en inglés Integrated Development Enviroment) que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código (programa donde se escribe el código fuente), un depurador (programa que corrige errores en el código fuente para que pueda ser bien compilado, un compilador (programa que traduce el código fuente a lenguaje de maquina), y un constructor de interfaz gráfica o GUI (es una forma de programar en la que no es necesario escribir el código para la parte grafica del programa, sino que se puede hacer de forma visual).

4.3 Lenguaje de programación C#

C# (pronunciado "C sharp") es un lenguaje de programación desarrollado por Microsoft que combina los principios de C y C++ con la facilidad de uso de Java. Es un lenguaje orientado a objetos y ofrece una amplia gama de características avanzadas, como recolección automática de basura, tipos de datos seguros, y soporte para programación concurrente. Además, C# está diseñado para ser interoperable con otros lenguajes de la plataforma .NET, lo que significa que puede utilizar bibliotecas de código escritas en otros lenguajes, como Visual Basic.NET o F#, lo que lo convierte en una opción flexible para el desarrollo de software.

Utilizar C# para tu tesis es una elección sólida por varias razones. En primer lugar, al ser un lenguaje de programación de propósito general, C# es adecuado para una amplia variedad de aplicaciones, incluidas las aplicaciones de escritorio y las aplicaciones web, lo que te brinda flexibilidad en el alcance de tu proyecto. Además, al utilizar el Kinect de Xbox One para controlar diversas funciones, como reproducir música y videos, así como para la domótica inteligente en una casa, C# te permite acceder fácilmente a las funcionalidades del dispositivo y aprovechar al máximo su potencial. Por último, la amplia comunidad de desarrolladores de C# y la disponibilidad de recursos en línea hacen que sea más fácil encontrar ayuda y recursos para tu proyecto.

4.4 Project Prague

Project Prague es un SDK primordial y fácil de usar que crea experiencias más intuitivas y naturales al permitir que los usuarios controlen e interactúen con las tecnologías a través de gestos con las manos. Basado en una extensa investigación, une a los desarrolladores y diseñadores de UX con la capacidad de diseñar e implementar rápidamente gestos manuales personalizados en sus aplicaciones.

El SDK le permite definir las poses de manos deseadas utilizando restricciones simples creadas con un lenguaje sencillo. Una vez que se define y registra un gesto en su código, recibirá una notificación cuando su usuario haga el gesto y podrá seleccionar una acción para asignar en respuesta.

Con Project Prague, puede permitir que sus usuarios controlen de manera intuitiva videos, marquen páginas web, reproduzcan música, envíen emojis o llamen a un asistente digital. Incluso puede hacer que los programas de productividad y comunicación cotidianos sean más fáciles de usar.

El uso de Project Prague requiere una cámara Intel RealSense SR300 o una cámara Microsoft Kinect v2.[9]

4.4.1 SDK

Un kit de herramientas de desarrollo de software (SDK) es un conjunto de herramientas y programas de software proporcionados por proveedores de hardware y software que los desarrolladores pueden usar para crear aplicaciones para plataformas específicas. Estos proveedores ponen a disposición sus SDK para ayudar a los desarrolladores a integrar fácilmente sus aplicaciones con sus servicios.

Los SDK incluyen documentación, interfaces de programación de aplicaciones (API), muestras de código, bibliotecas y procesos, así como guías que los desarrolladores pueden usar e integrar en sus aplicaciones. Los desarrolladores pueden usar los SDK para crear y mantener aplicaciones sin tener que escribir todo desde cero.

Más específicamente, los SDK incluyen:

Bibliotecas o API: piezas de código predefinidas que permiten a los desarrolladores realizar tareas de programación comunes en la plataforma.

Entorno de desarrollo integrado (IDE): un editor visual que ayuda a los desarrolladores a diseñar y diseñar elementos gráficos, como cuadros de texto y botones.

Herramientas para ayudar a los desarrolladores a realizar tareas como depurar, compilar, ejecutar y probar sus aplicaciones. [10]

4.4.2 API

Una API es un código que permite que dos programas de software se comuniquen entre sí. Una API define la forma correcta para que un desarrollador solicite servicios de un sistema operativo u otra aplicación y exponga datos dentro de diferentes contextos y a través de múltiples canales.

Cuando un desarrollador usa un SDK para desarrollar aplicaciones y crear sistemas, esas aplicaciones deben comunicarse con otras aplicaciones. Un SDK incluye una API para habilitar esa comunicación.

Otras diferencias incluyen:

Los SDK generalmente contienen API, pero las API no contienen SDK.
Aunque se puede usar una API para la comunicación, no se puede usar para crear nuevas aplicaciones.

Los SDK permiten a los desarrolladores crear aplicaciones y actuar como componentes básicos del producto de software.

Las API permiten la función de las aplicaciones dentro de los parámetros del SDK con el que están incluidas. Las API son el código que permite una comunicación claramente definida entre dos aplicaciones separadas.

Un SDK es la pieza de código de herramienta y componente que se ha creado para un propósito específico, mientras que una API es solo una interfaz para un servicio.

V. DISEÑO METODOLÓGICO

1. Tipo de desarrollo

Para el desarrollo de este software, se consideró inicialmente seguir un enfoque tradicional en cascada, que implica una secuencia lineal desde la concepción de ideas hasta la entrega del producto final. Sin embargo, al analizar el alcance del proyecto, se optó por utilizar la Metodología de Desarrollo Rápido de Aplicaciones (RAD, por sus siglas en inglés).

La metodología RAD comparte con el modelo en cascada la idea de trabajar en fases, pero a diferencia de este último, cada fase en RAD añade una funcionalidad al software en desarrollo. Esto permite verificar y probar las mejoras de forma incremental, incluso antes de completar el desarrollo completo de la herramienta. La ventaja principal que tiene con otras metodologías, es que se aprovecha mejor el tiempo y se evita la pérdida de trabajo previo al poder trabajar con partes del software de manera independiente en intervalos cortos..

En este proyecto, se planea diseñar inicialmente la aplicación base con las funcionalidades esenciales para la detección y codificación de gestos. Posteriormente, se agregarán el resto de las funcionalidades y adaptaciones como actualizaciones al mismo programa, permitiendo que las nuevas características se integren de manera coherente y no afecten las funcionalidades previas. Esto garantiza una experiencia fluida para el usuario final, que podrá utilizar las nuevas funciones con los mismos controles o comandos que ya conoce. Además, al seguir esta metodología, se asegura una mayor flexibilidad y adaptabilidad a posibles cambios o requerimientos futuros en el desarrollo del software, lo que resulta especialmente útil en proyectos de tecnología que evolucionan rápidamente.

Para el desarrollo de la aplicación se optó por realizar el proyecto en lenguaje de programación C# utilizando el programa Microsoft Visual Studio 2019. A la vez que es un tipo de lenguaje altamente utilizado aún hoy en día, debido a la naturaleza de utilizar diferentes funciones en un computador, y que la mayoría de computadores utiliza el sistema operativo Windows, se toma en cuenta la adaptabilidad del mismo.

Junto con lo anterior mencionado, se debe mencionar que también otra razón de haber escogido Visual es porque el dispositivo que estamos utilizando para el desarrollo del programa es un Kinect V2, una cámara de profundidad desarrollada también por Microsoft para su consola Xbox One.

El haber utilizado el método de Desarrollo Rápido de Aplicaciones nos permitió tener mejor comprensión del problema: Al dividir el proyecto en partes más pequeñas, pudimos abordar desafíos específicos de manera más efectiva. Esto nos permitió trabajar de forma precisa en cada una de las opciones ofrecidas.

2. Probando el Proyecto Praga

El proyecto Praga engloba diversas aplicaciones que se instalan por defecto durante el proceso de instalación de Microsoft Gestures. Estas aplicaciones fueron originalmente desarrolladas por el equipo de proyecto Praga mientras estaba en vigencia dentro de Microsoft. Sin embargo, cabe señalar que estas aplicaciones presentan ciertas limitaciones, como interfaces de usuario poco intuitivas y en general una experiencia en estado de acceso anticipado que no estaba destinada a ser visualizada por el público en ese momento. Inspirados por el trabajo previo de este equipo, decidimos tomar como referencia lo que habían desarrollado pero que se encontraba inconcluso. Este fue el punto de partida para nuestro propio desarrollo de programas.

En el contexto del proyecto Praga, también se incluían instrucciones detalladas sobre cómo implementar y reconocer nuevos gestos. Estas instrucciones proporcionaban indicaciones específicas sobre qué acciones debían realizar cada dedo para identificar correctamente un gesto. Entre las indicaciones se incluían aspectos como la dirección hacia la cual estaba orientada la palma, la disposición de los dedos (ya sea estirados o enrollados), y la relación espacial entre ellos, como si alguno estaba encima de otro o si se tocaban entre sí. Estas directrices detalladas constituían la base esencial para la comprensión y programación efectiva de gestos dentro de nuestro proyecto.

Para ver la documentación recuperada de Proyecto Praga, por favor referir al Anexo 1 del Documento

3. Desarrollo de la aplicación

El proceso de desarrollo de nuestra aplicación se inició con la implementación del primer módulo, específicamente el dedicado a la música. Esta elección estratégica se basó en la conveniencia proporcionada por la librería del proyecto Praga, la cual ya contenía códigos predefinidos para los gestos necesarios en esta función particular.

La razón de esta decisión es por la cantidad de gestos proporcionados por la librería del proyecto Praga, la cual ya contenía códigos necesarios para empezar a probar algunos comandos. Además de que nos podía proporcionar la confianza para empezar a trabajar el resto de módulos.

3.1 Gestures Samples

Son repositorios de aplicaciones varias realizadas por el equipo de Project Prague listas para ser montadas y ejecutadas en Visual Studio 2017. Estas aplicaciones no contaban con ninguna interfaz gráfica y sólo eran unas pantallas predeterminadas que viene con crear el proyecto en Visual Studio.

Dichas aplicaciones fueron revisadas y tomadas en cuenta para el inicio de la aplicación. Muchos de los módulos iniciales eran traídos de nuestro trabajo anterior de Feria Científica, pero muchos módulos finales fueron inspirados en estos:

Tabla 1 – Gesture Samples

Muestra	Caso de uso	¿Que aprenderás?
ConsoleCore, ConsoleNative	Probar la detección de gestos en una ventana de línea de comando	Cree varios gestos simples y regístrelos en el Servicio de gestos.
RotateSample, RotateSampleUwpManaged, RotateSampleUwpNative	Girar una imagen 90°	Crea un gesto sencillo compuesto por posturas de dos manos.
VideoPlayerGestureControl, VideoPlayerGestureControlUwpNative	Pausar y rebobinar la reproducción de audio/vídeo	Crea un gesto simple compuesto por múltiples posturas de la mano y un movimiento de la mano.
RockPaperScissors	Juega piedra, papel o tijera contra un robot invencible	Crea un gesto compuesto por múltiples poses de manos conectadas en una estructura compleja.
GrabAndPushUwpNative	Ampliar o reducir un control UWP mediante gestos	Crea un gesto simple y consume la corriente del esqueleto de la mano.
Camera3D	Manipular la cámara en una escena 3D.	Crea un gesto simple y consume la corriente del esqueleto de la mano.
GesturesPowerPointPlugin	Crea y presenta una presentación de diapositivas en PowerPoint usando gestos	Cree múltiples gestos simples y conéctelos para activar varias funciones de PowerPoint
CarGestures	Controla el sistema de infoentretenimiento de un coche virtual con gestos	Cree una interfaz de usuario WPF completa con múltiples gestos de diversa complejidad
Unity Package and Tutorial Projects	Sigue nuestro tutorial de Unity	Integra gestos y entradas de esqueletos de manos en tus juegos de Unity

Luego de revisar estas aplicaciones, varias carecían de varias funciones principales y otras que funcionaban bien, sólo utilizaban un gesto para hacer una acción específica, por lo que el principal potencial sería aprovechado al hacer nuestros propios gestos, luego de crear un proyecto propio con toda la información, y realizar diferentes módulos que aprovecharan todo el potencial de la cámara.

Pasamos ahora a mostrar todo el código realizado para cada uno de los módulos y la investigación realizada para ejecutar los gestos correctos y el comando enviado al computador para realizar la acción.

3.2. Music o Media Control.

Estas líneas de código importan las bibliotecas del sistema y otras bibliotecas específicas para la gestión de aplicaciones gráficas, el reconocimiento de gestos y la utilización de un teclado virtual.

- System: Proporciona las clases fundamentales y estructuras básicas de datos en el entorno .NET.
- System.Threading.Tasks: Proporciona clases y estructuras que admiten la ejecución de tareas asincrónicas en .NET.
- System.Windows.Forms: Proporciona clases para crear y administrar aplicaciones de Windows Forms, que son aplicaciones de escritorio con interfaces gráficas de usuario (GUI).
- Microsoft.Gestures: Proporciona funcionalidades para reconocimiento de gestos.
- Microsoft.Gestures.Endpoint: Proporciona un punto final para interactuar con el servicio de reconocimiento de gestos.
- WindowsInput: Proporciona clases para simular entrada de teclado y mouse en Windows.
- WindowsInput.Native: Proporciona tipos que representan las teclas físicas en un teclado, facilitando la simulación de pulsaciones de teclas específicas.

Figura 3 - Librerías necesarias, Media Control

```
1   using System;
2   using System.Threading.Tasks;
3   using System.Windows.Forms;
4   using Microsoft.Gestures;
5   using Microsoft.Gestures.Endpoint;
6   using WindowsInput;
7   using WindowsInput.Native;
```

Estas líneas de código establecen el uso de un servicio encargado de la gestión de gestos, así como la definición de los propios gestos a ser utilizados en la aplicación.

Figura 4 - Servicios encargados de la gestión de gestos, Media Control

```
namespace UIControl
{
    public partial class KMusicForm : Form
    {

        private static GesturesServiceEndpoint _gesturesService;
        private static Gesture _rotateGestureR; //Subir Volumen
        private static Gesture _rotateGestureL; //Bajar Volumen
        private static Gesture _dropTheMicGesture; //Mute Volumen
        private static Gesture _RotateRMedia; //Media Next
        private static Gesture _RotateLMedia; //Media Back
        private static Gesture _RockGesture; //Play/Pause
```

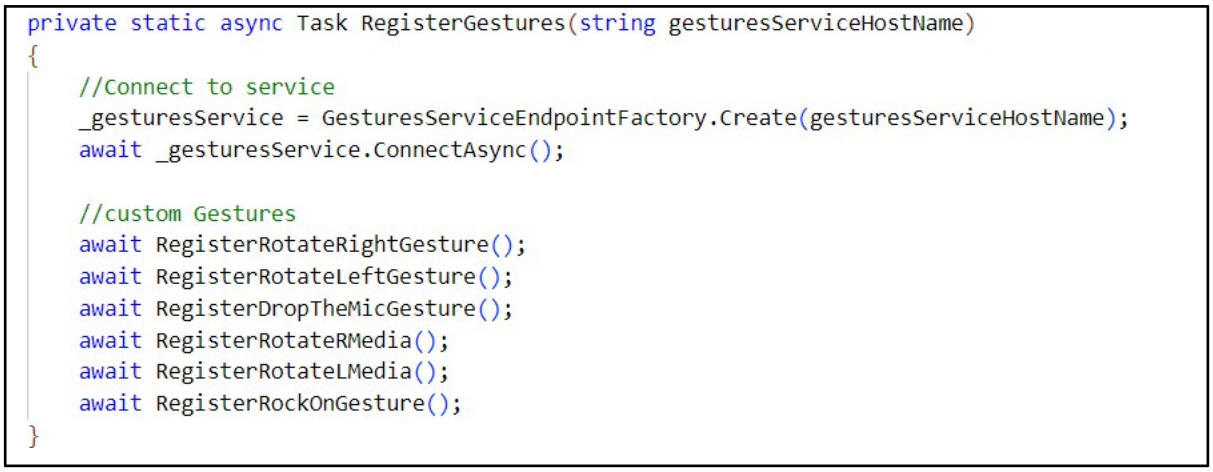
Estas líneas de código inicializan las variables predeterminadas en el constructor y establecen la conexión con el servicio encargado de manejar los gestos, el cual está alojado localmente en el localhost.

Figura 5 - En espera el servicio de la gestión de gestos, Media Control

```
28          public KMusicForm()
29          {
30              InitializeComponent();
31              InitializeGestures();
32          }
33
34
35          private async void InitializeGestures()
36          {
37              var gesturesServiceHostName = "localhost";
38              await RegisterGestures(gesturesServiceHostName);
39          }
```

Estas líneas de código establecen la conexión al servicio encargado de manejar los gestos y comienzan la espera de que los gestos sean activados o "triggered"

Figura 6 - Servicio encargado de la gestión de gestos, Media Control



```
private static async Task RegisterGestures(string gesturesServiceHostName)
{
    //Connect to service
    _gesturesService = GesturesServiceEndpointFactory.Create(gesturesServiceHostName);
    await _gesturesService.ConnectAsync();

    //custom Gestures
    await RegisterRotateRightGesture();
    await RegisterRotateLeftGesture();
    await RegisterDropTheMicGesture();
    await RegisterRotateRMedia();
    await RegisterRotateLMedia();
    await RegisterRockOnGesture();
}
```

3.2.1. Subir Volumen

En este fragmento de código se definen la variables 'Hold' y 'rotate' que representan la primera y segunda pose a tener en cuenta para la detección del gesto.
Hold:
El dedo pulgar e índice están estirados hacia adelante, apuntando hacia la cámara.
Los dedos medio, anular y meñique están doblados hacia adentro.
El dedo índice se encuentra a encima del pulgar y no están tocándose entre sí.

Rotate:
Los dedos pulgar e índice están estirados hacia adelante, apuntando a la cámara.
Los dedos medio, anular y meñique están doblados hacia adentro.

El dedo índice se encuentra a la derecha del pulgar y no están tocándose entre sí.

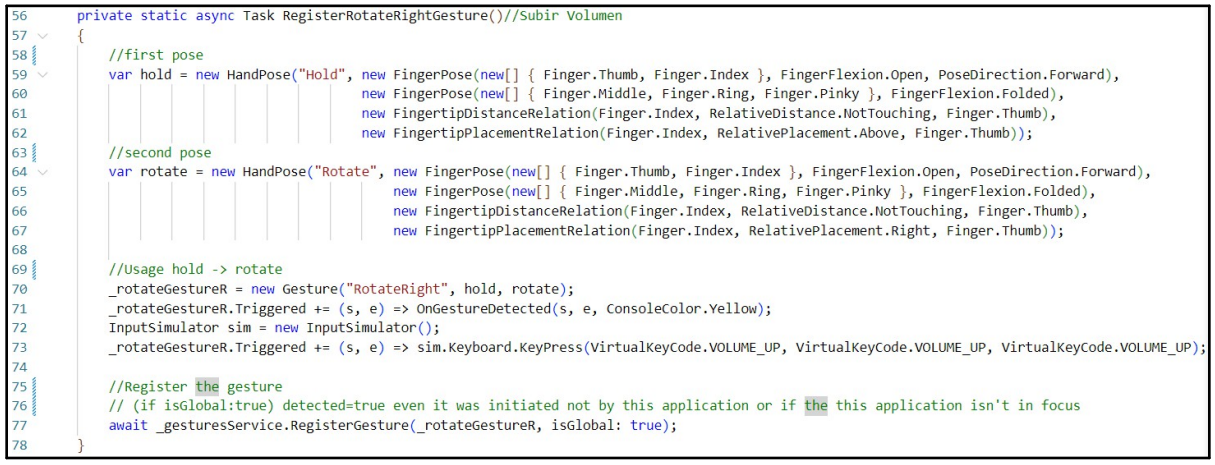
Se define el gesto utilizando las posiciones de mano previamente definidas y se especifica el orden a seguir para la detección:
_rotateGestureR = new Gesture("RotateRight", hold, rotate);

Cuando se detecta este gesto, llamará a la función OnGestureDetected, que se utiliza con fines de prueba.

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simulan tres pulsaciones en la tecla virtual de aumento de volumen (VirtualKeyCode.VOLUME_UP).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_rotateGestureR, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 7 - Subir Volumen, Media Control



```
56      private static async Task RegisterRotateRightGesture()//Subir Volumen
57      {
58          //first pose
59          var hold = new HandPose("Hold", new FingerPose(new[] { Finger.Thumb, Finger.Index }, FingerFlexion.Open, PoseDirection.Forward),
60                                          new FingerPose(new[] { Finger.Middle, Finger.Ring, Finger.Pinky }, FingerFlexion.Folded),
61                                          new FingertipDistanceRelation(Finger.Index, RelativeDistance.NotTouching, Finger.Thumb),
62                                          new FingertipPlacementRelation(Finger.Index, RelativePlacement.Above, Finger.Thumb));
63          //second pose
64          var rotate = new HandPose("Rotate", new FingerPose(new[] { Finger.Thumb, Finger.Index }, FingerFlexion.Open, PoseDirection.Forward),
65                                              new FingerPose(new[] { Finger.Middle, Finger.Ring, Finger.Pinky }, FingerFlexion.Folded),
66                                              new FingertipDistanceRelation(Finger.Index, RelativeDistance.NotTouching, Finger.Thumb),
67                                              new FingertipPlacementRelation(Finger.Index, RelativePlacement.Right, Finger.Thumb));
68
69          //Usage hold -> rotate
70          _rotateGestureR = new Gesture("RotateRight", hold, rotate);
71          _rotateGestureR.Triggered += (s, e) => OnGestureDetected(s, e, ConsoleColor.Yellow);
72          InputSimulator sim = new InputSimulator();
73          _rotateGestureR.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VOLUME_UP, VirtualKeyCode.VOLUME_UP, VirtualKeyCode.VOLUME_UP);
74
75          //Register the gesture
76          // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
77          await _gesturesService.RegisterGesture(_rotateGestureR, isGlobal: true);
78      }
```

3.2.2. Bajar Volumen

Las variables 'hold' y 'rotate' se están utilizando para almacenar configuraciones distintas en otro gesto, lo que implica que están siendo reutilizadas para definir las poses de mano en ese nuevo gesto.

Hold:
El dedo pulgar e índice están estirados hacia adelante, apuntando hacia la cámara.
Los dedos medio, anular y meñique están doblados hacia adentro.
El dedo índice se encuentra encima del pulgar y no están tocándose entre sí.

Rotate:
Los dedos pulgar e índice están estirados hacia adelante, apuntando a la cámara.

Los dedos medio, anular y meñique están doblados hacia adentro.
El dedo índice se encuentra a la izquierda del pulgar y no están tocándose entre sí.

Se define el gesto utilizando el mismo orden a seguir para la detección:
_rotateGestureL = new Gesture("RotateLeft", hold, rotate);

Se llama a la función OnGestureDetected, se inicializa la libreria Input Simulator, al detectarse el gesto, se simulan tres pulsaciones en la tecla virtual de disminucion de volumen (VirtualKeyCode.VOLUME_DOWN).

Se registra el uso del gesto y se declara que funcione aunque la ventana no sea la principal o esté abierta:
await _gesturesService.RegisterGesture(_rotateGestureL:, isGlobal: true);

Figura 8 - Bajar Volumen, Media Control

```
80  private static async Task RegisterRotateLeftGesture()//Bajar Volumen
81  {
82      //first pose
83      var hold = new HandPose("Hold", new FingerPose(new[] { Finger.Thumb }, FingerFlexion.Open, PoseDirection.Forward),
84                                      new FingerPose(new[] { Finger.Middle, Finger.Ring, Finger.Pinky }, FingerFlexion.Folded),
85                                      new FingertipDistanceRelation(Finger.Index, RelativeDistance.NotTouching, Finger.Thumb),
86                                      new FingertipPlacementRelation(Finger.Index, RelativePlacement.Above, Finger.Thumb));
87      //second pose
88      var rotate = new HandPose("Rotate", new FingerPose(new[] { Finger.Thumb, Finger.Index }, FingerFlexion.Open, PoseDirection.Forward),
89                                          new FingerPose(new[] { Finger.Middle, Finger.Ring, Finger.Pinky }, FingerFlexion.Folded),
90                                          new FingertipDistanceRelation(Finger.Index, RelativeDistance.NotTouching, Finger.Thumb),
91                                          new FingertipPlacementRelation(Finger.Index, RelativePlacement.Left, Finger.Thumb));
92
93      //Usage hold -> rotate
94      _rotateGestureL = new Gesture("RotateLeft", hold, rotate);
95      _rotateGestureL.Triggered += (s, e) => OnGestureDetected(s, e, ConsoleColor.DarkYellow);
96      InputSimulator sim = new InputSimulator();
97      _rotateGestureL.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VOLUME_DOWN, VirtualKeyCode.VOLUME_DOWN, VirtualKeyCode.VOLUME_DOWN
98
99  //Register the gesture
00  // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
01      await _gesturesService.RegisterGesture(_rotateGestureL, isGlobal: true);
02  }
```

3.2.3. Silenciar Volumen

Se almacena la configuración del gesto en las variables 'fist' y 'spread'.

Fist:
En el contexto de la mano, sin importar si está cerrada o abierta, la dirección hacia la que debe estar orientada la palma es hacia abajo.
Para la posición de los dedos, el pulgar, índice, medio, anular y meñique deben estar cerrados.

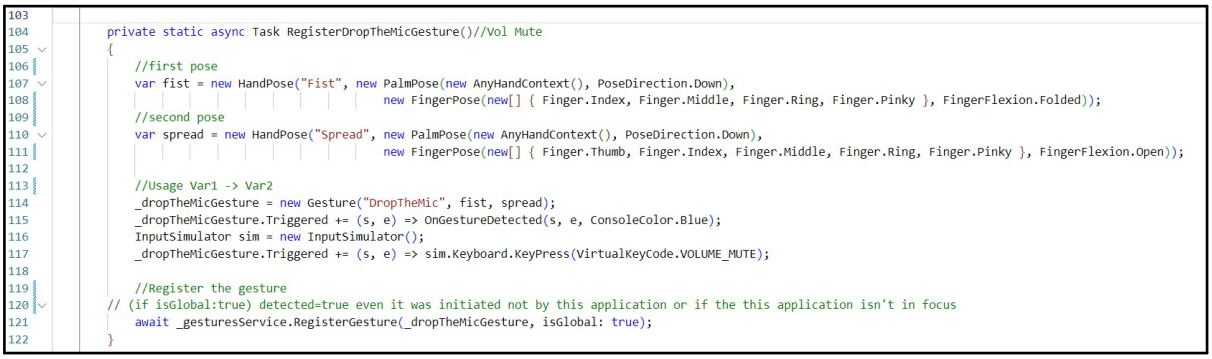
Spread:
En el contexto de la mano, sin importar si está cerrada o abierta, la dirección hacia la que debe estar orientada la palma es hacia abajo.
Para la posición de los dedos, el pulgar, índice, medio, anular y meñique deben estar Abiertos.

Se define el gesto utilizando el orden a seguir para la detección:
_dropTheMicGesture = new Gesture("DropTheMic", fist, spread);

Se llama a la función OnGestureDetected, se inicializa la libreria Input Simulator, al detectarse el gesto, se simula una pulsacion en la tecla virtual de Silenciar volumen (VirtualKeyCode.VOLUME_MUTE).

Se registra el uso del gesto y se declara que funcione aunque la ventana no sea la principal o esté abierta:
await _gesturesService.RegisterGesture(_dropTheMicGesture :, isGlobal: true);

Figura 9 - Silenciar Volumen, Media Control



```
103
104         private static async Task RegisterDropTheMicGesture()//Vol Mute
105         {
106             //first pose
107             var fist = new HandPose("Fist", new PalmPose(new AnyHandContext(), PoseDirection.Down),
108                                             new FingerPose(new[] { Finger.Index, Finger.Middle, Finger.Ring, Finger.Pinky }, FingerFlexion.Folded));
109             //second pose
110             var spread = new HandPose("Spread", new PalmPose(new AnyHandContext(), PoseDirection.Down),
111                                             new FingerPose(new[] { Finger.Thumb, Finger.Index, Finger.Middle, Finger.Ring, Finger.Pinky }, FingerFlexion.Open));
112
113             //Usage Var1 -> Var2
114             _dropTheMicGesture = new Gesture("DropTheMic", fist, spread);
115             _dropTheMicGesture.Triggered += (s, e) => OnGestureDetected(s, e, ConsoleColor.Blue);
116             InputSimulator sim = new InputSimulator();
117             _dropTheMicGesture.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VOLUME_MUTE);
118
119             //Register the gesture
120         // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
121             await _gesturesService.RegisterGesture(_dropTheMicGesture, isGlobal: true);
122         }
```

3.2.4. Siguiente Medio

Las variables 'hold' y 'rotate' se están utilizando para almacenar configuraciones distintas en otro gesto, lo que implica que están siendo reutilizadas para definir las poses de mano en ese nuevo gesto.

Hold:
El dedo pulgar e índice están estirados hacia adelante, apuntando hacia la cámara.
Los dedos medio, anular y meñique están estirados.
El dedo índice se encuentra encima del pulgar y no están tocándose entre sí.

Rotate:
Los dedos pulgar e índice están estirados hacia adelante, apuntando a la cámara.
Los dedos medio, anular y meñique están estirados.
El dedo índice se encuentra a la derecha del pulgar y no están tocándose entre sí.

Se define el gesto utilizando el mismo orden a seguir para la detección:
_RotateRMedia = new Gesture("RotateRMedia", hold, rotate);

Se llama a la función OnGestureDetected, se inicializa la libreria Input Simulator, al detectarse el gesto, se simulan una pulsacion de la tecla virtual Siguiente Medio(VirtualKeyCode.MEDIA_NEXT_TRACK);

Se registra el uso del gesto y se declara que funcione aunque la ventana no sea la principal o esté abierta:
await _gesturesService.RegisterGesture(_RotateRMedia, isGlobal: true); (editado)

Figura 10 – Siguiente Medio, Media Control

```
124         private static async Task RegisterRotateRMedia()//Media Next
125         {
126             //first pose
127             var hold = new HandPose("Hold", new FingerPose(new[] { Finger.Thumb, Finger.Index }, FingerFlexion.Open, PoseDirection.Forward),
128                                             new FingerPose(new[] { Finger.Middle, Finger.Ring, Finger.Pinky }, FingerFlexion.Open),
129                                             new FingertipDistanceRelation(Finger.Index, RelativeDistance.NotTouching, Finger.Thumb),
130                                             new FingertipPlacementRelation(Finger.Index, RelativePlacement.Above, Finger.Thumb));
131             //second pose
132             var rotate = new HandPose("Rotate", new FingerPose(new[] { Finger.Thumb, Finger.Index }, FingerFlexion.Open, PoseDirection.Forward),
133                                             new FingerPose(new[] { Finger.Middle, Finger.Ring, Finger.Pinky }, FingerFlexion.Open),
134                                             new FingertipDistanceRelation(Finger.Index, RelativeDistance.NotTouching, Finger.Thumb),
135                                             new FingertipPlacementRelation(Finger.Index, RelativePlacement.Right, Finger.Thumb));
136
137             //Usage Var1 -> Var2
138             _RotateRMedia = new Gesture("RotateRMedia", hold, rotate);
139             _RotateRMedia.Triggered += (s, e) => OnGestureDetected(s, e, ConsoleColor.Red);
140             InputSimulator sim = new InputSimulator();
141             _RotateRMedia.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.MEDIA_NEXT_TRACK);
142
143
144              //Register the gesture
145         // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
146             await _gesturesService.RegisterGesture(_RotateRMedia, isGlobal: true);
147         }
```

3.2.5. Anterior Medio

Las variables 'hold' y 'rotate' se están utilizando para almacenar configuraciones distintas en otro gesto, lo que implica que están siendo reutilizadas para definir las poses de mano en ese nuevo gesto.

Hold:
El dedo pulgar e índice están estirados hacia adelante, apuntando hacia la cámara.
Los dedos medio, anular y meñique están estirados.
El dedo índice se encuentra encima del pulgar y no están tocándose entre sí.

Rotate:
Los dedos pulgar e índice están estirados hacia adelante, apuntando a la cámara.
Los dedos medio, anular y meñique están estirados.
El dedo índice se encuentra a la izquierda del pulgar y no están tocándose entre sí.

Se define el gesto utilizando el mismo orden a seguir para la detección:
_RotateLMedia = new Gesture("RotateLMedia", hold, rotate);

Se llama a la función OnGestureDetected, se inicializa la libreria Input Simulator, al detectarse el gesto, se simula una pulsacion de la tecla virtual Anterior Medio((VirtualKeyCode.MEDIA_PREV_TRACK);

Se registra el uso del gesto y se declara que funcione aunque la ventana no sea la principal o esté abierta:
await _gesturesService.RegisterGesture(_RotateLMedia, isGlobal: true); (editado)

Figura 11 – Anterior Medio, Media Control

```
149     private static async Task RegisterRotateLMedia()//Media Prev
150     {
151         //first pose
152         var hold = new HandPose("Hold", new FingerPose(new[] { Finger.Thumb }, FingerFlexion.Open, PoseDirection.Forward),
153                                         new FingerPose(new[] { Finger.Middle, Finger.Ring, Finger.Pinky }, FingerFlexion.Open),
154                                         new FingertipDistanceRelation(Finger.Index, RelativeDistance.NotTouching, Finger.Thumb),
155                                         new FingertipPlacementRelation(Finger.Index, RelativePlacement.Above, Finger.Thumb));
156         //second pose
157         var rotate = new HandPose("Rotate", new FingerPose(new[] { Finger.Thumb, Finger.Index }, FingerFlexion.Open, PoseDirection.Forward),
158                                         new FingerPose(new[] { Finger.Middle, Finger.Ring, Finger.Pinky }, FingerFlexion.Open),
159                                         new FingertipDistanceRelation(Finger.Index, RelativeDistance.NotTouching, Finger.Thumb),
160                                         new FingertipPlacementRelation(Finger.Index, RelativePlacement.Left, Finger.Thumb));
161
162
163
164         //Usage Var1 -> Var2
165         _RotateLMedia = new Gesture("RotateLMedia", hold, rotate);
166         _RotateLMedia.Triggered += (s, e) => OnGestureDetected(s, e, ConsoleColor.DarkRed);
167         InputSimulator sim = new InputSimulator();
168         _RotateLMedia.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.MEDIA_PREV_TRACK);
169
170
171          //Register the gesture
172     // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
173         await _gesturesService.RegisterGesture(_RotateLMedia, isGlobal: true);
174     }
```

3.2.6. Reproducir/Pausa

Las variables 'iddle' y 'RockOn' se están utilizando para almacenar configuraciones distintas en otro gesto, lo que implica que están siendo reutilizadas para definir las poses de mano en ese nuevo gesto.

Iddle:
Todos los dedos están doblados hacia adentro.

RockOn:
Los dedos medio, anular y pulgar están doblados.
La yema del dedo medio está tocando la yema del dedo anular.
La yema del pulgar está tocando la yema del dedo anular.
El dedo índice se encuentra encima del pulgar.
El dedo índice y el meñique están estirados.

Se define el gesto utilizando el mismo orden a seguir para la detección:
_RockGesture = new Gesture("RockOnGesture", Iddle, RockOn);

Se llama a la función OnGestureDetected, se inicializa la libreria Input Simulator, al detectarse el gesto, se simula una pulsacion de la tecla virtual Play/Pause(KeyPress(VirtualKeyCode.MEDIA_PLAY_PAUSE);

Se registra el uso del gesto y se declara que funcione aunque la ventana no sea la principal o esté abierta:\
await _gesturesService.RegisterGesture(_RockGesture, isGlobal: true);

Figura 12 – Reproducir/Pausar, Media Control

```
176     private static async Task RegisterRockOnGesture()//Play/Pause
177     {
178         //first pose
179         var Iddle = new HandPose("iddle", new FingerPose(new AllFingersContext(), FingerFlexion.Folded));
180
181         //second pose
182         var RockOn = new HandPose("Rock", new FingerPose(new[] { Finger.Middle, Finger.Ring, Finger.Thumb }, FingerFlexion.Folded),
183                                           new FingertipDistanceRelation(Finger.Middle, RelativeDistance.Touching, Finger.Ring),
184                                           new FingertipDistanceRelation(Finger.Thumb, RelativeDistance.Touching, Finger.Ring),
185                                           new FingertipPlacementRelation(Finger.Index, RelativePlacement.Above, Finger.Thumb),
186                                           new FingerPose(new[] { Finger.Index, Finger.Pinky }, FingerFlexion.Open));
187
188
189         //Usage Var1 -> Var2
190         _RockGesture = new Gesture("RockOnGesture", Iddle, RockOn);
191         _RockGesture.Triggered += (s, e) => OnGestureDetected(s, e, ConsoleColor.DarkMagenta);
192         InputSimulator sim = new InputSimulator();
193         _RockGesture.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.MEDIA_PLAY_PAUSE);
194
195         //Register the gesture
196         // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
197         await _gesturesService.RegisterGesture(_RockGesture, isGlobal: true);
198     }
```

3.2.7. Interfaz del Módulo

Figura 13 – Interfaz de Modulo, Media Control

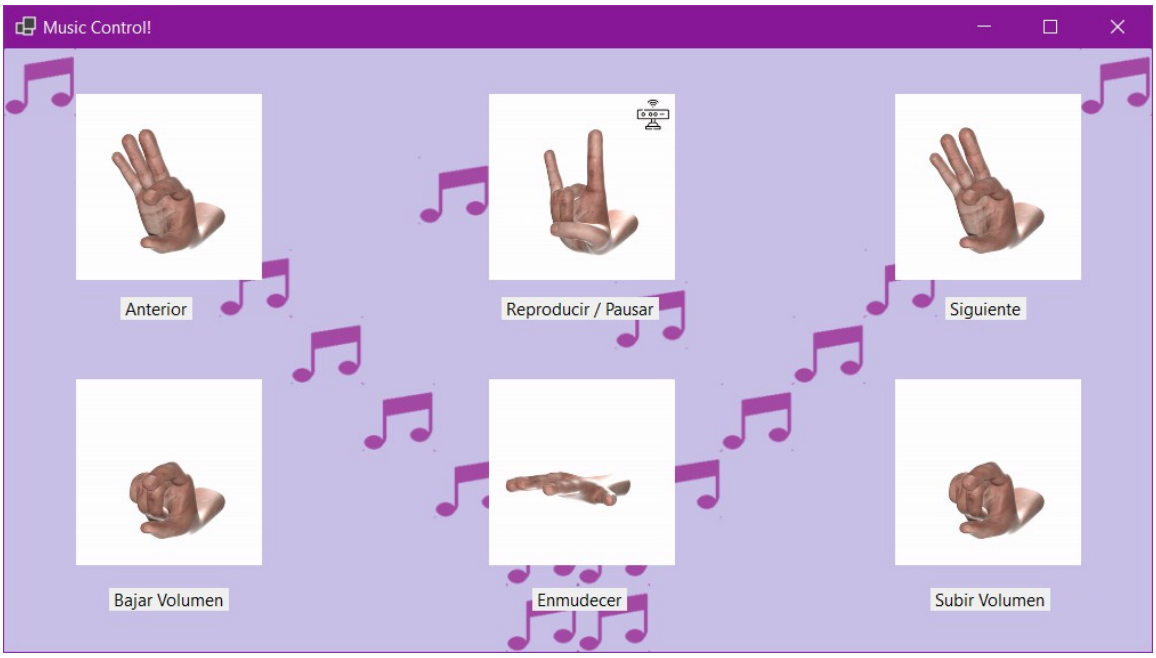
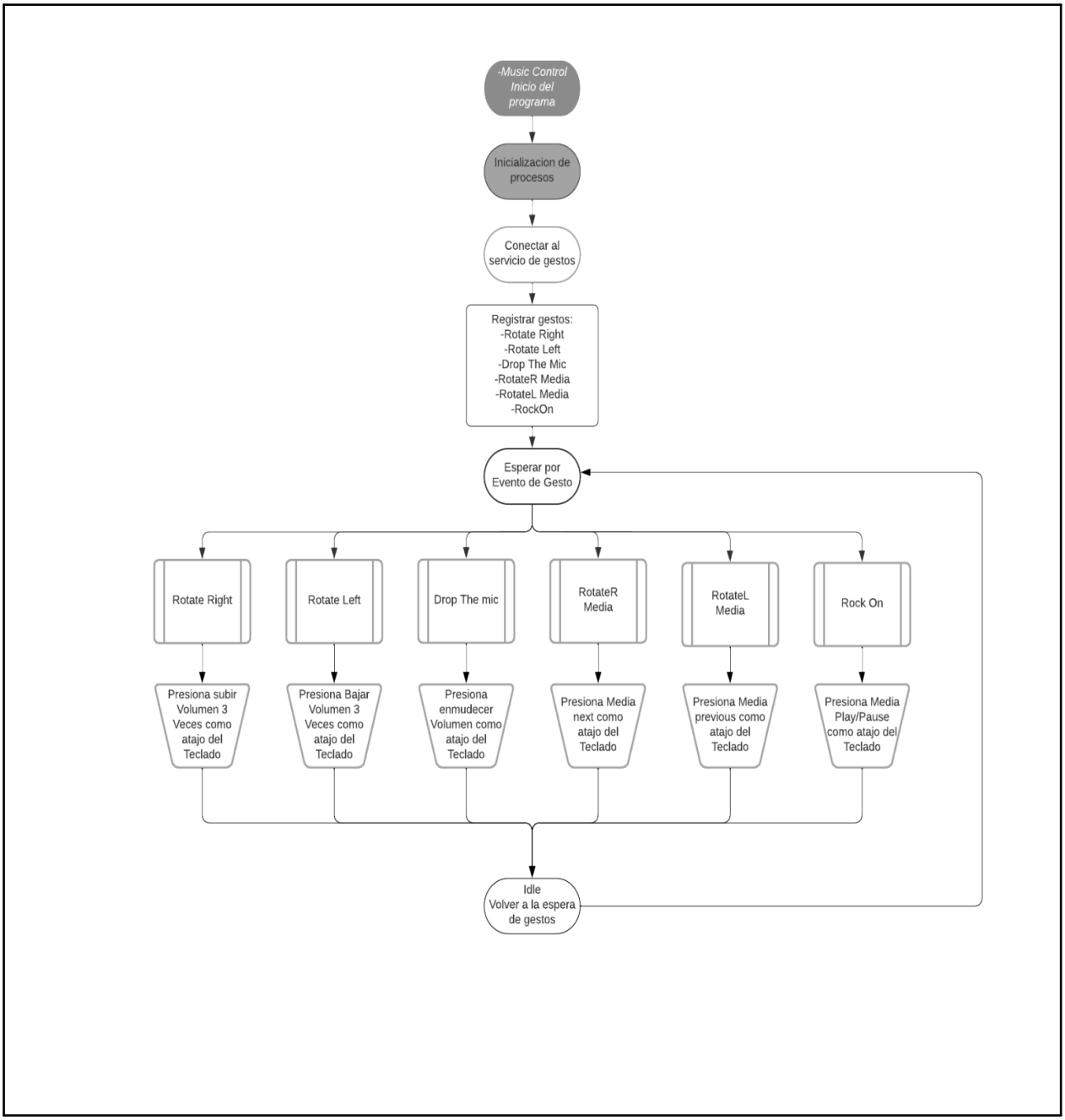


Figura 14 – Diagrama de Función, Media Control



3.3. Slideshow Control

Estas líneas de código importan las bibliotecas del sistema y otras bibliotecas específicas para la gestión de aplicaciones gráficas, el reconocimiento de gestos y la utilización de un teclado virtual.

- System: Proporciona las clases fundamentales y estructuras básicas de datos en el entorno .NET.
- System.Threading.Tasks: Proporciona clases y estructuras que admiten la ejecución de tareas asincrónicas en .NET.
- System.Windows.Forms: Proporciona clases para crear y administrar aplicaciones de Windows Forms, que son aplicaciones de escritorio con interfaces gráficas de usuario (GUI).
- Microsoft.Gestures: Proporciona funcionalidades para reconocimiento de gestos.
- Microsoft.Gestures.Endpoint: Proporciona un punto final para interactuar con el servicio de reconocimiento de gestos.
- WindowsInput: Proporciona clases para simular entrada de teclado y mouse en Windows.
- WindowsInput.Native: Proporciona tipos que representan las teclas físicas en un teclado, facilitando la simulación de pulsaciones de teclas específicas.

Figura 15 – Librerías Necesarias, Slideshow Control

```
1   using System;
2   using System.Threading.Tasks;
3   using System.Windows.Forms;
4   using Microsoft.Gestures;
5   using Microsoft.Gestures.Endpoint;
6   using WindowsInput;
7   using WindowsInput.Native;
8
```

Estas líneas de código establecen el uso de un servicio encargado de la gestión de gestos, así como la definición de los propios gestos a ser utilizados en la aplicación.

Figura 16 - Servicios encargados de la gestión de gestos, Slideshow Control

```
public partial class PPForm : Form
{

    private static GesturesServiceEndpoint _gesturesService;
    private static Gesture _PointGestureR;//Anterior diapositiva
    private static Gesture _PointGestureL;//Siguiente Diapositiva
    private static Gesture _ZoomInGesture;//Acercar Diapositiva
    private static Gesture _ZoomOutGesture;//Alejar Diapositiva
    private static Gesture _ZoomToFitGesture;//Ajustar Zoom
```

Estas líneas de código inicializan las variables predeterminadas en el constructor y establecen la conexión con el servicio encargado de manejar los gestos, el cual está alojado localmente en el localhost.

Se pone en estado de espera al servicio encargado de la gestión de gestos.

Figura 17 – En espera del servicio de la gestión de gestos, Slideshow Control

```
26      public PPForm()
27      {
28          InitializeComponent();
29          InitializeGestures();
30      }
31
32
33      private async void InitializeGestures()
34      {
35          //Connect to service
36          var gesturesServiceHostName = "localhost";
37          await RegisterGestures(gesturesServiceHostName);
38      }
```

Estas líneas de código establecen la conexión al servicio encargado de manejar los gestos y comienzan la espera de que los gestos sean activados o "triggered"

Figura 18 – Servicio encargado de la gestion de gestos, Slideshow Control

```
40      private static async Task RegisterGestures(string gesturesServiceHostName)
41      {
42
43          //Connect to service
44          _gesturesService = GesturesServiceEndpointFactory.Create(gesturesServiceHostName);
45          await _gesturesService.ConnectAsync();
46
47          //custom Gestures
48          await RegisterPointRightGesture();//Anterior Diapositiva
49          await RegisterPointLeftGesture();//Siguiente Diapositiva
50          await RegisterZoomInGesture();//Acercar Diapositiva
51          await RegisterZoomOutGesture();//Alejar Diapositiva
52          await RegisterZoomToFitGesture();//Ajustar Zoom
53
54      }
```

3.3.1. Anterior Diapositiva

En este fragmento de código se definen la variables 'Front' y 'Right' que representan la primera y segunda pose a tener en cuenta para la detección del gesto.
Front:
Todos los dedos, excepto el índice, están doblados hacia adentro.
El índice apunta hacia adelante.

Right:
Los dedos medio, anular y meñique están doblados.
El pulgar está estirado y apuntando hacia arriba.
El índice está extendido hacia la derecha.

Se define el gesto utilizando las posiciones de mano previamente definidas y se especifica el orden a seguir para la detección:
_PointGestureR = new Gesture("PointRight", front, right);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual hacia la izquierda (VirtualKeyCode.LEFT).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_PointGestureR, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 19 – Anterior Diapositiva, Slideshow Control

```
57          private static async Task RegisterPointRightGesture()//Anterior diapositiva
58          {
59              //first pose
60              var front = new HandPose("Front", new FingerPose(new[] { Finger.Middle, Finger.Ring, Finger.Pinky, Finger.Thumb }, FingerFlexion.Folded),
61                                               new FingerPose(Finger.Index, PoseDirection.Forward));
62              //second pose
63              var right = new HandPose("Right", new FingerPose(new[] { Finger.Middle, Finger.Ring, Finger.Pinky }, FingerFlexion.Folded),
64                                               new FingerPose(Finger.Thumb, FingerFlexion.Open, PoseDirection.Up),
65                                               new FingerPose(Finger.Index, PoseDirection.Right));
66
67              //Usage front -> right
68              _PointGestureR = new Gesture("PointRight", front, right);
69              InputSimulator sim = new InputSimulator();
70              _PointGestureR.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.LEFT);
71
72              //Register the gesture
73              // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
74              await _gesturesService.RegisterGesture(_PointGestureR, isGlobal: true);
75          }
76
```

3.3.2. Siguiente Diapositiva

Las variables 'Front' y 'Left' se están utilizando para almacenar configuraciones distintas en otro gesto, lo que implica que están siendo reutilizadas para definir las poses de mano en ese nuevo gesto.

Front:
Todos los dedos, excepto el índice, están doblados hacia adentro.
El índice apunta hacia adelante.

Left:
Los dedos medio, anular y meñique están doblados hacia adentro.
El pulgar está estirado y apuntando hacia arriba.
El índice está extendido hacia la izquierda.

Se define el gesto utilizando las posiciones de mano previamente definidas y se especifica el orden a seguir para la detección:
_PointGestureL = new Gesture("PointLeft", front, left);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual hacia la derecha (VirtualKeyCode.RIGHT).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_PointGestureL, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 20 – Siguiente Diapositiva

```
78      private static async Task RegisterPointLeftGesture()//Siguiente Diapositiva
79      {
80          //first pose
81          var front = new HandPose("Front", new FingerPose(new[] { Finger.Middle, Finger.Ring, Finger.Pinky, Finger.Thumb }, FingerFlexion.Folded),
82                                            new FingerPose(Finger.Index, PoseDirection.Forward));
83          //second pose
84          var left = new HandPose("Left", new FingerPose(new[] { Finger.Middle, Finger.Ring, Finger.Pinky }, FingerFlexion.Folded),
85                                          new FingerPose(Finger.Thumb, FingerFlexion.Open, PoseDirection.Up),
86                                          new FingerPose(Finger.Index, PoseDirection.Left));
87
88          //Usage front -> right
89          _PointGestureL = new Gesture("PointLeft", front, left);
90          InputSimulator sim = new InputSimulator();
91          _PointGestureL.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.RIGHT);
92
93          //Register the gesture
94          // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
95          await _gesturesService.RegisterGesture(_PointGestureL, isGlobal: true);
96      }
97
```

3.3.3. Acercar Diapositiva

Las variables 'Hold' y 'Zoom' se están utilizando para almacenar configuraciones distintas en otro gesto, lo que implica que están siendo reutilizadas para definir las poses de mano en ese nuevo gesto.

Hold:
Con la palma viendo hacia adelante
Los dedos índice, medio y el meñique están estirados y apuntando hacia Arriba.
Los dedos pulgar y anular están doblados hacia adentro.
El pulgar toca el dedo anular.

Zoom:
Con la palma viendo hacia adelante
Los dedos índice, medio y el meñique están estirados y apuntando hacia la Izquierda.
Los dedos pulgar y anular están doblados hacia adentro.
El pulgar toca el dedo anular.

Se define el gesto utilizando las posiciones de mano previamente definidas y se especifica el orden a seguir para la detección:
_ZoomInGesture = new Gesture("ZoomIn", hold, zoomin);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "+" (VirtualKeyCode.OEM_PLUS).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_ZoomInGesture, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 21 – Acercar Diapositiva Slideshow Control

```
98          private static async Task RegisterZoomInGesture()//Acercar Diapositiva
99          {
100             //first pose
101             var hold = new HandPose("Hold", new PalmPose(new AnyHandContext(), PoseDirection.Forward),
102                                             new FingerPose(new[] { Finger.Index, Finger.Pinky, Finger.Middle }, FingerFlexion.Open, PoseDirection.Up),
103                                             new FingerPose(new[] { Finger.Ring, Finger.Thumb }, FingerFlexion.Folded),
104                                             new FingertipDistanceRelation(Finger.Thumb, RelativeDistance.Touching, Finger.Ring));
105             //second pose
106             var zoomin = new HandPose("Zoom", new PalmPose(new AnyHandContext(), PoseDirection.Forward),
107                                             new FingerPose(new[] { Finger.Index, Finger.Pinky, Finger.Middle }, FingerFlexion.Open, PoseDirection.Left),
108                                             new FingerPose(new[] { Finger.Ring, Finger.Thumb }, FingerFlexion.Folded),
109                                             new FingertipDistanceRelation(Finger.Thumb, RelativeDistance.Touching, Finger.Ring));
110
111             //Usage Hold -> Zoom
112             _ZoomInGesture = new Gesture("ZoomIn", hold, zoomin);
113             InputSimulator sim = new InputSimulator();
114             _ZoomInGesture.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.CONTROL, VirtualKeyCode.OEM_PLUS);
115
116             //Register the gesture
117             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
118             await _gesturesService.RegisterGesture(_ZoomInGesture, isGlobal: true);
119         }
120
```

3.3.4. Alejar Diapositiva

Las variables 'Hold' y 'Zoom' se están utilizando para almacenar configuraciones distintas en otro gesto, lo que implica que están siendo reutilizadas para definir las poses de mano en ese nuevo gesto.

Hold:
Con la palma viendo hacia adelante
Los dedos índice, medio y el meñique están estirados y apuntando hacia Arriba.
Los dedos pulgar y anular están doblados hacia adentro.
El pulgar toca el dedo anular.

Zoom:
Con la palma viendo hacia adelante
Los dedos índice, medio y el meñique están estirados y apuntando hacia la Derecha.
Los dedos pulgar y anular están doblados hacia adentro.
El pulgar toca el dedo anular.

Se define el gesto utilizando las posiciones de mano previamente definidas y se especifica el orden a seguir para la detección:
_ZoomOutGesture = new Gesture("ZoomOut", hold, zoomout);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "-" (VirtualKeyCode.OEM_MINUS).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_ZoomOutGesture, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 22 – Alejar Diapositiva, Slideshow Control

```
120
121         private static async Task RegisterZoomOutGesture()//Alejar Diapositiva
122         {
123             //first pose
124             var hold = new HandPose("Hold", new PalmPose(new AnyHandContext(), PoseDirection.Forward),
125                                             new FingerPose(new[] { Finger.Index, Finger.Pinky, Finger.Middle }, FingerFlexion.Open, PoseDirection.Up),
126                                             new FingerPose(new[] { Finger.Ring, Finger.Thumb }, FingerFlexion.Folded),
127                                             new FingertipDistanceRelation(Finger.Thumb, RelativeDistance.Touching, Finger.Ring));
128             //second pose
129             var zoomout = new HandPose("Zoom", new PalmPose(new AnyHandContext(), PoseDirection.Forward),
130                                             new FingerPose(new[] { Finger.Index, Finger.Pinky, Finger.Middle }, FingerFlexion.Open, PoseDirection.Right),
131                                             new FingerPose(new[] { Finger.Ring, Finger.Thumb }, FingerFlexion.Folded),
132                                             new FingertipDistanceRelation(Finger.Thumb, RelativeDistance.Touching, Finger.Ring));
133
134
135             //Usage front -> right
136             _ZoomOutGesture = new Gesture("ZoomOut", hold, zoomout);
137             InputSimulator sim = new InputSimulator();
138             _ZoomOutGesture.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.CONTROL, VirtualKeyCode.OEM_MINUS);
139
140             //Register the gesture
141             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
142             await _gesturesService.RegisterGesture(_ZoomOutGesture, isGlobal: true);
143         }
144
```

3.3.5. Centrar Diapositiva

Las variables 'Hold' se esta utilizando para almacenar la configuracion de este gesto, lo que implica que solo se está usando una variable en este gesto.

Hold:
El pulgar, el anular y el medio están estirados y apuntando hacia adelante.
El índice y el meñique están estirados hacia arriba.
El pulgar se encuentra debajo del medio.

Se define el gesto utilizando la posicion de mano previamente definida:
_ZoomToFitGesture = new Gesture("ZoomFit", hold);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula secuencia de pulsaciones en la tecla virtuales Control Izquierdo, Alt y letra O "LControl" + "Alt" + "O" (VirtualKeyCode.LCONTROL, VirtualKeyCode.MENU, VirtualKeyCode.VK_O).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_ZoomToFitGesture, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 23 – Centrar Diapositiva, Slideshow Control

```
145
146         private static async Task RegisterZoomToFitGesture()//Ajustar Zoom
147         {
148             //first pose
149             var hold = new HandPose("Hold", new FingerPose(new[] { Finger.Thumb, Finger.Ring, Finger.Middle }, FingerFlexion.Open, PoseDirection.Forward),
150                                             new FingerPose(new[] { Finger.Index, Finger.Pinky }, FingerFlexion.Open, PoseDirection.Up),
151                                             new FingertipDistanceRelation(Finger.Thumb, RelativeDistance.Touching, Finger.Middle));
152
153
154             //Usage only Hold
155             _ZoomToFitGesture = new Gesture("ZoomFit", hold);
156             InputSimulator sim = new InputSimulator();
157             _ZoomToFitGesture.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.LCONTROL, VirtualKeyCode.MENU, VirtualKeyCode.VK_O);
158
159             //Register the gesture
160             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
161             await _gesturesService.RegisterGesture(_ZoomToFitGesture, isGlobal: true);
162         }
```

3.3.6. Interfaz de Usuario

Figura 24 – Interfaz del Modulo

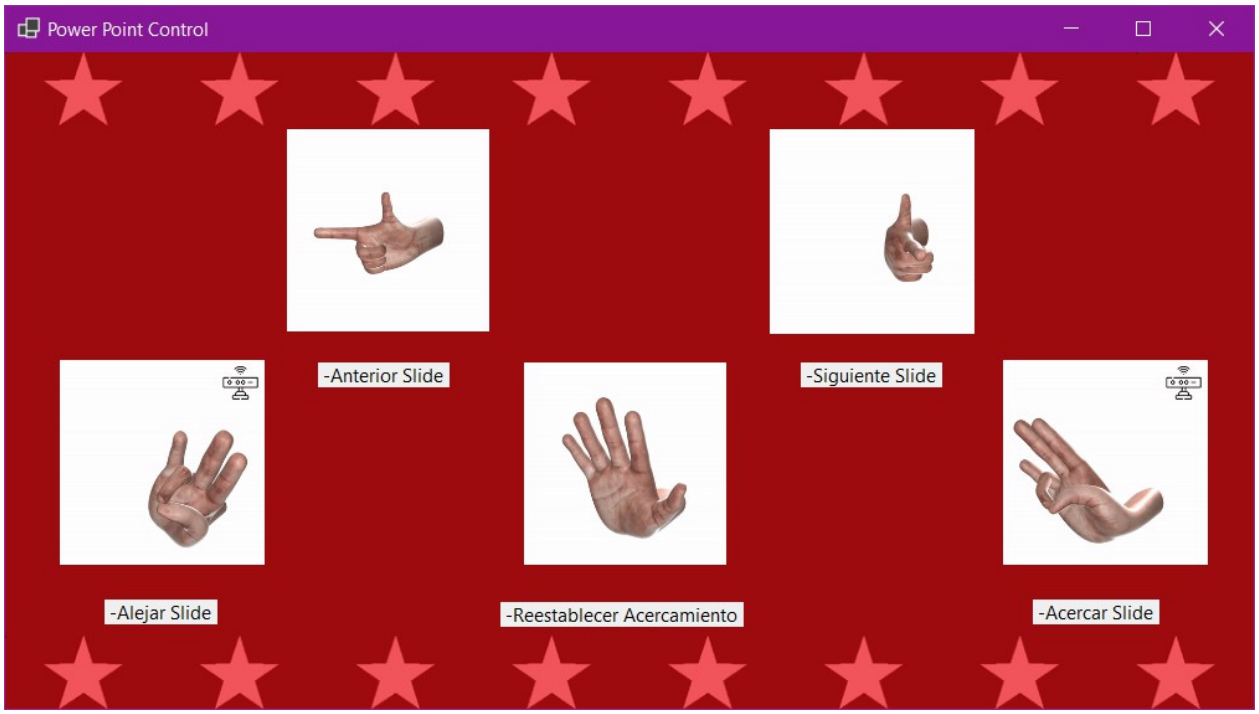
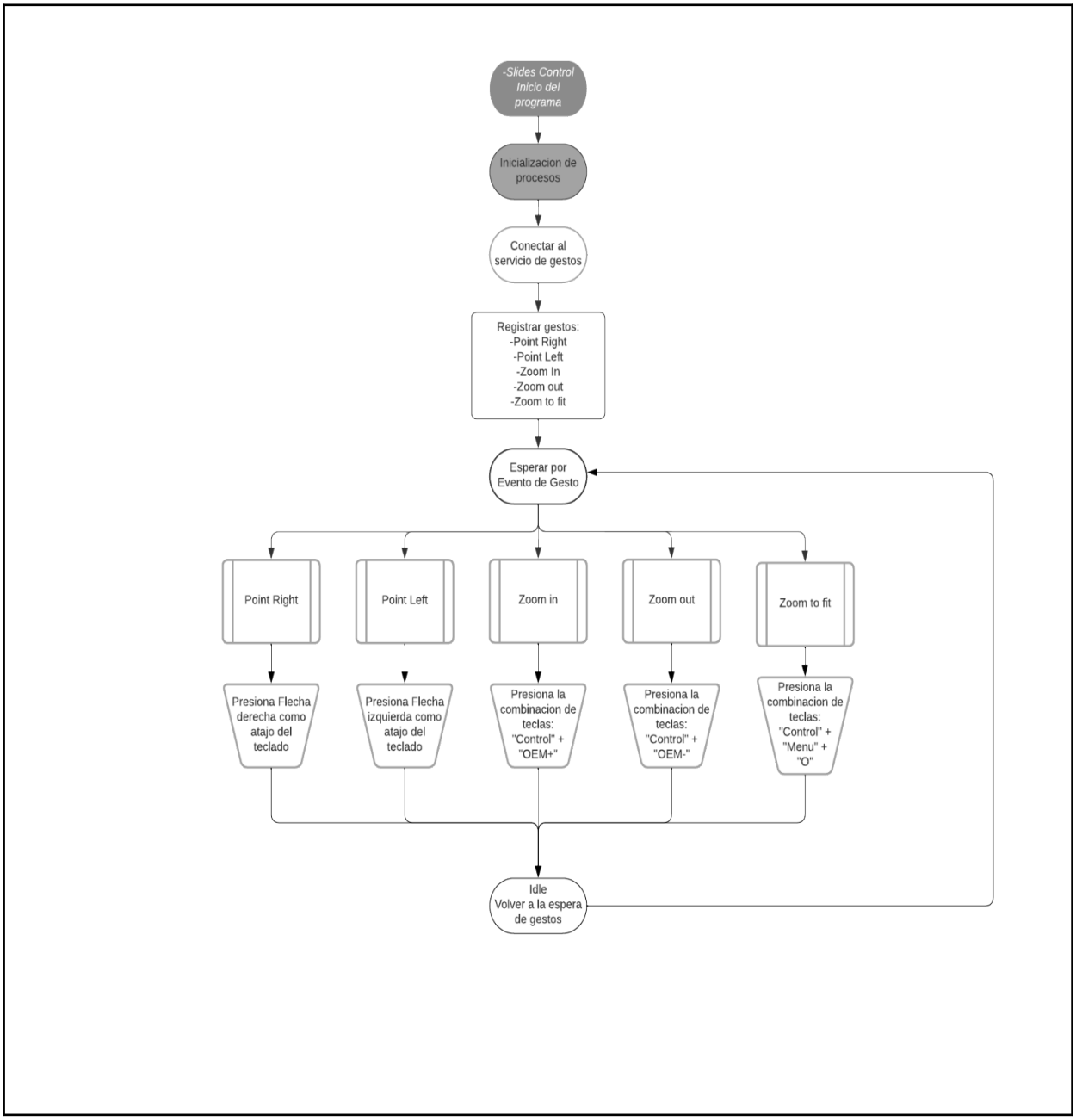


Figura 25 – Diagrama de función, Slideshow Control



3.4. Alphabet Control

Estas líneas de código importan las bibliotecas del sistema y otras bibliotecas específicas para la gestión de aplicaciones gráficas, el reconocimiento de gestos y la utilización de un teclado virtual.

- System: Proporciona las clases fundamentales y estructuras básicas de datos en el entorno .NET.
- System.Drawing: Proporciona funcionalidades para la creación y manipulación de gráficos, imágenes y dibujos en aplicaciones Windows Forms.
- System.Threading.Tasks: Proporciona clases y estructuras que admiten la ejecución de tareas asincrónicas en .NET.
- System.Windows.Forms: Proporciona clases para crear y administrar aplicaciones de Windows Forms, que son aplicaciones de escritorio con interfaces gráficas de usuario (GUI).
- Microsoft.Gestures: Proporciona funcionalidades para el reconocimiento de gestos.
- Microsoft.Gestures.Endpoint: Proporciona un punto final para interactuar con el servicio de reconocimiento de gestos.
- WindowsInput: Proporciona clases para simular entrada de teclado y ratón en Windows.
- WindowsInput.Native: Proporciona tipos que representan las teclas físicas en un teclado, facilitando la simulación de pulsaciones de teclas específicas.

Figura 26 – Librerias necesarias Alphabet control

```
1   using System;
2   using System.Drawing;
3   using System.Threading.Tasks;
4   using System.Windows.Forms;
5   using Microsoft.Gestures;
6   using Microsoft.Gestures.Endpoint;
7   using WindowsInput;
8   using WindowsInput.Native;
9
```

Estas líneas de código inicializan una serie de variables estáticas que representan gestos de letras del alfabeto, junto con la variable RutaArch que almacena la ruta de un directorio de recursos.

Las variables _gesturesService, _Letter_A hasta _Letter_Z representan respectivamente el servicio de gestos y gestos individuales para cada letra del alfabeto. Es estas variables están asociados a reconocer gestos de letras realizados por el usuario.

La variable RutaArch almacena la ruta de un directorio de recursos, que probablemente se utilice para acceder a recursos externos, como imágenes, archivos de sonido u otros archivos necesarios para la aplicación o proyecto.

Figura 27 – Servicios encargados de la gestion de gestos, Alphabet Control

```
12  public partial class AlphaForm : Form
13  {
14      public string RutaArch = "C:\\Users\\mitch\\Source\\Repos\\DJxRott\\KinectProject\\AlphaControlForm\\Resources\\";
15
16      private static GesturesServiceEndpoint _gesturesService;
17      private static Gesture _Letter_A;
18      private static Gesture _Letter_B;
19      private static Gesture _Letter_C;
20      private static Gesture _Letter_D;
21      private static Gesture _Letter_E;
22      private static Gesture _Letter_F;
23      private static Gesture _Letter_G;
24      private static Gesture _Letter_H;
25      private static Gesture _Letter_I;
26      private static Gesture _Letter_J;
27      private static Gesture _Letter_K;
28      private static Gesture _Letter_L;
29      private static Gesture _Letter_M;
30      private static Gesture _Letter_N;
31      private static Gesture _Letter_O;
32      private static Gesture _Letter_P;
33      private static Gesture _Letter_Q;
34      private static Gesture _Letter_R;
35      private static Gesture _Letter_S;
36      private static Gesture _Letter_T;
37      private static Gesture _Letter_U;
38      private static Gesture _Letter_V;
39      private static Gesture _Letter_W;
40      private static Gesture _Letter_X;
41      private static Gesture _Letter_Y;
42      private static Gesture _Letter_Z;
```

Estas líneas de código inicializan las variables predeterminadas en el constructor y establecen la conexión con el servicio encargado de manejar los gestos, el cual está alojado localmente en el localhost.

Se pone en estado de espera al servicio encargado de la gestión de gestos.

Figura 28 – En espera del servicio de la gestion de gestos, Alphabet Control

```
45          public AlphaForm()
46          {
47              InitializeComponent();
48              InitializeGestures();
49          }
50
51          private async void InitializeGestures()
52          {
53              //Inicialize the service
54              var gesturesServiceHostName = "localhost";
55              await RegisterGestures(gesturesServiceHostName);
56          }
57
```

Estas líneas de código asignan un controlador de eventos MouseEnter a cada botón BT_A, BT_B, BT_C, ..., BT_Z. Cada vez que el cursor del mouse entra en el área del botón correspondiente, se activará el evento asociado y se ejecutará el código contenido en el método.

Figura 29 – Servicio Encargado de la Gestion de Gestos, Alphabet Control

```
public void Form1_Load(object sender, EventArgs e)
{
    this.Text = "Music Control!";

    ////////////////////////////////////////////// Mouse Enter Event//////////////////////
    BT_A.MouseEnter += new EventHandler(BT_A_MouseEnter);
    BT_B.MouseEnter += new EventHandler(BT_B_MouseEnter);
    BT_C.MouseEnter += new EventHandler(BT_C_MouseEnter);
    BT_D.MouseEnter += new EventHandler(BT_D_MouseEnter);
    BT_E.MouseEnter += new EventHandler(BT_E_MouseEnter);
    BT_F.MouseEnter += new EventHandler(BT_F_MouseEnter);
    BT_G.MouseEnter += new EventHandler(BT_G_MouseEnter);
    BT_H.MouseEnter += new EventHandler(BT_H_MouseEnter);
    BT_I.MouseEnter += new EventHandler(BT_I_MouseEnter);
    BT_J.MouseEnter += new EventHandler(BT_J_MouseEnter);
    BT_K.MouseEnter += new EventHandler(BT_K_MouseEnter);
    BT_L.MouseEnter += new EventHandler(BT_L_MouseEnter);
    BT_M.MouseEnter += new EventHandler(BT_M_MouseEnter);
    BT_N.MouseEnter += new EventHandler(BT_N_MouseEnter);
    BT_O.MouseEnter += new EventHandler(BT_O_MouseEnter);
    BT_P.MouseEnter += new EventHandler(BT_P_MouseEnter);
    BT_Q.MouseEnter += new EventHandler(BT_Q_MouseEnter);
    BT_R.MouseEnter += new EventHandler(BT_R_MouseEnter);
    BT_S.MouseEnter += new EventHandler(BT_S_MouseEnter);
    BT_T.MouseEnter += new EventHandler(BT_T_MouseEnter);
    BT_U.MouseEnter += new EventHandler(BT_U_MouseEnter);
    BT_V.MouseEnter += new EventHandler(BT_V_MouseEnter);
    BT_W.MouseEnter += new EventHandler(BT_W_MouseEnter);
    BT_X.MouseEnter += new EventHandler(BT_X_MouseEnter);
    BT_Y.MouseEnter += new EventHandler(BT_Y_MouseEnter);
    BT_Z.MouseEnter += new EventHandler(BT_Z_MouseEnter);
}
```

Esta línea de código establece la conexión con el servicio de gestos.
_gesturesService = GesturesServiceEndpointFactory.Create(gesturesServiceHostName);

Luego, se registran gestos personalizados llamando a métodos específicos para cada letra del alfabeto, como RegLetter_A(), RegLetter_B(), ..., RegLetter_Z(). Cada método se encarga de registrar un gesto específico asociado a la letra correspondiente.

Figura 30 – Servicio a la espera de la escritura de cada letra , Alphabet Control

```
93          private static async Task RegisterGestures(string gesturesServiceHostName)
94          {
95              //Connect to service
96              _gesturesService = GesturesServiceEndpointFactory.Create(gesturesServiceHostName);
97              await _gesturesService.ConnectAsync();
98
99              //custom Gestures
100             await RegLetter_A();
101             await RegLetter_B();
102             await RegLetter_C();
103             await RegLetter_D();
104             await RegLetter_E();
105             await RegLetter_F();
106             await RegLetter_G();
107             await RegLetter_H();
108             await RegLetter_I();
109             await RegLetter_J();
110             await RegLetter_K();
111             await RegLetter_L();
112             await RegLetter_M();
113             await RegLetter_N();
114             await RegLetter_O();
115             await RegLetter_P();
116             await RegLetter_Q();
117             await RegLetter_R();
118             await RegLetter_S();
119             await RegLetter_T();
120             await RegLetter_U();
121             await RegLetter_V();
122             await RegLetter_W();
123             await RegLetter_X();
124             await RegLetter_Y();
125             await RegLetter_Z();
```

3.4.1. Letra A

Se usa la variable Iddle para definir el gesto a detectar.

Iddle:

La palma de la mano está orientada hacia adelante.
El índice, medio, anular y meñique están doblados hacia abajo.
El pulgar y el índice no están tocándose.
El pulgar está estirado hacia arriba.

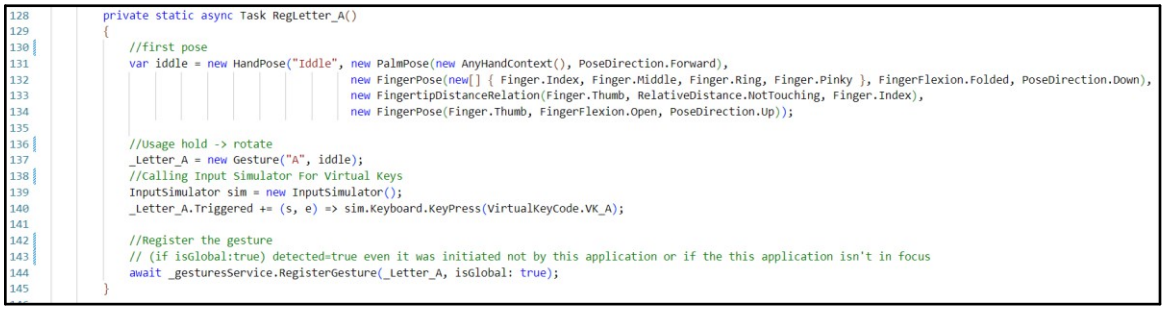
Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_A = new Gesture("A", iddle);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "A" (VirtualKeyCode.VK_A).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:

await _gesturesService.RegisterGesture(_Letter_A, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 31 – Letra A, Alphabet Control



```
private static async Task RegLetter_A()
{
    //first pose
    var iddle = new HandPose("Iddle", new PalmPose(new AnyHandContext(), PoseDirection.Forward),
                                      new FingerPose(new[] { Finger.Index, Finger.Middle, Finger.Ring, Finger.Pinky }, FingerFlexion.Folded, PoseDirection.Down),
                                      new FingertipDistanceRelation(Finger.Thumb, RelativeDistance.NotTouching, Finger.Index),
                                      new FingerPose(Finger.Thumb, FingerFlexion.Open, PoseDirection.Up));

    //Usage hold -> rotate
    _Letter_A = new Gesture("A", iddle);
    //Calling Input Simulator For Virtual Keys
    InputSimulator sim = new InputSimulator();
    _Letter_A.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_A);

    //Register the gesture
    // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
    await _gesturesService.RegisterGesture(_Letter_A, isGlobal: true);
}
```

3.4.2. Letra B

Se usa la variable Iddle para definir el gesto a detectar.

Hold:

El pulgar está doblado hacia la derecha.
El índice, medio, anular y meñique están estirados hacia arriba.
El índice toca al medio.
El medio toca al anular.
El anular toca al meñique.

Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_B = new Gesture("B", hold);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "B" (VirtualKeyCode.VK_B).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_B, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 32 – Letra B, Alphabet Control

```
private static async Task RegLetter_B()
{
    //first pose
    var hold = new HandPose("Hold", new FingerPose(Finger.Thumb, FingerFlexion.Folded, PoseDirection.Right),
                                    new FingerPose(new[] { Finger.Index, Finger.Middle, Finger.Ring, Finger.Pinky }, FingerFlexion.Open, PoseDirection.Up),
                                    new FingertipDistanceRelation(Finger.Index, RelativeDistance.Touching, Finger.Middle),
                                    new FingertipDistanceRelation(Finger.Middle, RelativeDistance.Touching, Finger.Ring),
                                    new FingertipDistanceRelation(Finger.Ring, RelativeDistance.Touching, Finger.Pinky));

    //Usage hold -> rotate
    _Letter_B = new Gesture("B", hold);
    //Calling Input Simulator For Virtual Keys
    InputSimulator sim = new InputSimulator();
    _Letter_B.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_B);

    //Register the gesture
    // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
    await _gesturesService.RegisterGesture(_Letter_B, isGlobal: true);
}
```

3.4.3. Letra C

Se usa la variable Iddle para definir el gesto a detectar.

Fist:
La palma de la mano está orientada hacia adelante.
El índice, medio, anular y meñique están estirados hacia adelante.
El pulgar está estirado hacia adelante.
La yema del dedo índice está colocada encima de la yema del pulgar.
El dedo pulgar no toca los demas dedos.

Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_C = new Gesture("C", Hold);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "C" (VirtualKeyCode.VK_C).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_C, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 33 – Letra C, Alphabet Control

```
166
167         private static async Task RegLetter_C()
168         {
169             //first pose
170             var fist = new HandPose("Fist", new PalmPose(new AnyHandContext(), PoseDirection.Forward),
171                                             new FingerPose(new[] { Finger.Index, Finger.Middle, Finger.Ring, Finger.Pinky }, FingerFlexion.Open, PoseDirection.Forward),
172                                             new FingerPose(Finger.Thumb, FingerFlexion.Open, PoseDirection.Forward),
173                                             new FingertipPlacementRelation(Finger.Index, RelativePlacement.Above, Finger.Thumb));
174
175             //Usage hold -> rotate
176             _Letter_C = new Gesture("C", fist);
177             //Calling Input Simulator For Virtual Keys
178             InputSimulator sim = new InputSimulator();
179             _Letter_C.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_C);
180
181             //Register the gesture
182             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
183             await _gesturesService.RegisterGesture(_Letter_C, isGlobal: true);
184
185         }
```

3.4.4. Letra D

Se usa la variable Fist para definir el gesto a detectar.

Fist:
La palma de la mano está orientada hacia la izquierda.
El dedo índice está estirado hacia arriba.
El pulgar está doblado.
La yema del pulgar toca a los dedos medio y anular.
Los dedos medio, anular y meñique están doblados hacia abajo.

Se define el gesto utilizando la posicion de mano previamente definida:

_Letter_D= new Gesture("D", Fist);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "D" (VirtualKeyCode.VK_D).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_D, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 34 – Letra D, Alphabet Control

```
186
187         private static async Task RegLetter_D()
188         {
189             //first pose
190             var Open = new HandPose("Open", new PalmPose(new AnyHandContext(), PoseDirection.Left),
191                 new FingerPose(Finger.Index, FingerFlexion.Open, PoseDirection.Up),
192                 new FingertipDistanceRelation(Finger.Thumb, RelativeDistance.Touching, new[] { Finger.Middle, Finger.Ring }),
193                 new FingerPose(Finger.Thumb, FingerFlexion.Folded));
194
195             //Usage hold -> rotate
196             _Letter_D = new Gesture("D", Open);
197             //Calling Input Simulator For Virtual Keys
198             InputSimulator sim = new InputSimulator();
199             _Letter_D.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_D);
200
201             //Register the gesture
202             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
203             await _gesturesService.RegisterGesture(_Letter_D, isGlobal: true);
204         }
```

3.4.5. Letra E

Se usa la variable Fist para definir el gesto a detectar.

Fist:
La palma de la mano está orientada hacia adelante.
Los dedos índice, medio, anular y meñique están doblados hacia abajo.
El pulgar está doblado y escondido.
La yema del pulgar toca a los dedos índice, medio, anular y meñique.
La yema del pulgar está colocada debajo de la yema del dedo medio.

Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_E= new Gesture("E", Fist);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "E" (VirtualKeyCode.VK_E).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_E, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 35 – Letra E, Alphabet Control

```
206
207         private static async Task RegLetter_E()
208         {
209             //first pose
210             var fist = new HandPose("Fist", new PalmPose(new AnyHandContext(), PoseDirection.Forward),
211                                             new FingerPose(new[] { Finger.Index, Finger.Middle, Finger.Ring, Finger.Pinky }, PoseDirection.Down),
212                                             new FingerPose(Finger.Thumb, FingerFlexion.FoldedTucked),
213                                             new FingertipDistanceRelation(new[] { Finger.Index, Finger.Middle, Finger.Ring, Finger.Pinky },
214                                             RelativeDistance.Touching, Finger.Thumb),
215                                             new FingertipPlacementRelation(Finger.Thumb, RelativePlacement.Below, Finger.Middle));
216
217             //Usage hold -> rotate
218             _Letter_E = new Gesture("E", fist);
219             //Calling Input Simulator For Virtual Keys
220             InputSimulator sim = new InputSimulator();
221             _Letter_E.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_E);
222
223             //Register the gesture
224             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
225             await _gesturesService.RegisterGesture(_Letter_E, isGlobal: true);
226         }
```

3.4.6. Letra F

Se usa la variable Ok para definir el gesto a detectar.

Ok:

El pulgar está doblado hacia dentro.
La yema del dedo índice toca al pulgar.
La yema del dedo índice está colocada encima del pulgar.
La yema del dedo medio toca al dedo anular.
Los dedos medio, anular y meñique están estirados hacia arriba.

Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_F= new Gesture("F", Ok);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "F" (VirtualKeyCode.VK_F).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_F, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 36 – Letra F, Alphabet Control

```
227
228         private static async Task RegLetter_F()
229         {
230             //first pose
231             var Ok = new HandPose("Oks", new FingerPose(new[] { Finger.Thumb }, FingerFlexion.Folded, PoseDirection.Undefined),
232                                             new FingertipDistanceRelation(Finger.Index, RelativeDistance.Touching, Finger.Thumb),
233                                             new FingertipPlacementRelation(Finger.Index, RelativePlacement.Above, Finger.Thumb),
234                                             new FingertipDistanceRelation(Finger.Middle, RelativeDistance.Touching, Finger.Ring),
235                                             new FingerPose(new[] { Finger.Middle, Finger.Ring, Finger.Pinky }, FingerFlexion.Open, PoseDirection.Up));
236
237             //Usage hold -> rotate
238             _Letter_F = new Gesture("F", Ok);
239             //Calling Input Simulator For Virtual Keys
240             InputSimulator sim = new InputSimulator();
241             _Letter_F.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_F);
242
243             //Register the gesture
244             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
245             await _gesturesService.RegisterGesture(_Letter_F, isGlobal: true);
246         }
247
```

3.4.7. Letra G

Se usa la variable Rock para definir el gesto a detectar.

Rock:
La palma de la mano está orientada hacia atrás.
Los dedos medio, anular y meñique están doblados.
El dedo índice está estirado hacia la izquierda.
El pulgar está estirado hacia arriba.

Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_F= new Gesture("G", RockOn);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "G" (VirtualKeyCode.VK_G).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_G, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 37 – Letra G, Alphabet Control

```
248         private static async Task RegLetter_G()
249         {
250             //first pose
251             var RockOn = new HandPose("Rock", new PalmPose(new AnyHandContext(), PoseDirection.Backward),
252                                               new FingerPose(new[] { Finger.Middle, Finger.Ring, Finger.Pinky }, FingerFlexion.Folded),
253                                               new FingerPose(Finger.Index, FingerFlexion.Open, PoseDirection.Left),
254                                               new FingerPose(Finger.Thumb, FingerFlexion.Open, PoseDirection.Up));
255
256             //Usage hold -> rotate
257             _Letter_G = new Gesture("G", RockOn);
258             //Calling Input Simulator For Virtual Keys
259             InputSimulator sim = new InputSimulator();
260             _Letter_G.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_G);
261
262             //Register the gesture
263             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
264             await _gesturesService.RegisterGesture(_Letter_G, isGlobal: true);
265         }
```

3.4.8. Letra H

Se usa la variable Iddle para definir el gesto a detectar.

Iddle:
La palma de la mano está orientada hacia atrás.
Los dedos medio, anular y meñique están doblados.
El dedo índice está estirado hacia la izquierda.
El pulgar está estirado hacia arriba.

Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_H= new Gesture("H", Iddle);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "H" (VirtualKeyCode.VK_H).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_H, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 38 – Letra H, Alphabet Control

```
266
267         private static async Task RegLetter_H()
268         {
269             //first pose
270             var iddle = new HandPose("Iddle", new PalmPose(new AnyHandContext(), PoseDirection.Backward),
271                                               new FingerPose(new[] { Finger.Index, Finger.Middle }, FingerFlexion.Open, PoseDirection.Left),
272                                               new FingertipDistanceRelation(Finger.Index, RelativeDistance.Touching, Finger.Middle),
273                                               new FingerPose(new[] { Finger.Ring, Finger.Pinky }, FingerFlexion.Folded));
274
275             //Usage hold -> rotate
276             _Letter_H = new Gesture("H", iddle);
277             //Calling Input Simulator For Virtual Keys
278             InputSimulator sim = new InputSimulator();
279             _Letter_H.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_H);
280
281             //Register the gesture
282             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
283             await _gesturesService.RegisterGesture(_Letter_H, isGlobal: true);
284         }
285
```

3.4.9. Letra I

Se usa la variable Iddle para definir el gesto a detectar.

Iddle:

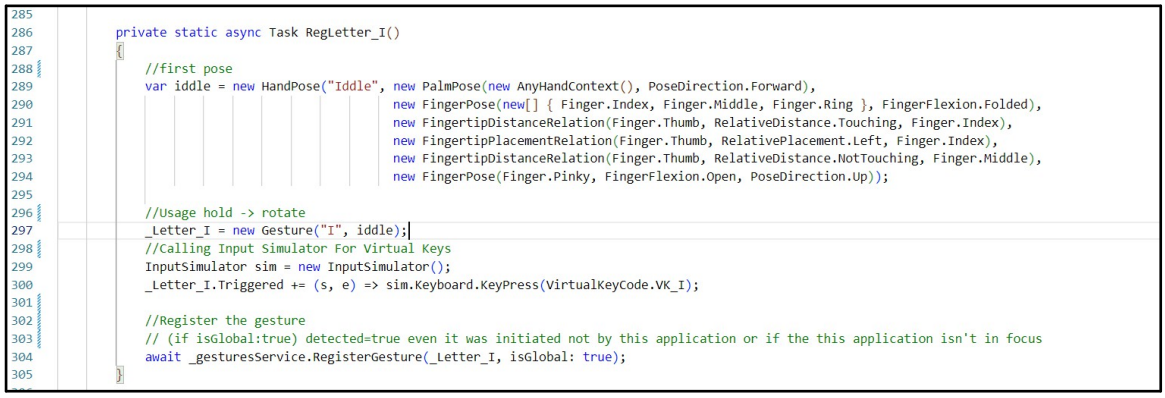
La palma de la mano está orientada hacia adelante.
Los dedos índice, medio y anular están doblados.
La yema del pulgar toca al dedo índice.
La yema del pulgar está colocada a la izquierda de la yema del dedo índice.
La yema del pulgar no toca al dedo medio.
El dedo meñique está estirado hacia arriba.

Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_I= new Gesture("I", Iddle);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "I" (VirtualKeyCode.VK_I).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_I, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano

Figura 39 – Letra I, Alphabet Control



```
285
286         private static async Task RegLetter_I()
287         {
288             //first pose
289             var iddle = new HandPose("Iddle", new PalmPose(new AnyHandContext(), PoseDirection.Forward),
290                                               new FingerPose(new[] { Finger.Index, Finger.Middle, Finger.Ring }, FingerFlexion.Folded),
291                                               new FingertipDistanceRelation(Finger.Thumb, RelativeDistance.Touching, Finger.Index),
292                                               new FingertipPlacementRelation(Finger.Thumb, RelativePlacement.Left, Finger.Index),
293                                               new FingertipDistanceRelation(Finger.Thumb, RelativeDistance.NotTouching, Finger.Middle),
294                                               new FingerPose(Finger.Pinky, FingerFlexion.Open, PoseDirection.Up));
295
296             //Usage hold -> rotate
297             _Letter_I = new Gesture("I", iddle);
298             //Calling Input Simulator For Virtual Keys
299             InputSimulator sim = new InputSimulator();
300             _Letter_I.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_I);
301
302             //Register the gesture
303             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
304             await _gesturesService.RegisterGesture(_Letter_I, isGlobal: true);
305         }
```

3.4.10. Letra J

Se usa la variable Iddle para definir el gesto a detectar.

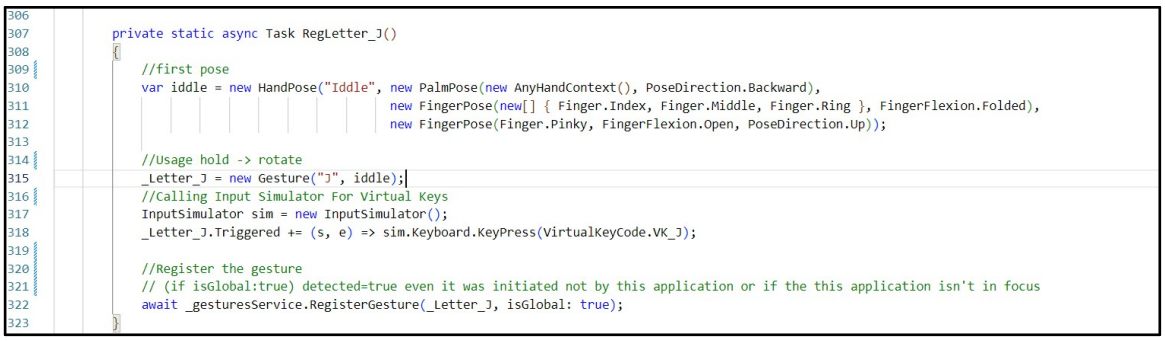
Iddle:
La palma de la mano está orientada hacia atrás.
Los dedos índice, medio y anular están doblados.
El dedo meñique está estirado hacia arriba..

Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_J= new Gesture("J", Iddle);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "J" (VirtualKeyCode.VK_J).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_J, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 40 – Letra J, Alphabet Control



```
306
307         private static async Task RegLetter_J()
308         {
309             //first pose
310             var iddle = new HandPose("Iddle", new PalmPose(new AnyHandContext(), PoseDirection.Backward),
311                                               new FingerPose(new[] { Finger.Index, Finger.Middle, Finger.Ring }, FingerFlexion.Folded),
312                                               new FingerPose(Finger.Pinky, FingerFlexion.Open, PoseDirection.Up));
313
314             //Usage hold -> rotate
315             _Letter_J = new Gesture("J", iddle);
316             //Calling Input Simulator For Virtual Keys
317             InputSimulator sim = new InputSimulator();
318             _Letter_J.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_J);
319
320             //Register the gesture
321             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
322             await _gesturesService.RegisterGesture(_Letter_J, isGlobal: true);
323         }
```

3.4.11. Letra K

Se usa la variable Fist para definir el gesto a detectar.

Fist:
La palma de la mano está orientada hacia atrás.
El dedo índice está estirado hacia arriba.
El pulgar está estirado hacia la derecha.
Los dedos medio, anular y meñique están doblados.

Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_K= new Gesture("K", Fist);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "K" (VirtualKeyCode.VK_K).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_K, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 41 – Letra K, Alphabet Control

```
324
325         private static async Task RegLetter_K()
326         {
327             //first pose
328             var fist = new HandPose("Fist", new PalmPose(new AnyHandContext(), PoseDirection.Backward),
329                                             new FingerPose(Finger.Index, FingerFlexion.Open, PoseDirection.Up),
330                                             new FingerPose(Finger.Thumb, FingerFlexion.Open, PoseDirection.Right),
331                                             new FingerPose(new[] { Finger.Middle, Finger.Ring, Finger.Pinky, }, FingerFlexion.Folded));
332
333             //Usage hold -> rotate
334             _Letter_K = new Gesture("K", fist);
335             //Calling Input Simulator For Virtual Keys
336             InputSimulator sim = new InputSimulator();
337             _Letter_K.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_K);
338
339             //Register the gesture
340             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
341             await _gesturesService.RegisterGesture(_Letter_K, isGlobal: true);
342         }
```

3.4.12. Letra L

Se usa la variable Open para definir el gesto a detectar.

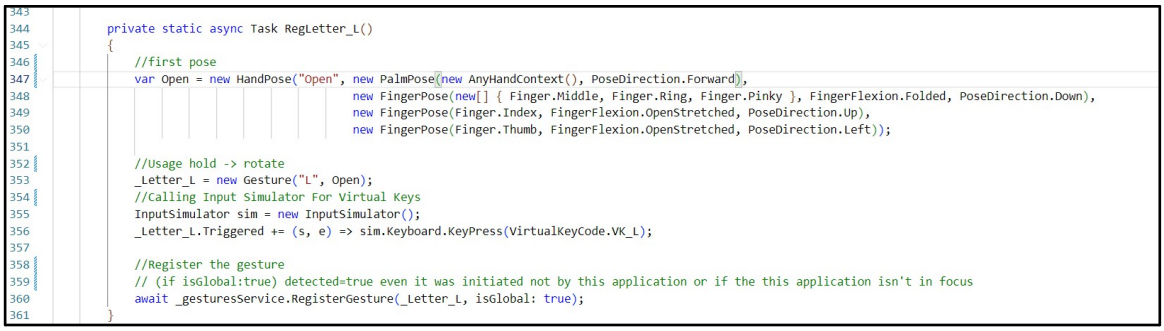
Open:
La palma de la mano está orientada hacia adelante.
El dedo índice está estirado hacia arriba.
El pulgar está estirado hacia la Izquierda.
Los dedos medio, anular y meñique están doblados.

Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_L= new Gesture("L", Open);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "L" (VirtualKeyCode.VK_L).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_L, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 42 – Letra L, Alphabet Control



```
343
344         private static async Task RegLetter_L()
345         {
346             //first pose
347             var Open = new HandPose("Open", new PalmPose(new AnyHandContext(), PoseDirection.Forward),
348                                             new FingerPose(new[] { Finger.Middle, Finger.Ring, Finger.Pinky }, FingerFlexion.Folded, PoseDirection.Down),
349                                             new FingerPose(Finger.Index, FingerFlexion.OpenStretched, PoseDirection.Up),
350                                             new FingerPose(Finger.Thumb, FingerFlexion.OpenStretched, PoseDirection.Left));
351
352             //Usage hold -> rotate
353             _Letter_L = new Gesture("L", Open);
354             //Calling Input Simulator For Virtual Keys
355             InputSimulator sim = new InputSimulator();
356             _Letter_L.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_L);
357
358             //Register the gesture
359             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
360             await _gesturesService.RegisterGesture(_Letter_L, isGlobal: true);
361         }
```

3.4.13. Letra M

Se usa la variable Fist para definir el gesto a detectar.

Fist:
La palma de la mano está orientada hacia atrás.
Los dedos índice, medio y anular están estirados hacia abajo.
Los dedos pulgar y meñique estan doblados.
La yema del dedo índice toca al dedo medio.
La yema del dedo medio toca al dedo anular.
La yema del dedo índice no toca al pulgar.
La yema del dedo anular no toca al meñique.

Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_M= new Gesture("M", Fist);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "M" (VirtualKeyCode.VK_M).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_M, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 43 – Letra M, Alphabet Control

```
362
363         private static async Task RegLetter_M()
364         {
365             //first pose
366             var fist = new HandPose("Fist", new PalmPose(new AnyHandContext(), PoseDirection.Backward),
367                                             new FingerPose(new[] { Finger.Index, Finger.Ring, Finger.Middle }, FingerFlexion.OpenStretched, PoseDirection.Down),
368                                             new FingertipDistanceRelation(Finger.Index, RelativeDistance.Touching, Finger.Middle),
369                                             new FingertipDistanceRelation(Finger.Middle, RelativeDistance.Touching, Finger.Ring),
370                                             new FingertipDistanceRelation(Finger.Index, RelativeDistance.NotTouching, Finger.Thumb),
371                                             new FingertipDistanceRelation(Finger.Ring, RelativeDistance.NotTouching, Finger.Pinky));
372
373             ///Usage hold -> rotate
374             _Letter_M = new Gesture("M", fist);
375             //Calling Input Simulator For Virtual Keys
376             InputSimulator sim = new InputSimulator();
377             _Letter_M.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_M);
378
379             //Register the gesture
380             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
381             await _gesturesService.RegisterGesture(_Letter_M, isGlobal: true);
382         }
```

3.4.14. Letra N

Se usa la variable Fist para definir el gesto a detectar.

Fist:
La palma de la mano está orientada hacia atrás.
Los dedos índice y medio están estirados hacia abajo.
La yema del dedo índice toca al dedo medio.
La yema del dedo índice está colocada a la izquierda de la yema del dedo medio.
Los dedos anular, meñique y pulgar están doblados.

Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_N= new Gesture("N", Fist);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "N" (VirtualKeyCode.VK_N).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_N, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 44 – Letra N, Alphabet Control

```
384
385         private static async Task RegLetter_N()
386         {
387             //first pose
388             var fist = new HandPose("Fist", new PalmPose(new AnyHandContext(), PoseDirection.Backward),
389                                             new FingerPose(new[] { Finger.Index, Finger.Middle }, FingerFlexion.Open, PoseDirection.Down),
390                                             new FingertipDistanceRelation(Finger.Index, RelativeDistance.Touching, Finger.Middle),
391                                             new FingertipPlacementRelation(Finger.Index, RelativePlacement.Left, Finger.Middle),
392                                             new FingerPose(new[] { Finger.Ring, Finger.Pinky, Finger.Thumb }, FingerFlexion.Folded));
393
394             //Usage hold -> rotate
395             _Letter_N = new Gesture("N", fist);
396             //Calling Input Simulator For Virtual Keys
397             InputSimulator sim = new InputSimulator();
398             _Letter_N.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_N);
399
400             //Register the gesture
401             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
402             await _gesturesService.RegisterGesture(_Letter_N, isGlobal: true);
403         }
```

3.4.15. Letra O

Se usa la variable RockOn para definir el gesto a detectar.

RockOn:
La orientación de la palma de la mano no está definida.
Los dedos medio, anular, meñique, pulgar e índice están estirados.
La yema del pulgar toca las yemas del índice y medio.

Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_O= new Gesture("O", RockOn);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "O" (VirtualKeyCode.VK_O).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_O, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 45 – Letra O, Alphabet Control

```
105         private static async Task RegLetter_O()
106         {
107
108             //first pose
109             var RockOn = new HandPose("Rock", new PalmPose(new AnyHandContext(), PoseDirection.Undefined),
110                                               new FingerPose(new[] { Finger.Middle, Finger.Ring, Finger.Pinky, Finger.Thumb, Finger.Index }, FingerFlexion.Open),
111                                               new FingertipDistanceRelation(Finger.Thumb, RelativeDistance.Touching, new[] { Finger.Index, Finger.Middle }));
112
113             //Usage hold -> rotate
114             _Letter_O = new Gesture("O", RockOn);
115             //Calling Input Simulator For Virtual Keys
116             InputSimulator sim = new InputSimulator();
117             _Letter_O.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_O);
118
119             //Register the gesture
120             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
121             await _gesturesService.RegisterGesture(_Letter_O, isGlobal: true);
122         }
```

3.4.16. Letra P

Se usa la variable Iddle para definir el gesto a detectar.

Iddle:
El pulgar está doblado.
La yema del dedo índice toca la yema del pulgar.
La yema del dedo índice está ubicada debajo de la yema del pulgar.
La yema del dedo medio toca la yema del anular.
Los dedos medio, anular y meñique están estirados hacia abajo.

Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_P= new Gesture("P", Iddle);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "P" (VirtualKeyCode.VK_P).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_P, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 46 – Letra P, Alphabet Control

```
423
424         private static async Task RegLetter_P()
425         {
426             //first pose
427             var iddle = new HandPose("Iddle", new FingerPose(new[] { Finger.Thumb }, FingerFlexion.Folded, PoseDirection.Undefined),
428                                               new FingertipDistanceRelation(Finger.Index, RelativeDistance.Touching, Finger.Thumb),
429                                               new FingertipPlacementRelation(Finger.Index, RelativePlacement.Below, Finger.Thumb),
430                                               new FingertipDistanceRelation(Finger.Middle, RelativeDistance.Touching, Finger.Ring),
431                                               new FingerPose(new[] { Finger.Middle, Finger.Ring, Finger.Pinky }, FingerFlexion.Open, PoseDirection.Down));
432
433             //Usage hold -> rotate
434             _Letter_P = new Gesture("P", iddle);
435             _Letter_P.Triggered += (s, e) => OnGestureDetected(s, e, ConsoleColor.Yellow);
436             //Calling Input Simulator For Virtual Keys
437             InputSimulator sim = new InputSimulator();
438             _Letter_P.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_P);
439
440             //Register the gesture
441             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
442             await _gesturesService.RegisterGesture(_Letter_P, isGlobal: true);
443         }
```

3.4.17. Letra Q

Se usa la variable Hold para definir el gesto a detectar.

Hold:

La palma de la mano está orientada hacia abajo.
Los dedos pulgar e índice están estirados hacia abajo.
La yema del pulgar está ubicada a la izquierda de la yema del índice.
La yema del pulgar no toca la yema del índice.
Los dedos medio, anular y meñique están doblados.

Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_Q= new Gesture("Q", Hold);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "Q" (VirtualKeyCode.VK_Q).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_Q, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 47 – Letra Q, Alphabet Control

```
144
145         private static async Task RegLetter_Q()
146         {
147             //first pose
148             var hold = new HandPose("Hold", new PalmPose(new AnyHandContext(), PoseDirection.Down),
149                                             new FingerPose(new[] { Finger.Thumb, Finger.Index }, FingerFlexion.Open, PoseDirection.Down),
150                                             new FingertipPlacementRelation( Finger.Thumb, RelativePlacement.left, Finger.Index),
151                                             new FingertipDistanceRelation(Finger.Thumb, RelativeDistance.NotTouching, Finger.Index),
152                                             new FingerPose(new[] { Finger.Middle, Finger.Ring, Finger.Pinky }, FingerFlexion.Folded));
153
154             //Usage hold -> rotate
155             _Letter_Q = new Gesture("Q", hold);
156             //Calling Input Simulator For Virtual Keys
157             InputSimulator sim = new InputSimulator();
158             _Letter_Q.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_Q);
159
160             //Register the gesture
161             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
162             await _gesturesService.RegisterGesture(_Letter_Q, isGlobal: true);
163         }
```

3.4.18. Letra R

Se usa la variable Fist para definir el gesto a detectar.

Fist:
La palma de la mano está orientada hacia adelante.
Los dedos índice y pulgar están estirados hacia arriba.
El dedo medio está estirado hacia arriba.
Los dedos anular y meñique están doblados hacia abajo.
La yema del dedo índice está tocando la yema del dedo medio.
La yema del pulgar está debajo de la yema del índice.

Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_R= new Gesture("R", Fist);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "R" (VirtualKeyCode.VK_R).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_R, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 48 – Letra R, Alphabet Control

```
464
465         private static async Task RegLetter_R()
466         {
467             //first pose
468             var fist = new HandPose("Fist", new PalmPose(new AnyHandContext(), PoseDirection.Forward),
469                                             new FingerPose(new[] { Finger.Index, Finger.Thumb }, FingerFlexion.Open, PoseDirection.Up),
470                                             new FingerPose(Finger.Middle, FingerFlexion.Open, PoseDirection.Up),
471                                             new FingerPose(new[] { Finger.Pinky, Finger.Ring }, FingerFlexion.Folded, PoseDirection.Down),
472                                             new FingertipDistanceRelation(Finger.Index, RelativeDistance.Touching, Finger.Middle),
473                                             new FingertipPlacementRelation(Finger.Thumb, RelativePlacement.Below, Finger.Index));
474
475             //Usage hold -> rotate
476             _Letter_R = new Gesture("R", fist);
477             //Calling Input Simulator For Virtual Keys
478             InputSimulator sim = new InputSimulator();
479             _Letter_R.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_R);
480
481             //Register the gesture
482             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
483             await _gesturesService.RegisterGesture(_Letter_R, isGlobal: true);
484         }
485
```

3.4.19. Letra S

Se usa la variable Iddle para definir el gesto a detectar.

Iddle:
Los dedos índice, medio, anular y meñique están doblados hacia adentro.
La palma de la mano está orientada hacia adelante.
El dedo pulgar está estirado hacia adelante.
La yema del pulgar está debajo de la yema del índice.

Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_S= new Gesture("S", Iddle);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "S" (VirtualKeyCode.VK_S).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_S, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 49 – Letra S, Alphabet Control

```
485
486         private static async Task RegLetter_S()
487         {
488             //first pose
489             var iddle = new HandPose("Iddle", new FingerPose(new[] { Finger.Index, Finger.Middle, Finger.Ring, Finger.Pinky }, FingerFlexion.Folded),
490                                               new PalmPose(new AnyHandContext(), PoseDirection.Forward),
491                                               new FingerPose(Finger.Thumb, FingerFlexion.Open, PoseDirection.Forward));
492
493             //Usage hold -> rotate
494             _Letter_S = new Gesture("S", iddle);
495             //Calling Input Simulator For Virtual Keys
496             InputSimulator sim = new InputSimulator();
497             _Letter_S.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_S);
498
499             //Register the gesture
500             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
501             await _gesturesService.RegisterGesture(_Letter_S, isGlobal: true);
502         }
503
```

3.4.20. Letra T

Se usa la variable Fist para definir el gesto a detectar.

Fist:
La palma de la mano está orientada hacia la izquierda.
El meñique está extendido hacia arriba.
Los dedos índice, medio, anular y pulgar están doblados hacia adentro.

Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_T= new Gesture("T", Fist);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "T" (VirtualKeyCode.VK_T).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_T, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 50 – Letra T, Alphabet Control

```
503
504         private static async Task RegLetter_T()
505         {
506             //first pose
507             var fist = new HandPose("Fist", new PalmPose(new AnyHandContext(), PoseDirection.Left),
508                                             new FingerPose(Finger.Pinky, FingerFlexion.Open, PoseDirection.Up),
509                                             new FingerPose(new[] { Finger.Index, Finger.Middle, Finger.Ring, Finger.Thumb }, FingerFlexion.Folded));
510
511
512             //Usage hold -> rotate
513             _Letter_T = new Gesture("T", fist);
514             _Letter_T.Triggered += (s, e) => OnGestureDetected(s, e, ConsoleColor.Red);
515             //Calling Input Simulator For Virtual Keys
516             InputSimulator sim = new InputSimulator();
517             _Letter_T.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_T);
518
519
520             //Register the gesture
521             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
522             await _gesturesService.RegisterGesture(_Letter_T, isGlobal: true);
523         }
```

3.4.21. Letra U

Se usa la variable Fist para definir el gesto a detectar.

Fist:

Los dedos índice y medio están extendidos hacia arriba.
Los dedos anular, meñique y pulgar están doblados hacia adentro.
El pulgar está orientado hacia la derecha.
El índice está tocando al dedo medio.
El índice se encuentra a la izquierda del dedo medio.

Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_U= new Gesture("U", Fist);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "U" (VirtualKeyCode.VK_U).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_U, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 51 – Letra U, Alphabet Control

```
522         private static async Task RegLetter_U()
523         {
524             //first pose
525             var fist = new HandPose("Fist", new FingerPose(new[] { Finger.Index, Finger.Middle }, FingerFlexion.Open, PoseDirection.Up),
526                                             new FingerPose(new[] { Finger.Ring, Finger.Pinky, Finger.Thumb }, FingerFlexion.Folded),
527                                             new FingerPose(Finger.Thumb, PoseDirection.Right),
528                                             new FingertipDistanceRelation(Finger.Index, RelativeDistance.Touching, Finger.Middle),
529                                             new FingertipPlacementRelation(Finger.Index, RelativePlacement.Left, Finger.Middle));
530
531
532             //Usage hold -> rotate
533             _Letter_U = new Gesture("U", fist);
534             //Calling Input Simulator For Virtual Keys
535             InputSimulator sim = new InputSimulator();
536             _Letter_U.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_U);
537
538             //Register the gesture
539             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
540             await _gesturesService.RegisterGesture(_Letter_U, isGlobal: true);
541         }
```

3.4.22. Letra V

Se usa la variable Fist para definir el gesto a detectar.

Fist:
Los dedos índice y medio están extendidos hacia arriba.
Los dedos anular, meñique y pulgar están doblados hacia adentro.
El pulgar está orientado hacia la derecha.
El índice está tocando al dedo medio.
El índice se encuentra a la izquierda del dedo medio.

Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_V= new Gesture("V", Fist);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "V" (VirtualKeyCode.VK_V).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_V, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 52 – Letra V, Alphabet Control

```
542
543         private static async Task RegLetter_V()
544         {
545             //first pose
546             var fist = new HandPose("Fist", new FingerPose(new[] { Finger.Index, Finger.Middle }, FingerFlexion.OpenStretched, PoseDirection.Up),
547                                             new FingerPose(new[] { Finger.Ring, Finger.Pinky, Finger.Thumb }, FingerFlexion.Folded),
548                                             new FingertipDistanceRelation(Finger.Index, RelativeDistance.NotTouching, Finger.Middle),
549                                             new FingertipPlacementRelation(Finger.Thumb, RelativePlacement.InFront, Finger.Ring));
550
551             //Usage hold -> rotate
552             _Letter_V = new Gesture("V", fist);
553             //Calling Input Simulator For Virtual Keys
554             InputSimulator sim = new InputSimulator();
555             _Letter_V.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_V);
556
557             //Register the gesture
558             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
559             await _gesturesService.RegisterGesture(_Letter_V, isGlobal: true);
560         }
```

3.4.23. Letra W

Se usa la variable Fist para definir el gesto a detectar.

Fist:

Los dedos índice, medio y anular están extendidos hacia arriba.
Los dedos meñique y pulgar están doblados hacia adentro.
El índice no está tocando al dedo medio.
El medio no está tocando al anular.
El pulgar está tocando al meñique.
El pulgar está colocado adelante del meñique.

Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_W= new Gesture("W", Fist);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "W" (VirtualKeyCode.VK_W).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_W, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 53 – Letra W, Alphabet Control

```
562     private static async Task RegLetter_W()
563     {
564         //first pose
565         var fist = new HandPose("Fist", new FingerPose(new[] { Finger.Index, Finger.Middle, Finger.Ring }, FingerFlexion.Open, PoseDirection.Up),
566                                         new FingerPose(new[] { Finger.Pinky, Finger.Thumb }, FingerFlexion.Folded),
567                                         new FingertipDistanceRelation(Finger.Index, RelativeDistance.NotTouching, Finger.Middle),
568                                         new FingertipDistanceRelation(Finger.Middle, RelativeDistance.NotTouching, Finger.Ring),
569                                         new FingertipPlacementRelation(Finger.Thumb, RelativePlacement.InFront, Finger.Pinky));
570
571         //Usage hold -> rotate
572         _Letter_W = new Gesture("W", fist);
573         //Calling Input Simulator For Virtual Keys
574         InputSimulator sim = new InputSimulator();
575         _Letter_W.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_W);
576
577         //Register the gesture
578         // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
579         await _gesturesService.RegisterGesture(_Letter_W, isGlobal: true);
580     }
```

3.4.24. Letra X

Se usa la variable Fist para definir el gesto a detectar.

Fist:

La palma de la mano esta orientada hacia adelante.
Los dedos meñique, medio y anular están doblados.
El índice está extendido hacia adelante.
El pulgar está tocando al dedo medio.

Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_X= new Gesture("X", Fist);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "X" (VirtualKeyCode.VK_X).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_X, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 54 – Letra X, Alphabet Control

```
582
583         private static async Task RegLetter_X()
584         {
585             //first pose
586             var fist = new HandPose("Fist", new PalmPose(new AnyHandContext(), PoseDirection.Forward),
587                                             new FingerPose(new[] { Finger.Pinky, Finger.Middle, Finger.Ring }, FingerFlexion.Folded),
588                                             new FingerPose(Finger.Index, FingerFlexion.Open, PoseDirection.Forward),
589                                             new FingertipDistanceRelation(Finger.Thumb, RelativeDistance.Touching, Finger.Middle)
590                                             );
591
592             //Usage hold -> rotate
593             _Letter_X = new Gesture("X", fist);
594             //Calling Input Simulator For Virtual Keys
595             InputSimulator sim = new InputSimulator();
596             _Letter_X.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_X);
597
598             //Register the gesture
599             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
600             await _gesturesService.RegisterGesture(_Letter_X, isGlobal: true);
601         }
```

3.4.25. Letra Y

Se usa la variable Fist para definir el gesto a detectar.

Fist:
Los dedos medio, anular e índice están doblados.
El pulgar está extendido hacia la izquierda.
El meñique está extendido hacia la derecha.

Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_X= new Gesture("Y", Fist);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "Y" (VirtualKeyCode.VK_Y).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_Y, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 55 – Letra Y, Alphabet Control

```
602
603         private static async Task RegLetter_Y()
604         {
605             //first pose
606             var fist = new HandPose("Fist", new FingerPose(new[] { Finger.Middle, Finger.Ring, Finger.Index }, FingerFlexion.Folded),
607                                             new FingerPose(Finger.Thumb, FingerFlexion.Open, PoseDirection.Left),
608                                             new FingerPose(Finger.Pinky, FingerFlexion.Open, PoseDirection.Right));
609
610             //Usage hold -> rotate
611             _Letter_Y = new Gesture("Y", fist);
612
613             //Calling Input Simulator For Virtual Keys
614             InputSimulator sim = new InputSimulator();
615             _Letter_Y.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_Y);
616
617             //Register the gesture
618             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
619             await _gesturesService.RegisterGesture(_Letter_Y, isGlobal: true);
620         }
```

3.4.26. Letra Z

Se usa la variable Fist para definir el gesto a detectar.

Fox:
La palma de la mano está orientada hacia atrás.
Los dedos medio y meñique están extendidos hacia la izquierda.
Los dedos anular, pulgar e índice están doblados.

Se define el gesto utilizando la posicion de mano previamente definida:
_Letter_Z= new Gesture("Z", Fox);

Se inicializa la libreria Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsacion en la tecla virtual "Z" (VirtualKeyCode.VK_Z).

Se registra el uso del gesto y se declara si se desea que este funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_Letter_Y, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 56 – Letra Z, Alphabet Control

```
621
622         private static async Task RegLetter_Z()
623         {
624             //first pose
625             var Fox = new HandPose("zeta", new PalmPose(new AnyHandContext(), PoseDirection.Backward),
626                                            new FingerPose(new[] { Finger.Middle, Finger.Pinky }, FingerFlexion.Open, PoseDirection.Left),
627                                            new FingerPose(new[] { Finger.Ring, Finger.Thumb, Finger.Index }, FingerFlexion.Folded));
628
629             //Usage hold -> rotate
630             _Letter_Z = new Gesture("Z", Fox);
631             //Calling Input Simulator For Virtual Keys
632             InputSimulator sim = new InputSimulator();
633             _Letter_Z.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_Z);
634
635             //Register the gesture
636             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
637             await _gesturesService.RegisterGesture(_Letter_Z, isGlobal: true);
638         }
639
```

3.4.27. Interfaz de Usuario

En este código se define un manejador de eventos que se activa cuando el cursor del mouse entra en el área del botón. El método BT_A_MouseEnter se ejecutará cuando ocurra este evento.

this.PictureBox.Image = new Bitmap(RutaArch + "Letra_A.gif");: Asigna una imagen al control PictureBox. La imagen se carga desde un archivo llamado "Letra_A.gif" que se encuentra en la ruta especificada por la variable RutaArch.

PictureBox.Refresh();: Refresca el control PictureBox para que se actualice y muestre la nueva imagen asignada.

this.LabelLetra.Location = new Point(81, 97);: Establece la posición del control LabelLetra en la ventana en las coordenadas (81, 97).

LabelLetra.Text = ("A");: Establece el texto del control LabelLetra como "A".

LabelLetra.ForeColor = Color.Black;: Establece el color del texto del control LabelLetra como negro.

this.LB1.Location = new Point(478, 79);: Establece la posición del control LB1 en la ventana en las coordenadas (478, 79).

LB1.Text = ("Recomendaciones: \r\n La punta del dedo Pulgar debe \r\n Ir un poco mas arriba que\r\n Los demas dedos.");: Establece el texto del control LB1 con una serie de recomendaciones.

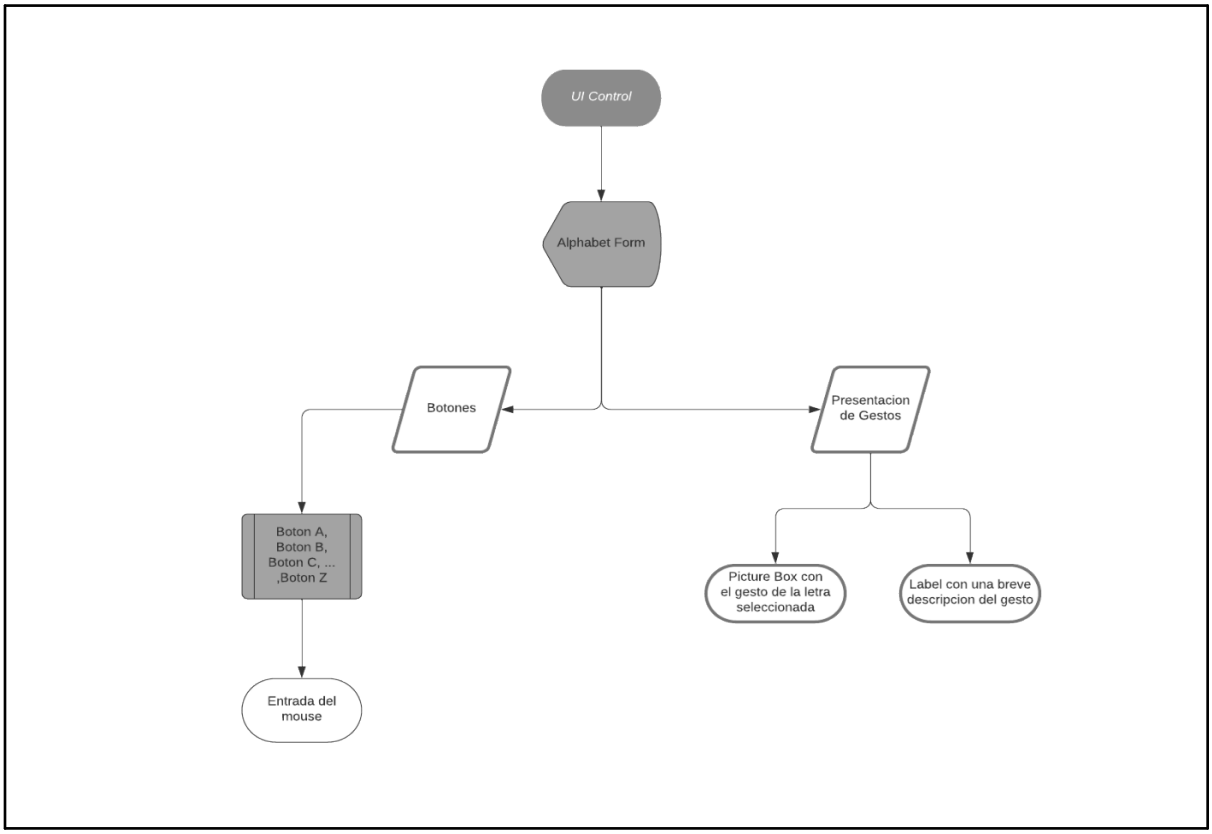
LB1.ForeColor = Color.Black;: Establece el color del texto del control LB1 como negro.

Los demás eventos son similares, variando únicamente en la letra mostrada, el gesto y la descripción del gesto (para los que la tienen). Cada evento BT_B_MouseEnter, BT_C_MouseEnter, etc., sigue la misma estructura, con la letra, la imagen y la descripción correspondientes.

```
643     private void BT_A_MouseEnter(object sender, EventArgs e)//Boton A
644     {
645         this.PictureBox.Image = new Bitmap(RutaArch + "Letra_A.gif");
646         PictureBox.Refresh();
647         this.LabelLetra.Location = new Point(81, 97);
648         LabelLetra.Text = ("A");
649         LabelLetra.ForeColor = Color.Black;
650         this.LB1.Location = new Point(478, 79);
651         LB1.Text = ("Recomendaciones: \r\n La punta del dedo Pulgar debe \r\n Ir un poco mas arriba que\r\n Los demas dedos.");
652         LB1.ForeColor = Color.Black;
653     }
654     private void BT_B_MouseEnter(object sender, EventArgs e)//Boton B
655     {
656         LB1.ForeColor = Color.FromArgb(255, 72, 177, 37);
657         this.PictureBox.Image = new Bitmap(RutaArch + "Letra_B.gif");
658         PictureBox.Refresh();
659         this.LabelLetra.Location = new Point(81, 97);
660         LabelLetra.Text = ("B");
661         LabelLetra.ForeColor = Color.Black;
662     }
663     private void BT_C_MouseEnter(object sender, EventArgs e)//Boton C
664     {
665         LB1.ForeColor = Color.FromArgb(255, 72, 177, 37);
666         this.PictureBox.Image = new Bitmap(RutaArch + "Letra_C.gif");
667         PictureBox.Refresh();
668         this.LabelLetra.Location = new Point(81, 97);
669         LabelLetra.Text = ("C");
670         LabelLetra.ForeColor = Color.Black;
671     }
672     private void BT_D_MouseEnter(object sender, EventArgs e)//Boton D
```

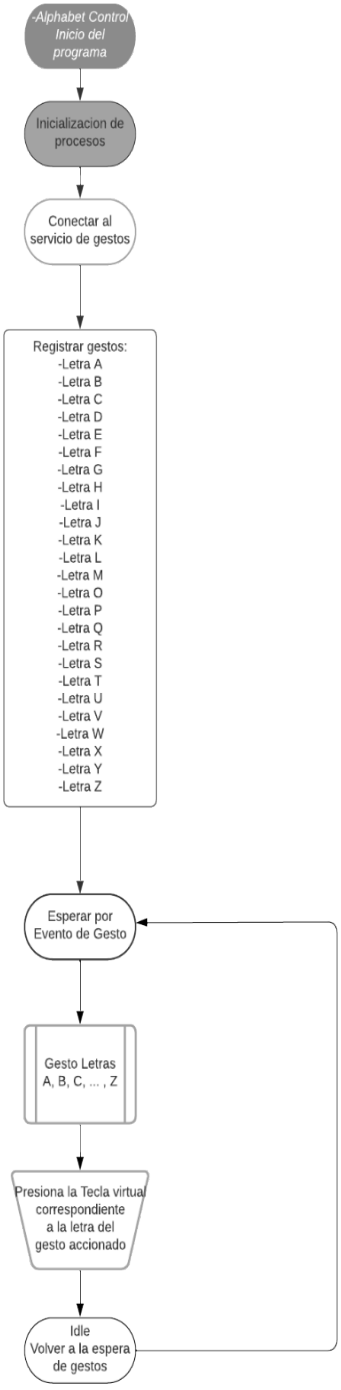
Figura 58 – Interfaz del modulo, Alphabet Control

Figura 59 – Diagrama de Funcion, Alphabet Control



58

Figura 60 – Diagrama de la ventana, Alphabet Control



3.5. Domótica Control

Se importan las bibliotecas del sistema y otras bibliotecas específicas para la gestión de aplicaciones gráficas, el reconocimiento de gestos y la utilización de un teclado virtual.

- using System;: Esta directiva permite el uso de tipos y miembros en el espacio de nombres System, que es el espacio de nombres fundamental que contiene tipos y miembros fundamentales de .NET Framework.
- using WindowsInput;: Esta directiva permite el uso de clases y tipos relacionados con la simulación de entrada de teclado y mouse en Windows.
- using WindowsInput.Native;: Esta directiva permite el uso de tipos relacionados con las teclas y los eventos de teclado a nivel de sistema.
- using System.Threading.Tasks;: Esta directiva permite el uso de tipos y miembros relacionados con la programación asincrónica en .NET, especialmente para trabajar con tareas y operaciones asincrónicas.
- using System.Windows.Forms;: Esta directiva permite el uso de clases y tipos relacionados con la creación de aplicaciones de Windows Forms en .NET Framework.
- using Microsoft.Gestures;: Esta directiva permite el uso de clases y tipos relacionados con el reconocimiento de gestos mediante la librería Microsoft Gestures.
- using Microsoft.Gestures.Endpoint;: Esta directiva permite el uso de clases y tipos relacionados con la comunicación con el servicio de reconocimiento de gestos de Microsoft.
- using System.Drawing;: Esta directiva permite el uso de tipos y miembros relacionados con la creación y manipulación de gráficos y elementos visuales en .NET Framework.

Figura 61. Librerías necesarias, Lightbulb Control

```
1   using System;
2   using WindowsInput;
3   using WindowsInput.Native;
4   using System.Threading.Tasks;
5   using System.Windows.Forms;
6   using Microsoft.Gestures;
7   using Microsoft.Gestures.Endpoint;
8   using System.Drawing;
9
```

Estas líneas de código establecen el uso de un servicio encargado de la gestión de gestos, así como la definición de los propios gestos a ser utilizados en la aplicación y variables encargadas de la gestion de los estados del bombillo .

1. _gesturesService: Representa el servicio encargado de detectar y procesar gestos dentro de la aplicación.
2. Color: Almacena el color actual del dispositivo controlado por la aplicación. Los valores posibles son: 0 (blanco), 1 (rojo), 2 (verde), 3 (azul) y 5 (morado).

3. Brightness: Almacena la intensidad actual del brillo del dispositivo controlado por la aplicación, en un rango de valores de 0 a 100.
4. Toggle: Indica el estado de encendido o apagado del dispositivo controlado por la aplicación. El valor 1 representa encendido, mientras que el valor 0 representa apagado.
5. FirstTime: Una bandera que indica si es la primera vez que se ejecuta la aplicación o si se está ejecutando por primera vez después de una reinicialización.
6. rutaArchivoBat: Almacena la ruta completa de un archivo .bat, posiblemente utilizado para realizar operaciones del sistema o configuraciones adicionales.
7. ColorChange: Variable utilizada para rastrear si se han realizado cambios en el color desde la última interacción.
8. BrightnessChange: Variable utilizada para rastrear si se han realizado cambios en la luminosidad desde la última interacción.

Figura 62. Servicios encargados de la gestión de gestos, Lightbulb Control

```
11  namespace UIControl
12  {
13      public partial class ControlBForm : Form
14      {
15          private static GesturesServiceEndpoint _gesturesService;
16          private static Gesture _rotateHSGestureR;//Subir Brillo
17          private static Gesture _rotateHSGestureL;//Bajar Brillo
18          private static Gesture _LikeGesture;// Togle On/Off
19          private static Gesture _DisLikeGesture;//Guardar Cambios
20          private static Gesture _DoceGesture;//Rotar colores en este orden: Blanco/Rojo/Verde/Azul/Morado
21          private static Gesture _CrownGesture;//Volver al color blanco
22
23          //Variables
24          private static int Color = 0; //Color = 0(White), Color = 1(Red), Color = 2(Green), Color = 3(Blue), Color = 5(Purple)
25          private static int Brightness = 100; //Define la intensidad del brillo 0-100
26          private static int Toggle = 1; //Define estado del bombillo 1 = ON, 0=OFF
27          private static int FirstTime = 0;
28          private static string rutaArchivoBat = @""; //Se reemplaza con la ruta completa del archivo .bat
29          private static string ColorChange = ""; //Variable para revisar si se hizo un cambio en el color
30          private static string BrightnessChange = "";//Variable para revisar si se hizo un cambio en la luminosidad
31
```

Estas líneas de código inicializan las variables predeterminadas en el constructor y establecen la conexión con el servicio encargado de manejar los gestos, el cual está alojado localmente en el localhost

Se pone en estado de espera al servicio encargado de la gestión de gestos.

Figura 63. En espera del servicio de gestión de gestos, Lightbulb Control

```
32          public ControlBForm()
33          {
34              InitializeComponent();
35              InitializeGestures();
36          }
37
38          private async void InitializeGestures()
39          {
40              // One can optionally pass the hostname/IP address where the gestures service is hosted
41              var gesturesServiceHostName = "localhost";
42              await RegisterGestures(gesturesServiceHostName);
43          }
44
```

Estas líneas de código establecen la conexión al servicio encargado de manejar los gestos y comienzan la espera de que los gestos sean activados o "triggered"

Figura 64. Servicio encargado de la gestión de gestos, Lightbulb Control

```
43
44          private static async Task RegisterGestures(string gesturesServiceHostName)
45          {
46              //Connect to service
47              _gesturesService = GesturesServiceEndpointFactory.Create(gesturesServiceHostName);
48              await _gesturesService.ConnectAsync();
49
50              //custom Gestures
51              await RegisterHandShakeRightGesture(); // Subir Brillo
52              await RegisterHandShakeLeftGesture(); // Bajar Brillo
53              await RegisterLikeGesture(); // Togle On/off
54              await RegisterDisLikeGesture(); //Guardar cambios
55              await RegisterDoceGesture(); //Rotar colores en este orden Blanco/Rojo/Verde/Azul/Morado
56              await RegisterCrownGesture(); //Volver al color blanco(Se sigue respetando el orden de colores desde el principio)
57          }
```

3.5.1. Subir Brillo

En este fragmento de código se definen las variables 'front' y 'right' que representan la primera y segunda pose a tener en cuenta para la detección del gesto.

Front:
Todos los dedos están estirados hacia adelante, apuntando hacia la cámara.
La palma de la mano está orientada hacia la izquierda.

Right:
Todos los dedos exepto el pulgar están estirados hacia adelante, apuntando a la cámara.
El dedo pulgar está apuntando hacia arriba.

Se define el gesto utilizando las posiciones de mano previamente definidas y se especifica el orden a seguir para la detección:
_rotateHSGestureR = new Gesture("HandshakeRight", front, right);

Cuando se detecta este gesto, llamará a la función Executor con la información del evento y un color de consola amarillo.

Se inicializa la librería Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsación de la tecla de número 8 en el teclado virtual.

Se registra el uso del gesto y se declara que este gesto funcione incluso si la ventana que almacena estos gestos no está abierta:
await _gesturesService.RegisterGesture(_rotateHSGestureR, isGlobal: true);
Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 65. Subir brillo, Lightbulb Control

```
58
59          private static async Task RegisterHandShakeRightGesture()
60          {
61              //first pose
62              var front = new HandPose("Front", new FingerPose(new AllFingersContext(), FingerFlexion.Open, PoseDirection.Forward),
63                                                new PalmPose(new AnyHandContext(), PoseDirection.Left));
64
65              //second pose
66              var right = new HandPose("Right", new FingerPose(new AllFingersContext(), FingerFlexion.Open),
67                                                new FingerPose(new[] { Finger.Index, Finger.Middle, Finger.Ring, Finger.Pinky }, PoseDirection.Right),
68                                                new FingerPose(Finger.Thumb, PoseDirection.Up),
69                                                new PalmPose(new AnyHandContext(), PoseDirection.Forward));
70
71              //Usage front -> right
72              _rotateHSGestureR = new Gesture("HandshakeRight", front, right);
73              _rotateHSGestureR.Triggered += (s, e) => Executor(s, e, ConsoleColor.Yellow);
74
75              InputSimulator sim = new InputSimulator();
76              _rotateHSGestureR.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_8);
77
78
79              //Register the gesture
80              // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
81              await _gesturesService.RegisterGesture(_rotateHSGestureR, isGlobal: true);
82          }
```

3.5.2. Bajar Brillo

En este fragmento de código se definen las variables 'front' y 'left' que representan la primera y segunda pose a tener en cuenta para la detección del gesto.

Front:

El dedo pulgar e índice están estirados hacia adelante, apuntando hacia la cámara.
La palma de la mano está orientada hacia la izquierda.

Left:
Todos los dedos estan estirados.
Todos los dedos exepto el pulgar estan estirados hacia la izquierda.
El dedo pulgar está apuntando hacia arriba.
La palma de la mano está orientada hacia atrás.

Se define el gesto utilizando las posiciones de mano previamente definidas y se especifica el orden a seguir para la detección:

_rotateHSGestureL = new Gesture("HandshakeLeft", front, left);

Cuando se detecta este gesto, llamará a la función Executor con la información del evento y un color de consola amarillo oscuro.

Se inicializa la librería Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsación de la tecla de número 8 en el teclado virtual.

Se registra el uso del gesto y se declara que este gesto funcione incluso si la ventana que almacena estos gestos no está abierta:

await _gesturesService.RegisterGesture(_rotateHSGestureL, isGlobal: true);

Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 66. Bajar brillo, Lightbulb Control

```
84
85          private static async Task RegisterHandShakeLeftGesture()
86          {
87              //first pose
88              var front = new HandPose("Front", new FingerPose(new AllFingersContext(), FingerFlexion.Open, PoseDirection.Forward),
89                                                new PalmPose(new AnyHandContext(), PoseDirection.Left));
90              //second pose
91              var left = new HandPose("Left", new FingerPose(new AllFingersContext(), FingerFlexion.Open),
92                                              new FingerPose(new[] { Finger.Index, Finger.Middle, Finger.Ring, Finger.Pinky }, PoseDirection.Left),
93                                              new FingerPose(Finger.Thumb, PoseDirection.Up),
94                                              new PalmPose(new AnyHandContext(), PoseDirection.Backward));
95
96              //Usage front -> right
97              _rotateHSGestureL = new Gesture("HandshakeLeft", front, left);
98              _rotateHSGestureL.Triggered += (s, e) => Executor(s, e, ConsoleColor.DarkYellow);
99              InputSimulator sim = new InputSimulator();
00              _rotateHSGestureL.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_8);
01
02
03
04              //Register the gesture
05              // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
06              await _gesturesService.RegisterGesture(_rotateHSGestureL, isGlobal: true);
07          }
```

3.5.3. Rotar Colores

En este fragmento de código se definen las variables 'front' y 'left' que representan la primera y segunda pose a tener en cuenta para la detección del gesto.

Front:

Todos los dedos están estirados hacia arriba.
La palma de la mano está orientada hacia adelante.

Left:

Los dedos índice, pulgar y meñique están doblados.
La punta del dedo pulgar toca la punta del dedo meñique.
Los dedos medio y anular están estirados hacia arriba.
La palma de la mano está orientada hacia adelante.

Se define el gesto utilizando las posiciones de mano previamente definidas y se especifica el orden a seguir para la detección:

_DoceGesture = new Gesture("Doce", front, left);

Cuando se detecta este gesto, llamará a la función Executor con la información del evento y un color de consola amarillo oscuro.

Se inicializa la librería Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsación de la tecla 3 en el teclado virtual.

Se registra el uso del gesto y se declara que este gesto funcione incluso si la ventana que almacena estos gestos no está abierta:

await _gesturesService.RegisterGesture(_DoceGesture, isGlobal: true);

Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 67. Rotar Colores, Lightbulb Control

```
107
108         private static async Task RegisterDoceGesture()
109         {
110             //first pose
111             var front = new HandPose("Front", new FingerPose(new AllFingersContext(), FingerFlexion.Open, PoseDirection.Up),
112                                               new PalmPose(new AnyHandContext(), PoseDirection.Forward));
113             //second pose
114             var left = new HandPose("Left", new FingerPose(new[] { Finger.Index, Finger.Thumb, Finger.Pinky }, FingerFlexion.Folded),
115                                             new FingertipDistanceRelation(Finger.Thumb, RelativeDistance.Touching, Finger.Pinky),
116                                             new FingerPose(new[] {Finger.Middle, Finger.Ring}, FingerFlexion.Open, PoseDirection.Up ),
117                                             new PalmPose(new AnyHandContext(), PoseDirection.Forward));
118
119             //Usage front -> right
120             _DoceGesture = new Gesture("Doce", front, left);
121             _DoceGesture.Triggered += (s, e) => Executor(s, e, ConsoleColor.DarkYellow);
122
123             InputSimulator sim = new InputSimulator();
124             _DoceGesture.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_3);
125
126             //Register the gesture
127             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
128             await _gesturesService.RegisterGesture(_DoceGesture, isGlobal: true);
129         }
```

3.5.4. Reiniciar Colores

En este fragmento de código se definen las variables 'front' y 'left' que representan la primera y segunda pose a tener en cuenta para la detección del gesto.

Front:

Todos los dedos están estirados hacia arriba.
La palma de la mano está orientada hacia adelante.

Left:

Los dedos medio, pulgar y meñique están estirados hacia arriba.
Los dedos índice y anular están estirados hacia adelante.
La palma de la mano está orientada hacia adelante.

Se define el gesto utilizando las posiciones de mano previamente definidas y se especifica el orden a seguir para la detección:
\
_CrownGesture = new Gesture("Crown", front, left);

Cuando se detecta este gesto, llamará a la función Executor con la información del evento y un color de consola amarillo oscuro.

Se inicializa la librería Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsación de la tecla 3 en el teclado virtual.

Se registra el uso del gesto y se declara que este gesto funcione incluso si la ventana que almacena estos gestos no está abierta:

await _gesturesService.RegisterGesture(_CrownGesture, isGlobal: true);

Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 68. Reiniciar Colores, Lightbulb Control

```
130
131         private static async Task RegisterCrownGesture()
132         {
133             //first pose
134             var front = new HandPose("Front", new FingerPose(new AllFingersContext(), FingerFlexion.Open, PoseDirection.Up),
135                                               new PalmPose(new AnyHandContext(), PoseDirection.Forward));
136             //second pose
137             var left = new HandPose("Left", new FingerPose(new[] { Finger.Middle, Finger.Thumb, Finger.Pinky }, FingerFlexion.Open, PoseDirection.Up),
138                                               new FingerPose(new[] { Finger.Index, Finger.Ring }, FingerFlexion.Open, PoseDirection.Forward),
139                                               new PalmPose(new AnyHandContext(), PoseDirection.Forward));
140
141             //Usage front -> right
142             _CrownGesture = new Gesture("Crown", front, left);
143             _CrownGesture.Triggered += (s, e) => Executor(s, e, ConsoleColor.DarkYellow);
144
145             InputSimulator sim = new InputSimulator();
146             _CrownGesture.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_3);
147
148
149             //Register the gesture
150             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
151             await _gesturesService.RegisterGesture(_CrownGesture, isGlobal: true);
152         }
153
```

3.5.5. Encender/Apagar

En este fragmento de código se definen las variables 'like' y 'right' que representan la primera y segunda pose a tener en cuenta para la detección del gesto.

Like:

Los dedos índice, medio, anular y meñique están doblados.
El dedo pulgar está estirado hacia arriba.
La palma de la mano está orientada hacia la izquierda.

Right:

Todos los dedos están doblados.
La palma de la mano está orientada hacia la izquierda.

Se define el gesto utilizando las posiciones de mano previamente definidas y se especifica el orden a seguir para la detección:

_LikeGesture = new Gesture("LikeG", right, like);

Cuando se detecta este gesto, llamará a la función Executor con la información del evento y un color de consola amarillo.

Se inicializa la librería Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsación de la tecla 7 en el teclado virtual.

Se registra el uso del gesto y se declara que este gesto funcione incluso si la ventana que almacena estos gestos no está abierta:

await _gesturesService.RegisterGesture(_LikeGesture, isGlobal: true);

Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 69. Encender/Apagar, Lightbulb Control

```
153
154         private static async Task RegisterLikeGesture()
155         {
156             //first pose
157             var like = new HandPose("Like", new FingerPose(new[] { Finger.Index, Finger.Middle, Finger.Ring, Finger.Pinky }, FingerFlexion.Folded),
158                                             new FingerPose(Finger.Thumb, FingerFlexion.Open, PoseDirection.Up),
159                                             new PalmPose(new AnyHandContext(), PoseDirection.Left));
160             //second pose
161             var right = new HandPose("Right", new FingerPose(new AllFingersContext(), FingerFlexion.Folded),
162                                             new PalmPose(new AnyHandContext(), PoseDirection.Left)) ;
163
164             //Usage front -> right
165             _LikeGesture = new Gesture("LikeG", right, like);
166             _LikeGesture.Triggered += (s, e) => Executor(s, e, ConsoleColor.Yellow);
167             InputSimulator sim = new InputSimulator();
168             _LikeGesture.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_7);
169
170
171             //Register the gesture
172             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
173             await _gesturesService.RegisterGesture(_LikeGesture, isGlobal: true);
174         }
```

3.5.6. Guardar Cambios

En este fragmento de código se definen las variables 'like' y 'right' que representan la primera y segunda pose a tener en cuenta para la detección del gesto.

Like:

Los dedos índice, medio, anular y meñique están doblados.
El dedo pulgar está estirado hacia abajo.
La palma de la mano está orientada hacia la derecha.

Right:

Todos los dedos están doblados.
La palma de la mano está orientada hacia la derecha.

Se define el gesto utilizando las posiciones de mano previamente definidas y se especifica el orden a seguir para la detección:

_DisLikeGesture = new Gesture("DisLikeG", right, like);

Cuando se detecta este gesto, llamará a la función Executor con la información del evento y un color de consola amarillo.

Se inicializa la librería Input Simulator para simular pulsaciones de teclado. Cuando se detecta el gesto, se simula una pulsación de la tecla 9 en el teclado virtual.

Se registra el uso del gesto y se declara que este gesto funcione incluso si la ventana que almacena estos gestos no está abierta:

await _gesturesService.RegisterGesture(_DisLikeGesture, isGlobal: true);

Esto permite que el gesto sea reconocido incluso si la ventana asociada no está activa o en primer plano.

Figura 70. Guardar Cambios, Lightbulb Control

```
175
176         private static async Task RegisterDisLikeGesture()
177         {
178             //first pose
179             var like = new HandPose("Like", new FingerPose(new[] { Finger.Index, Finger.Middle, Finger.Ring, Finger.Pinky }, FingerFlexion.Folded),
180                                             new FingerPose(Finger.Thumb, FingerFlexion.Open, PoseDirection.Down),
181                                             new PalmPose(new AnyHandContext(), PoseDirection.Right));
182             //second pose
183             var right = new HandPose("Right", new FingerPose(new AllFingersContext(), FingerFlexion.Folded),
184                                               new PalmPose(new AnyHandContext(), PoseDirection.Right));
185
186             //Usage front -> right
187             _DisLikeGesture = new Gesture("DisLikeG", right, like);
188             _DisLikeGesture.Triggered += (s, e) => Executor(s, e, ConsoleColor.Yellow);
189
190             InputSimulator sim = new InputSimulator();
191             _DisLikeGesture.Triggered += (s, e) => sim.Keyboard.KeyPress(VirtualKeyCode.VK_9);
192
193             //Register the gesture
194             // (if isGlobal:true) detected=true even it was initiated not by this application or if the this application isn't in focus
195             await _gesturesService.RegisterGesture(_DisLikeGesture, isGlobal: true);
196         }
```

3.5.7. Comandos de control del bombillo

Este método, llamado "Executor", se encarga de controlar el bombillo a través de la ejecución de archivos batch.

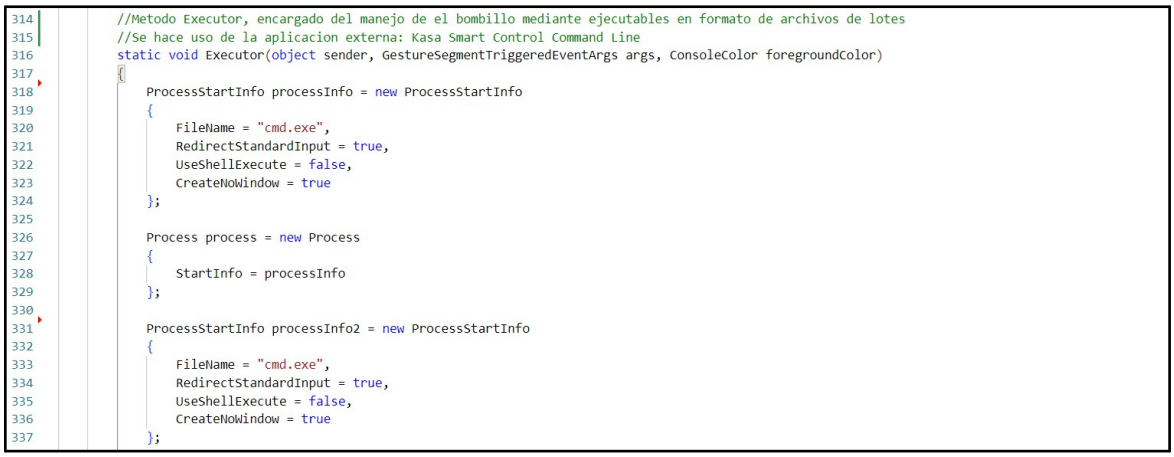
Se define un objeto ProcessStartInfo para configurar los detalles de cómo se iniciará el proceso del símbolo del sistema (cmd.exe).
Esto incluye el nombre del archivo ejecutable (cmd.exe), la redirección de la entrada estándar para permitir la escritura de comandos, y la configuración para no crear una ventana nueva (CreateNoWindow).

Se crea un nuevo objeto Process y se le asigna la información de inicio del proceso definida anteriormente.

Se repiten los pasos 1 y 2 para otro proceso, ya que hay dos comandos que se podrían ejecutar juntos.

El propósito específico de estos procesos será ejecutar comandos relacionados con el control del bombillo a través de la aplicación externa "Kasa Smart Control Command Line".

Figura 71. Executor Kasa Smart Control CMD, Lightbulb Control



```
314         //Metodo Executor, encargado del manejo de el bombillo mediante ejecutables en formato de archivos de lotes
315         //Se hace uso de la aplicacion externa: Kasa Smart Control Command Line
316         static void Executor(object sender, GestureSegmentTriggeredEventArgs args, ConsoleColor foregroundColor)
317         {
318             ProcessStartInfo processInfo = new ProcessStartInfo
319             {
320                 FileName = "cmd.exe",
321                 RedirectStandardInput = true,
322                 UseShellExecute = false,
323                 CreateNoWindow = true
324             };
325
326             Process process = new Process
327             {
328                 StartInfo = processInfo
329             };
330
331             ProcessStartInfo processInfo2 = new ProcessStartInfo
332             {
333                 FileName = "cmd.exe",
334                 RedirectStandardInput = true,
335                 UseShellExecute = false,
336                 CreateNoWindow = true
337             };
```

Este fragmento de código se activa si la variable FirstTime es igual a 0.
Aquí se ejecutan una serie de acciones específicas:

Se define la ruta del archivo batch para establecer el color blanco del bombillo en la variable rutaArchivoBat3.

Se inicia el proceso del símbolo del sistema (cmd.exe) asociado con process2.
Se escribe la ruta del archivo batch en la entrada estándar del proceso (StandardInput) para ejecutar el comando que establece el color blanco.

Se cierra el proceso process2.

Se redefine la ruta del archivo batch para establecer el brillo al 100% en rutaArchivoBat3.

Se inicia el proceso del símbolo del sistema asociado con process.

Se escribe la ruta del archivo batch en la entrada estándar del proceso para ejecutar el comando que establece el brillo al 100%.
Se cierra el proceso process.

Finalmente, se incrementa el valor de FirstTime para indicar que estas acciones se han realizado por primera vez.

Figura 72. Establecer brillo al 100%, Lightbulb Control

```
343
344             if (FirstTime == 0)
345             {
346                 string rutaArchivoBat3 = @"C:\Users\mitch\OneDrive\Escritorio\Batch\ColorWhite.cmd";
347
348
349                 //////Color Blanco///////
350                 process2.Start();
351                 process2.StandardInput.WriteLine($"\"{rutaArchivoBat3}\"");
352                 process2.Close();
353
354                 ///////Brillo al 100%/////////////////
355                 rutaArchivoBat3 = @"C:\Users\mitch\OneDrive\Escritorio\Batch\Brightness100.cmd";
356                 process.Start();
357                 process.StandardInput.WriteLine($"\"{rutaArchivoBat3}\"");
358                 process.Close();
359                 FirstTime++;
360             }
```

Este bloque de código gestiona la variable de brillo (Brightness) en respuesta a los gestos detectados.

Si el nombre del segmento de gesto es "HandshakeLeft", se disminuye el valor de Brightness en 20 unidades.
Luego se verifica si el nuevo valor de brillo es menor que cero; si es así, se establece el brillo en cero.
Se establece la variable BrightnessChange en "Yes" para indicar que ha habido un cambio en el brillo.

Por otro lado, si el nombre del segmento de gesto es "HandshakeRight", se aumenta el valor de Brightness en 20 unidades.
Luego se verifica si el nuevo valor de brillo es mayor que 100; si es así, se establece el brillo en 100;
También se establece la variable BrightnessChange en "Yes" para indicar que ha habido un cambio en el brillo.

Figura 73. Cambio al brillo, Lightbulb Control

```
361
362             ////////////////////////////////////// Manejo variable Brillo ////////////////////////////////////////////
363             if (args.GestureSegment.Name == "HandshakeLeft")
364             {
365                 Brightness -= 20;
366                 if (Brightness < 0)
367                 {
368                     Brightness = 0;
369                 }
370
371
372                 BrightnessChange = "Yes";
373             }
374
375             else if (args.GestureSegment.Name == "HandshakeRight")
376             {
377                 Brightness += 20;
378                 if (Brightness > 100)
379                 {
380                     Brightness = 100;
381                     Console.Write("Brillo al 100%");
382                 }
383
384                 BrightnessChange = "Yes";
385             }
386
```

Este fragmento de código gestiona la variable de color (Color) en respuesta a los gestos detectados.

Si el nombre del segmento de gesto es "Doce", se incrementa el valor de Color en uno. Luego se verifica si el nuevo valor de color es igual a 5; si es así, se establece el color en 0 para reiniciar el ciclo de colores.
Se establece la variable ColorChange en "Yes" para indicar que ha habido un cambio en el color.

Figura 74. Cambio al Color, Lightbulb Control

```
386
387         ////////////////////////////////////// Manejo variable Color ////////////////////////////////////////////////
388         if (args.GestureSegment.Name == "Doce")
389          {
390             Color++;
391             if (Color == 5)
392             {
393                 Color = 0;
394             }
395
396             ColorChange = "Yes";
397
398         }
```

Este fragmento de código maneja el estado de encendido/apagado en respuesta al gesto detectado.

Si el nombre del segmento de gesto es "LikeG", se ejecuta un archivo por lotes que controla el estado de encendido/apagado. Luego se realiza una serie de verificaciones para actualizar el estado de la variable Toggle:

Si Toggle es igual a 0, se incrementa en uno.
Si Toggle es igual a 1, se decrementa en uno.
Si Toggle no es ni 0 ni 1, se establece en 0 para reiniciar el estado.

Finalmente, se cierra el proceso.

Figura 75. Cambio al estado Encendido/Apagdo, Lightbulb Control

```
399
400         ////////////////////////////////////// Manejo estado Encendido/Apagado ////////////////////////////////////////////////
401         if (args.GestureSegment.Name == "LikeG")
402         {
403             string rutaArchivoBat2 = @"C:\Users\mitch\OneDrive\Escritorio\Batch\Toggle.cmd";
404             process2.Start();
405             process2.StandardInput.WriteLine($"\"{rutaArchivoBat2}\"");
406             process2.Close();
407             if (Toggle == 0)
408             {
409                 Toggle++;
410             }
411             else if (Toggle == 0)
412             {
413                 Toggle--;
414             }
415             else if (Toggle != 0 || Toggle != 1)
416             {
417                 Toggle = 0;
418             }
419         }
420
```

Este bloque de código maneja el guardado de cambios en el color y el brillo en respuesta al gesto "DisLikeG".

Para el brillo:

Si BrightnessChange es "Yes", se selecciona el archivo de lote correspondiente al nivel de brillo actual(entre 0% y 100%) y se ejecuta para guardar el cambio.
Luego se establece BrightnessChange en "No" para indicar que no se detecta un cambio adicional en el brillo.

Para el color:
Si ColorChange es "Yes", se selecciona el archivo de lote correspondiente al color actual(Blanco, Rojo, Verde, Azul y Morado) y se ejecuta para guardar el cambio.
Luego se establece ColorChange en "No" para indicar que no se detecta un cambio adicional en el color.

Figura 76. Cambio al brillo, Lightbulb Control

```
420             ////////////////////////////////////// Manejo Guardado de Cambios en Color y Brillo ////////////////////////////////////////////////
421             if (args.GestureSegment.Name == "DisLikeG")
422             {
423                 ////////////////////Brillo////////////////
424
425                 if (BrightnessChange == "Yes")
426                 {
427                     if (Brightness == 100)
428                     {
429                         //Gesto RotateLeft Bajar el Brillo a 80%
430                         rutaArchivoBat = @"C:\Users\mitch\OneDrive\Escritorio\Batch\Brightness100.cmd";
431                         Console.Write("Brillo al 100%");
432                     }
433                     else if (Brightness == 80)
434                     {
435                         //Gesto RotateLeft Bajar el Brillo a 80%
436                         rutaArchivoBat = @"C:\Users\mitch\OneDrive\Escritorio\Batch\Brightness80.cmd";
437                         Console.Write("Brillo al 80%");
438                     }
439                     else if (Brightness == 60)
440                     {
441                         //Gesto RotateLeft Bajar el Brillo a 60%
442                         rutaArchivoBat = @"C:\Users\mitch\OneDrive\Escritorio\Batch\Brightness60.cmd";
443                         Console.Write("Brillo al 60%");
444                     }
445                     else if (Brightness == 40)
446                     {
447                         //Gesto RotateLeft Para Bajar el Brillo a 40%
448                         rutaArchivoBat = @"C:\Users\mitch\OneDrive\Escritorio\Batch\Brightness40.cmd";
449                         Console.Write("Brillo al 20%");
450                     }
451                     else if (Brightness == 20)
452                     {
453                         //Gesto RotateLeft Para Bajar el Brillo a 20%
454                         rutaArchivoBat = @"C:\Users\mitch\OneDrive\Escritorio\Batch\Brightness20.cmd";
455                         Console.Write("Brillo al 20%");
456                     }
```

Este bloque de código maneja el guardado de cambios en el color y el brillo en respuesta al gesto "DisLikeG".
Para el brillo:
Si BrightnessChange es "Yes", se selecciona el archivo de lote correspondiente al nivel de brillo actual(entre 0% y 100%) y se ejecuta para guardar el cambio.
Luego se establece BrightnessChange en "No" para indicar que no se detecta un cambio adicional en el brillo.

Figura 77. Confirmando el cambio para el brillo, Lightbulb Control

```
457                     else if (Brightness == 0)
458                     {
459                         //Gesto RotateLeft Bajar el Brillo a 0%
460                         rutaArchivoBat = @"C:\Users\mitch\OneDrive\Escritorio\Batch\Brightness0.cmd";
461                         Console.Write("Brillo al 0%");
462                     }
463                     BrightnessChange = "No";
464                     process.Start();
465                     process.StandardInput.WriteLine($"\"{rutaArchivoBat}\"");
466                     process.Close();
467                 }
468
```

Para el color:
Si ColorChange es "Yes", se selecciona el archivo de lote correspondiente al color actual(1 = Blanco, 2 = Rojo, 3 = Verde, 4 = Azul y 5 = Morado) y se ejecuta para guardar el cambio.
Luego se establece ColorChange en "No" para indicar que no se detecta un cambio adicional en el color.
Y se manda a ejecutar el archivo por lotes especificado.

Figura 78. Seleccionando el comando para el color, Lightbulb Control

```
469             if(ColorChange == "Yes")
470             {
471                 if (Color == 0)
472                 {
473                     // Cambiar el color del bombillo a Blanco "W"
474                     rutaArchivoBat = @"C:\Users\mitch\OneDrive\Escritorio\Batch\ColorWhite.cmd";
475                 }
476                 if (Color == 1)
477                 {
478                     // Cambiar el color del bombillo a Rojo "R"
479                     rutaArchivoBat = @"C:\Users\mitch\OneDrive\Escritorio\Batch\ColorRed.cmd";
480                 }
481                 if (Color == 2)
482                 {
483                     // Cambiar el color del bombillo a Verde "G"
484                     rutaArchivoBat = @"C:\Users\mitch\OneDrive\Escritorio\Batch\ColorGreen.cmd";
485                 }
486                 if (Color == 3)
487                 {
488                     // Cambiar el color del bombillo a Azul "B"
489                     rutaArchivoBat = @"C:\Users\mitch\OneDrive\Escritorio\Batch\ColorBlue.cmd";
490                 }
491                 if (Color == 4)
492                 {
493                     // Cambiar el color del bombillo a Morado "M"
494                     rutaArchivoBat = @"C:\Users\mitch\OneDrive\Escritorio\Batch\ColorPurple.cmd";
495                 }
496                 ColorChange = "No";
497                 process.Start();
498                 process.StandardInput.WriteLine($"\"{rutaArchivoBat}\"");
499                 process.Close();
500             }
501
502
503         }
```

Este bloque de código maneja el cambio directo al color blanco.

- Cuando se detecta el gesto "Crown", se selecciona el archivo de lote correspondiente al color blanco y se ejecuta para cambiar el color del bombillo.
- Luego, la variable Color se establece en 0 para indicar que el color del bombillo se ha cambiado a blanco.

Figura 79. Reiniciando el color a blanco, Lightbulb Control

```
504
505             ////////////////////////////////////// Volver al Color Blanco ////////////////////////////////////////////
506             if (args.GestureSegment.Name == "Crown")
507             {
508                 string rutaArchivoBat2 = @"C:\Users\mitch\OneDrive\Escritorio\Batch\ColorWhite.cmd";
509                 process2.Start();
510                 process2.StandardInput.WriteLine($"\"{rutaArchivoBat2}\"");
511                 process2.Close();
512                 Color = 0;
513             }
514         }
515     }
516 }
517
```

3.5.8. Interfaz de Usuario.

Este método se activa cuando alguien presiona una tecla en el formulario. Si alguien toca una tecla, este método sabe qué tecla fue y qué hacer en respuesta a eso.

La variable Resource almacena la dirección de la carpeta de recursos de la aplicación.

Se verifica si la tecla presionada es la tecla "8" en el teclado numérico si es positivo se procede a comprobar el nivel de brillo.

Comprobación de nivel de brillo:

Si el nivel de brillo es 100, se carga la imagen correspondiente al brillo completo.
Si el nivel de brillo es 80, se carga la imagen correspondiente al brillo del 80%.
Si el nivel de brillo es 60, se carga la imagen correspondiente al brillo del 60%.
Si el nivel de brillo es 40, se carga la imagen correspondiente al brillo del 40%.
Si el nivel de brillo es 20, se carga la imagen correspondiente al brillo del 20%.
Si el nivel de brillo es 0, se carga la imagen correspondiente al brillo mínimo.

Se realiza una actualización de la imagen de brillo:
Para cada caso, se carga la imagen correspondiente utilizando la ruta de la carpeta de recursos concatenada con el nombre de archivo específico para cada nivel de brillo. Después, se actualiza el control PictureBox para mostrar la nueva imagen de brillo.

Se actualiza la imagen de la actividad si se detectan cambios en el brillo o el color.
Si se detecta un cambio en el brillo o el color, se carga una nueva imagen que refleja que no se han guardado los cambios.
Además, se establecen las propiedades e.Handled y e.SuppressKeyPress para evitar que el formulario maneje la tecla presionada, lo que ayuda a prevenir acciones no deseadas.

Figura 80. Cambiando el brillo en la interfaz, Lightbulb Control

```
//Este método se activa cuando alguien presiona una tecla en el formulario.
private void Form1_KeyDown(object sender, KeyEventArgs e)
{
    //Variable que contiene la direccion donde se encuentran los recursos medios a usar en el formulario
    string Resour = "C:\\Users\\mitch\\source\\repos\\DJxRott\\KinectProject\\BulbControlForm\\Resources\\";

    if (e.KeyCode == Keys.D8)
    {
        if(Brightness == 100)
        {
            this.PBBrillo.Image = new Bitmap( Resour + "Brightness_100.jpg");
            PBBrillo.Refresh();
        }
        if (Brightness == 80)
        {
            this.PBBrillo.Image = new Bitmap( Resour + "Brightness_80.jpg");
            PBBrillo.Refresh();
        }
        if (Brightness == 60)
        {
            this.PBBrillo.Image = new Bitmap( Resour + "Brightness_60.jpg");
            PBBrillo.Refresh();
        }
        if (Brightness == 40)
        {
            this.PBBrillo.Image = new Bitmap( Resour + "Brightness_40.jpg");
            PBBrillo.Refresh();
        }
        if (Brightness == 20)
        {
            this.PBBrillo.Image = new Bitmap( Resour + "Brightness_20.jpg");
            PBBrillo.Refresh();
        }
```

Figura 81. Confirmando cambio del brillo en la interfaz, Lightbulb Control

```
        if (Brightness == 0)
        {
            this.PBBrillo.Image = new Bitmap( Resour + "Brightness_0.jpg");
            PBBrillo.Refresh();
        }

        if(BrightnessChange == "Yes" || ColorChange == "Yes")
        {
            this.PBActua.Image = new Bitmap( Resour + "ChangesNS.jpg");
            PBActua.Refresh();
        }

        //C:\Users\mitch\source\repos\DJxRott\KinectProject\UIControl\Resources
        e.Handled = true;
        e.SuppressKeyPress = true;
    }
```

Se verifica si la tecla presionada es la tecla "7" en el teclado numero y si es positivo se procede a realizar las siguientes acciones basadas en el valor de Toggle:

Si el valor de Toggle es igual a 1, se carga la imagen "ON.png" en el cuadro de imagen PBoxOnOff.
Si el valor de Toggle es igual a 0, se carga la imagen "OFF.png" en el cuadro de imagen PBoxOnOff.

Además, se establecen las propiedades e.Handled y e.SuppressKeyPress para evitar que el formulario maneje la tecla presionada, lo que ayuda a prevenir acciones no deseadas.

Figura 82. Cambiando el estado Encendido/Apagado en la interfaz, Lightbulb Control

```
247
248             if (e.KeyCode == Keys.D7)
249             {
250                 if (Toggle == 1)
251                 {
252                     this.PBoxOnOff.Image = new Bitmap( Resour + "ON.png");
253                     PBoxOnOff.Refresh();
254                 }
255                 else if (Toggle == 0)
256                 {
257                     this.PBoxOnOff.Image = new Bitmap( Resour + "OFF.png");
258                     PBoxOnOff.Refresh();
259                 }
260
261                 //C:\Users\mitch\source\repos\DJxRott\KinectProject\UIControl\Resources
262                 e.Handled = true;
263                 e.SuppressKeyPress = true;
264             }
```

Se verifica si la tecla presionada es la tecla "3" en el teclado numérico. Si se detecta esta tecla, se realizan las siguientes acciones basadas en el valor de Color:

Si el valor de Color es igual a 0, se carga la imagen "ColorW.jpg" en el control PictureBox PBColor para representar el color blanco.
Si el valor de Color es igual a 1, se carga la imagen "ColorR.jpg" en PBColor para representar el color rojo.
Si el valor de Color es igual a 2, se carga la imagen "ColorG.jpg" en PBColor para representar el color verde.
Si el valor de Color es igual a 3, se carga la imagen "ColorB.jpg" en PBColor para representar el color azul.
Si el valor de Color es igual a 4, se carga la imagen "ColorM.jpg" en PBColor para representar el color magenta.

Luego de realizar estas operaciones, se verifica si la variable BrightnessChange es "Yes" o si la variable ColorChange es "Yes". En caso afirmativo, se carga la imagen "ChangesNS.jpg" en el control PictureBox PBActua para indicar que se han realizado cambios en el brillo o el color.

Finalmente, se establecen las propiedades e.Handled y e.SuppressKeyPress como verdaderas para evitar que el formulario maneje la tecla presionada y se realicen acciones no deseadas.

Figura 83. Cambiando el color en la interfaz, Lightbulb Control

```
266         if (e.KeyCode == Keys.D3)
267         {
268             if (Color == 0)
269             {
270                 this.PBColor.Image = new Bitmap( Resour + "ColorW.jpg");
271                 PBColor.Refresh();
272             }
273             if (Color == 1)
274             {
275                 this.PBColor.Image = new Bitmap( Resour + "ColorR.jpg");
276                 PBColor.Refresh();
277             }
278             if (Color == 2)
279             {
280                 this.PBColor.Image = new Bitmap( Resour + "ColorG.jpg");
281                 PBColor.Refresh();
282             }
283             if (Color == 3)
284             {
285                 this.PBColor.Image = new Bitmap( Resour + "ColorB.jpg");
286                 PBColor.Refresh();
287             }
288             if (Color == 4)
289             {
290                 this.PBColor.Image = new Bitmap( Resour + "ColorM.jpg");
291                 PBColor.Refresh();
292             }//ColorM
293 
294             if (BrightnessChange == "Yes" || ColorChange == "Yes")
295             {
296                 this.PBActua.Image = new Bitmap( Resour + "ChangesNS.jpg");
297                 PBActua.Refresh();
298             }
299 
300             //C:\Users\mitch\source\repos\DJxRott\KinectProject\UIControl\Resources
301             e.Handled = true;
302             e.SuppressKeyPress = true;
```

Se verifica si la tecla presionada es la tecla "9" en el teclado numérico. Si se detecta esta tecla, se realiza la siguiente acción:

Se carga la imagen "gud.jpg" en el control PictureBox PBActua.
Luego de cargar la imagen, se refresca el control PictureBox para que se muestre la nueva imagen.

Figura 84. Confirmando cambio en la interfaz, Lightbulb Control

```
306             if (e.KeyCode == Keys.D9)
307             {
308                 this.PBActua.Image = new Bitmap(Resour + "gud.jpg");
309                 PBActua.Refresh();
310             }
311 
312         }
```

Figura 85. Interfaz del Módulo, Lightbulb Control

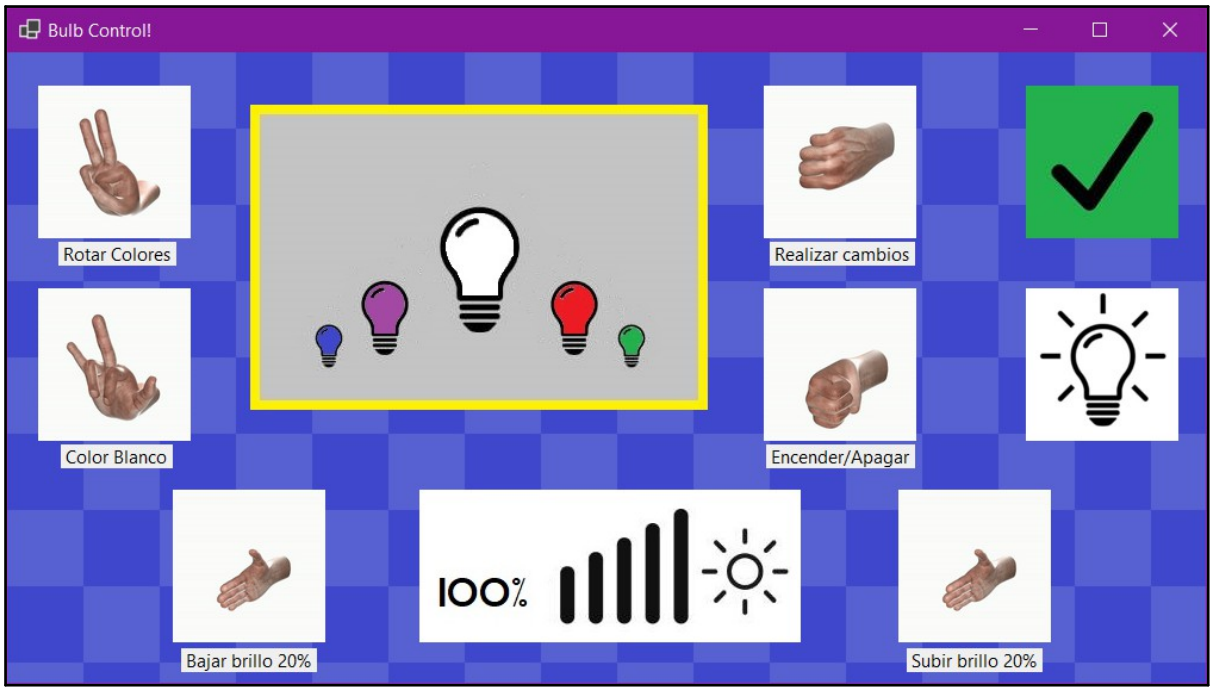


Figura 86. Diagrama de función, Lightbulb Control

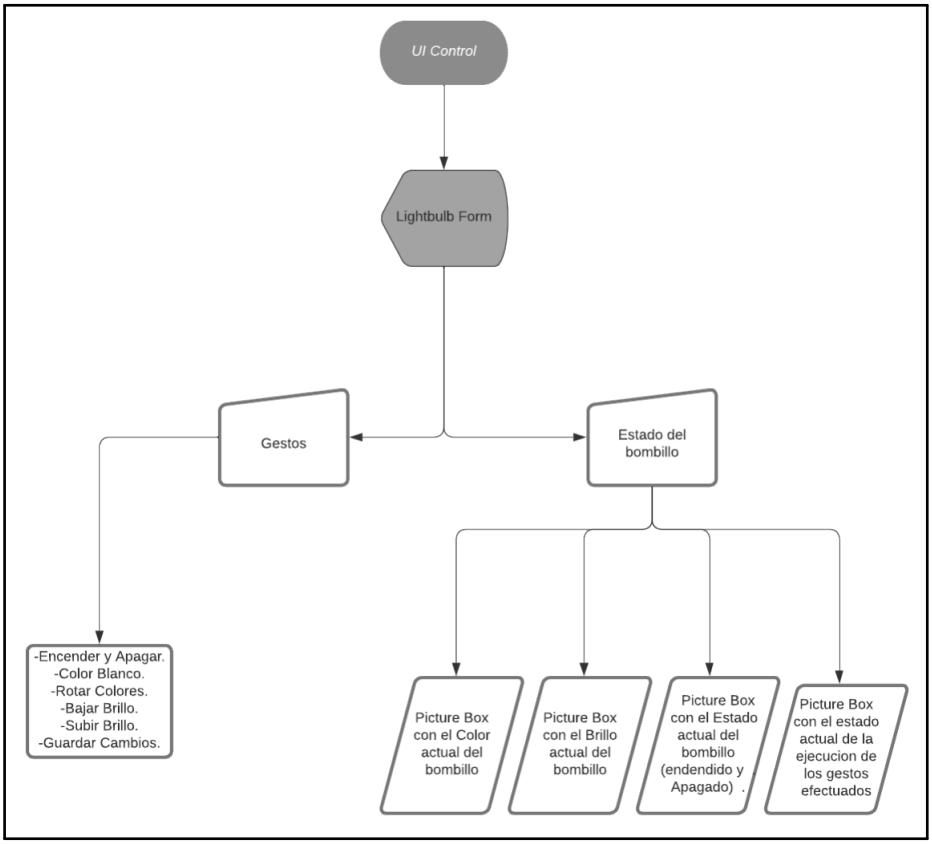
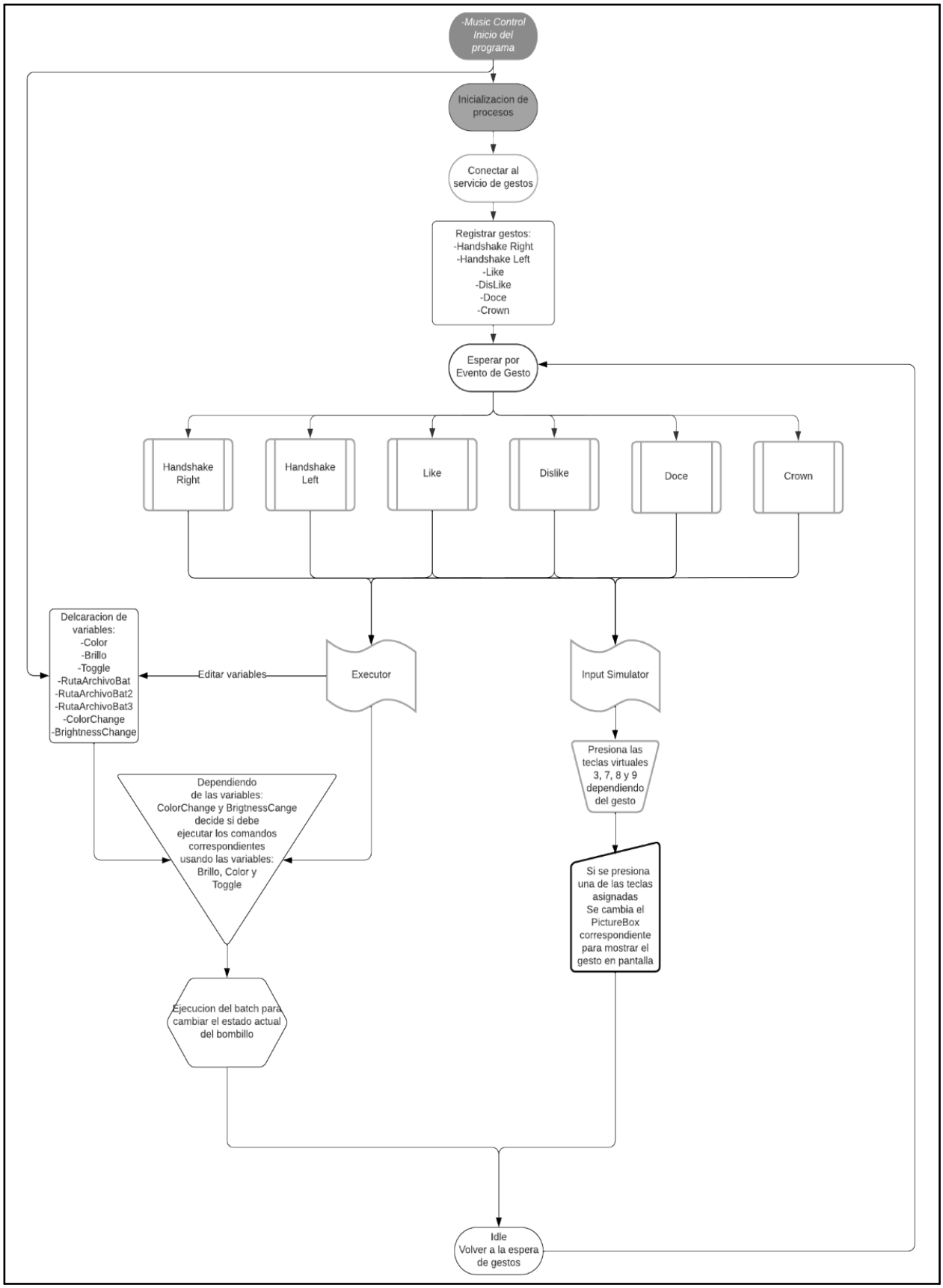


Figura 87. Diagrama de la ventana, Lightbulb Control



3.6. Menú Principal

Estas líneas de código importan las bibliotecas del sistema y otras bibliotecas específicas para la gestión de aplicaciones gráficas, el reconocimiento de gestos y la gestion del metodo asincrono.

- System.Threading: Proporciona clases y tipos que admiten la programación multiproceso.
- System.Drawing: Contiene clases que permiten la manipulación de imágenes, gráficos y dibujos.
- System: Contiene clases fundamentales y tipos básicos del lenguaje C#.
- System.Threading.Tasks: Proporciona tipos que simplifican la implementación de operaciones asincrónicas y paralelas.
- System.Windows.Forms: Contiene clases para crear aplicaciones de Windows Forms.

Estas directivas son esenciales para trabajar con elementos gráficos y realizar tareas asincrónicas en una aplicación de Windows Forms.

Figura 88. Directivas del sistema, Menú Principal

```
1   using System.Threading;
2   using System.Drawing;
3   using System;
4   using System.Threading.Tasks;
5   using System.Windows.Forms;
6
```

Este fragmento de código se ejecuta cuando el formulario se carga por primera vez. Aquí se configuran y personalizan varios botones para que tengan una apariencia triangular:

-Creación del gráfico del botón: Se utiliza la clase GraphicsPath para crear un objeto que represente la forma del botón. En este caso, se utilizan puntos para definir los vértices del triángulo que conformará el botón.
-Definición de los puntos del triángulo: Se definen los puntos que forman el triángulo en función de las coordenadas x e y. Estos puntos se utilizan para crear el gráfico del botón.
-Adición de líneas al gráfico del botón: Se agregan las líneas que conectan los puntos definidos anteriormente al gráfico del botón utilizando el método AddLines del objeto GraphicsPath.
-Dimensionamiento y posicionamiento del botón: Se establece el tamaño y la posición del botón mediante la propiedad Size y se asigna el gráfico del botón al botón visual mediante la propiedad Region. Esto asegura que el botón tenga la forma del triángulo definido anteriormente.

-Asignación de eventos del ratón: Se asignan manejadores de eventos para los eventos de entrada y salida del ratón (entrar y salir del área del botón). Estos eventos controlan la apariencia del botón cuando el cursor del ratón entra o sale del área del botón, proporcionando retroalimentación visual al usuario.

Los botones son los siguientes:
Claro, aquí tienes los botones ordenados por su función:

1. BMediaControl: Botón de Control de Música
2. BBulbControl: Botón de Control de Bombillo
3. BAlphabetControl: Botón del Control de Alfabeto
4. BSlideControl: Botón de Control de Diapositivas

Estos botones proporcionan una interfaz visual única y distintiva para interactuar con diferentes funcionalidades dentro de la aplicación.

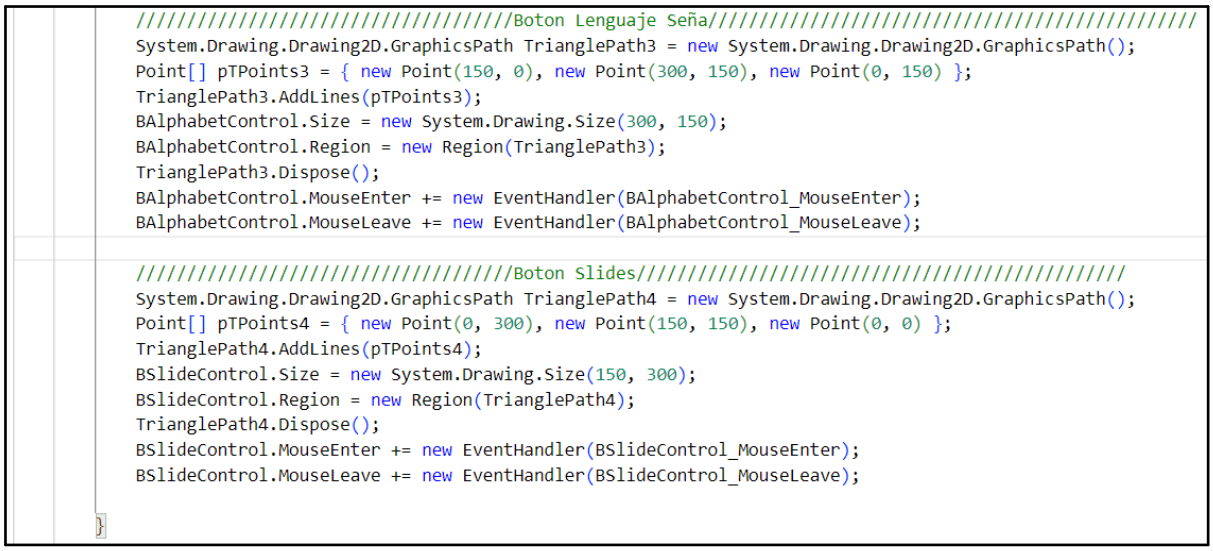
Figura 89. Creando los botones del Menú Principal

```
namespace UIControl
{
    public partial class KinectProject : Form
    {
        private static KinectProject _instance;

        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Kinect Project Test";
            ////////////////////////////////////Boton Musica//////////////////////////////////////////////////
            System.Drawing.Drawing2D.GraphicsPath TrianglePath = new System.Drawing.Drawing2D.GraphicsPath();
            Point[] pTPoints = { new Point(150, 150), new Point(300, 0), new Point(0, 0) };
            TrianglePath.AddLines(pTPoints);
            BMediaControl.Size = new System.Drawing.Size(300, 150);
            BMediaControl.Region = new Region(TrianglePath);
            TrianglePath.Dispose();
            BMediaControl.MouseEnter += new EventHandler(BMediaControl_MouseEnter);
            BMediaControl.MouseLeave += new EventHandler(BMediaControl_MouseLeave);

            ////////////////////////////////////Boton Bombillo//////////////////////////////////////////////////
            System.Drawing.Drawing2D.GraphicsPath TrianglePath2 = new System.Drawing.Drawing2D.GraphicsPath();
            Point[] pTPoints2 = { new Point(150, 300), new Point(150, 0), new Point(0, 150) };
            TrianglePath2.AddLines(pTPoints2);
            BBulbControl.Size = new System.Drawing.Size(150, 300);
            BBulbControl.Region = new Region(TrianglePath2);
            TrianglePath2.Dispose();
            BBulbControl.MouseEnter += new EventHandler(BBulbControl_MouseEnter);
            BBulbControl.MouseLeave += new EventHandler(BBulbControl_MouseLeave);
```

Figura 90. Cont. Creando los botones del Menú Principal



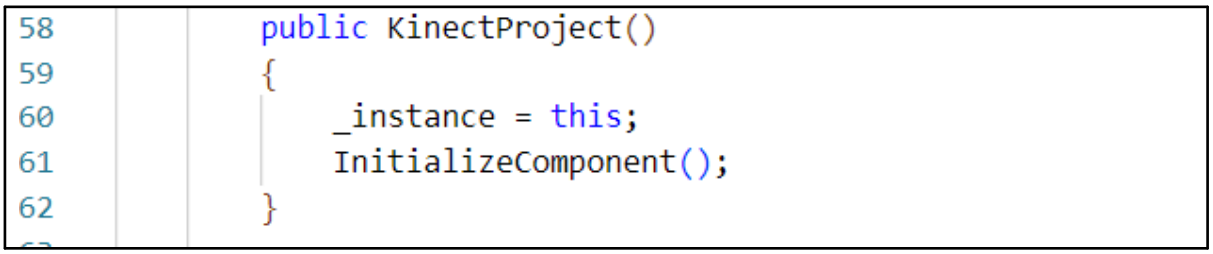
```
////////////////////////////////////Boton Lenguaje Seña/////////////////////////////////////////////////
System.Drawing.Drawing2D.GraphicsPath TrianglePath3 = new System.Drawing.Drawing2D.GraphicsPath();
Point[] pTPoints3 = { new Point(150, 0), new Point(300, 150), new Point(0, 150) };
TrianglePath3.AddLines(pTPoints3);
BAlphabetControl.Size = new System.Drawing.Size(300, 150);
BAlphabetControl.Region = new Region(TrianglePath3);
TrianglePath3.Dispose();
BAlphabetControl.MouseEnter += new EventHandler(BAlphabetControl_MouseEnter);
BAlphabetControl.MouseLeave += new EventHandler(BAlphabetControl_MouseLeave);

////////////////////////////////////Boton Slides/////////////////////////////////////////////////
System.Drawing.Drawing2D.GraphicsPath TrianglePath4 = new System.Drawing.Drawing2D.GraphicsPath();
Point[] pTPoints4 = { new Point(0, 300), new Point(150, 150), new Point(0, 0) };
TrianglePath4.AddLines(pTPoints4);
BSlideControl.Size = new System.Drawing.Size(150, 300);
BSlideControl.Region = new Region(TrianglePath4);
TrianglePath4.Dispose();
BSlideControl.MouseEnter += new EventHandler(BSlideControl_MouseEnter);
BSlideControl.MouseLeave += new EventHandler(BSlideControl_MouseLeave);

}
```

Este constructor inicializa una instancia de la clase KinectProject. Establece la instancia _instance como la instancia actual de la clase KinectProject y luego inicializa los componentes del formulario mediante el método InitializeComponent().

Figura 91. Inicializando el Kinect para su uso, Menú Principal



```
58      public KinectProject()
59      {
60          _instance = this;
61          InitializeComponent();
62      }
```

Estos métodos controlan los eventos de entrada y salida del mouse para cada uno de los botones en el formulario. Cuando el mouse entra en el área del botón, se actualizan las etiquetas label1 y label2 para proporcionar información sobre la función del botón correspondiente. Cuando el mouse sale del área del botón, se llama al método Leave_F() para restaurar las etiquetas a su estado predeterminado.

Los métodos son:

BMediaControl_MouseEnter: Se activa cuando el mouse entra en el área del botón de control multimedia. Muestra información sobre el control de medios.

BMediaControl_MouseLeave: Se activa cuando el mouse sale del área del botón de control multimedia. Llama al método Leave_F().

BSlideControl_MouseEnter: Se activa cuando el mouse entra en el área del botón de control de presentaciones. Muestra información sobre el control de presentaciones.
BSlideControl_MouseLeave: Se activa cuando el mouse sale del área del botón de control de presentaciones. Llama al método Leave_F().

BBulbControl_MouseEnter: Se activa cuando el mouse entra en el área del botón de control de bombillas. Muestra información sobre el control de bombillas inteligentes.

BBulbControl_MouseLeave: Se activa cuando el mouse sale del área del botón de control de bombillas. Llama al método Leave_F().

BAlphabetControl_MouseEnter: Se activa cuando el mouse entra en el área del botón de control de lenguaje de señas. Muestra información sobre el control de lenguaje de señas.

BAlphabetControl_MouseLeave: Se activa cuando el mouse sale del área del botón de control de lenguaje de señas. Llama al método Leave_F().

Figura 92. Descripciones de los Módulos del Menú Principal

```
63
64      private void BMediaControl_MouseEnter(object sender, EventArgs e)
65      {
66          this.label1.Location = new Point(100, 11);
67          label1.Text = "Media Control";
68          this.label2.Location = new Point(11, 73);
69          label2.Text = "Integra gestos programados en tu sistema \r\npara tener el control total de funciones \r\nmultimedia
70              " permitiéndote\r\nreproducir, pausar, detener, subir y bajar \r\nvolumen de cualquier tipo de media que esté \
71              " ya sea vídeos en línea, \r\nreproductores de música o cualquier tipo de \r\nmedio audiovisual.";
72
73      }
74
75      private void BMediaControl_MouseLeave(object sender, EventArgs e)
76      {
77          Leave_F();
78      }
79
80      private void BSlideControl_MouseEnter(object sender, EventArgs e)
81      {
82          this.label1.Location = new Point(70, 11);
83          label1.Text = "PowerPoint Control";
84          this.label2.Location = new Point(11, 91);
85          label2.Text = "Utiliza diferentes movimientos y gestos para\r\nrealizar interacciones con aplicaciones " +
86              "de\r\npresentación de diapositivas, tales como Prezi\r\no PowerPoint. Esto incluye la integración " +
87              "de\r\ncomandos simples para avanzar y retroceder\r\ndiapositivas, así como para controlar el nivel\r\nde " +
88              "zoom dentro de cada diapositiva.";
89      }
90
91      private void BSlideControl_MouseLeave(object sender, EventArgs e)
92      {
93          Leave_F();
94      }
```

Figura 93. Cont. Descripciones de los Módulos del Menú Principal

```
96          private void BBulbControl_MouseEnter(object sender, EventArgs e)
97          {
98              this.label1.Location = new Point(85, 11);
99              label1.Text = "Lightbulb Control";
100             this.label2.Location = new Point(11, 91);
101             label2.Text = "Usando gestos programados utiliza \r\nconexiones con APIs de domótica para \r\ncontrolar " +
102                 "bombillos inteligentes, permitiendo \r\nencender y apagar de forma remota cada \r\nuno, además de alterar " +
103                 "a voluntad los colores \r\ny el brillo que mostrarán los que esten \r\nconectados a la misma red.";
104         }
105
106         private void BBulbControl_MouseLeave(object sender, EventArgs e)
107         {
108             Leave_F();
109         }
110
111         private void BAlphabetControl_MouseEnter(object sender, EventArgs e)
112         {
113             this.label1.Location = new Point(90, 11);
114             label1.Text = "Alphabet Control";
115             this.label2.Location = new Point(11, 91);
116             label2.Text = "Por medio de gestos que corresponden al \r\nabecedario del lenguaje de señas americano " +
117                 "\r\n(con ligeras modificaciones) permite el uso de \r\nun teclado virtual simulado que se pueda " +
118                 "\r\nutilizar en cualquier otra aplicación dentro del \r\ncomputador.";
119         }
120
121         private void BAlphabetControl_MouseLeave(object sender, EventArgs e)
122         {
123             Leave_F();
124         }
```

El método Leave_F restaura las etiquetas label1 y label2 a su estado predeterminado cuando el mouse sale del área de cualquier botón. Esto proporciona una descripción general de la aplicación y cómo obtener más información sobre las funciones disponibles.

Figura 94. Texto de presentación del Menú Principal

```
126         private void Leave_F()
127         {
128             this.label1.Location = new Point(40, 11);
129             label1.Text = "Kinect Control All in one!";
130             this.label2.Location = new Point(65, 73);
131             label2.Text = "Pon el mouse sobre cualquier \r\nboton para obtener \r\ninformacion sobre la aplicacion";
132
133         }
134
```

Estos métodos manejan los eventos de clic en los botones BMediaControl, BSlideControl, BAlphabetControl y BBulbControl.

Cada uno de estos métodos crea una nueva instancia de un formulario específico y lo muestra cuando el botón correspondiente es clicado, permitiendo así la navegación a las diferentes funcionalidades de la aplicación.

Figura 95. Comando para abrir cada módulo desde el Menú Principal

```
134
135         private void BMediaControl_Click(object sender, EventArgs e)
136         {
137             KMusicForm F1 = new KMusicForm();
138             F1.Show();
139         }
140
141         private void BSlideControl_Click(object sender, EventArgs e)
142         {
143             PPForm F4 = new PPForm();
144             F4.Show();
145         }
146
147         private void BAlphabetControl_Click(object sender, EventArgs e)
148         {
149             AlphaForm F3 = new AlphaForm();
150             F3.Show();
151         }
152
153         private void BBulbControl_Click(object sender, EventArgs e)
154         {
155             ControlBForm F2 = new ControlBForm();
156             F2.Show();
157         }
158     }
159
```

3.6.1 Diseños Prototipos

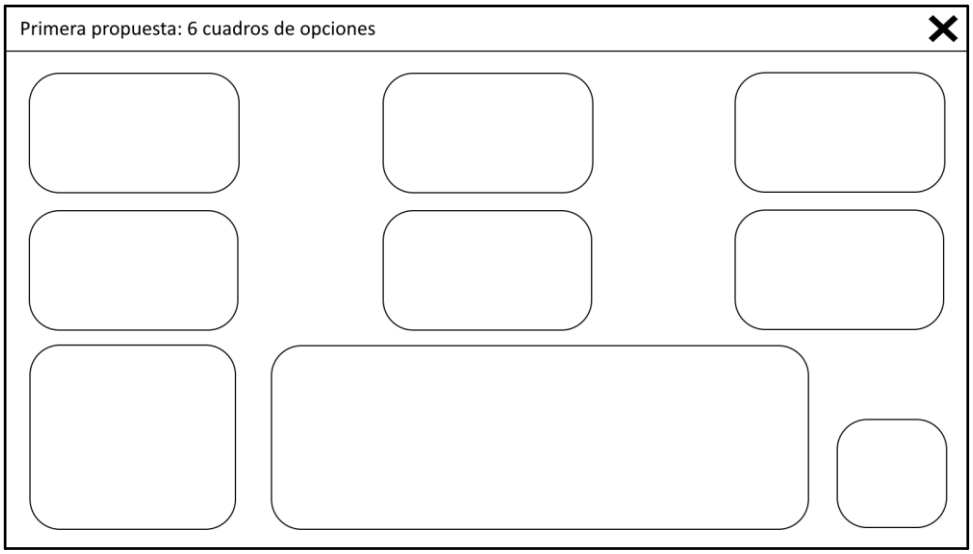
3.6.1.1. Primera propuesta

La primera iteración de la interfaz de usuario se realizó antes de delimitar el alcance de la aplicación que se iba a realizar. Dado que el modelo del desarrollo fue desarrollo rápido de aplicaciones, originalmente no se tenía un vistazo finalizado de cuantos módulos se iban a desarrollar, por lo que se hizo un preliminar bastante amplio para luego delimitarlo correctamente durante el proceso.

La primera imagen corresponde a la imagen general del producto, junto con la cantidad original de módulos que eran seis.

Dentro de la ventana de la app, se muestran 6 botones grandes en la parte superior de la ventana, con un cuadro abajo de los botones centrado en la ventana, con un cuadro mucho más pequeño a la izquierda del mismo y uno mucho más pequeño en la esquina inferior derecha.

Figura 96. Primera propuesta del Menú Principal



Esto con la finalidad de dar un énfasis en las opciones disponibles (que el usuario realmente sólo puede desplazarse por los 6 botones superiores) mientras se muestra el resto de la información de forma visual en los cuadros abajo mencionados.

Cada uno de los botones corresponde a una opción del programa, que permite controlar este aspecto específico de cada una de las opciones mostradas, de forma que se puede tener un control específico para una acción en concreto sin necesidad de forzar ni al usuario a recordar una enorme cantidad de comandos, ni a la aplicación que utilice muchos recursos a la vez.

En el ejemplo a continuación, se muestran varias de las opciones delimitadas para uso del proyecto hasta el momento. Entre las opciones, se encuentran representados varias acciones generales que el usuario puede elegir hacer en cualquier momento, agrupando varias de las acciones en un solo botón para evitar que el usuario tenga que estar cambiando constantemente de ventanas. Entre las opciones se encuentra: Media (Reproducción de comandos de control de música), Slides control (Control de Powerpoint), Juegos (Por el momento únicamente Piedra, Papel o Tijeras), Control de Hogar (Domótica entre varias opciones de seguridad), Teclado (Función de detección de letras del alfabeto de señas) y Comandos de PC (Funciones varias del Windows).

En la parte inferior izquierda de la ventana, se mostraría en el cuadro los comandos disponibles en este momento para moverse a través de las opciones y seleccionar las mismas. A la derecha de este cuadro de los comandos con los gestos, aparece un cuadro de descripción, en el que se muestra un pequeño texto con la información general de la opción seleccionada.

Finalmente, hay un botón específico de Opciones (Reservado para un gesto en específico, accesible con un gesto cada vez que quiera) por el cual permitirá cambiar los gestos por defecto que se incluyen en el servicio.

Este prototipo fue desechado por varios motivos, los cuales son listados a continuación:

- Dos de los módulos pensados originalmente no quedaron en la aplicación final: El de comandos de Windows era un poco redundante y no era cómodo de utilizar, y el módulo de juegos no presentaba ningún uso útil para el usuario, por lo que la siguiente propuesta tendría los módulos que ya se estaban trabajando en ese momento.

- El diseño estaba mal distribuido, dado que al dar prioridad a las opciones superiores, no quedaba claro que la parte inferior podía ser un botón físico o activado por gestos.

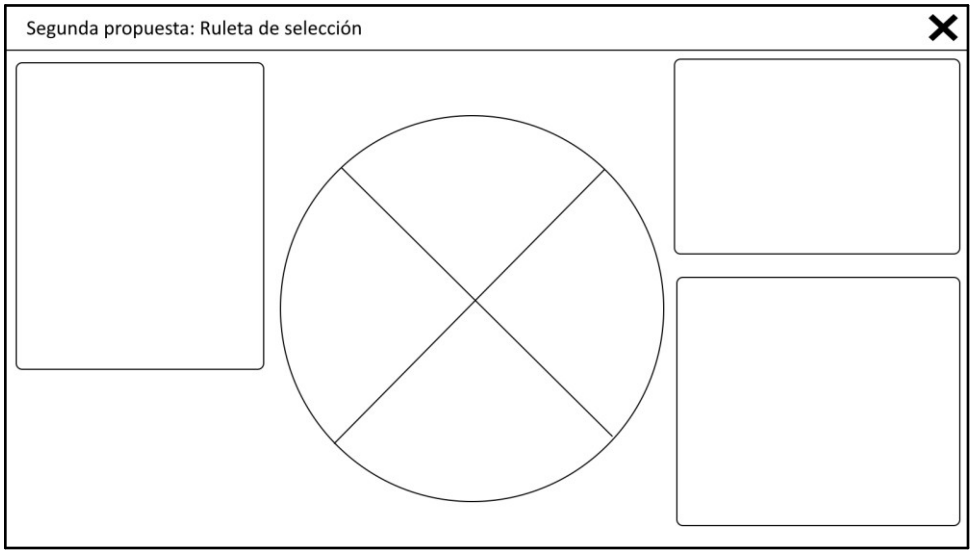
- Dado que se necesitaba instruir al usuario acerca de como moverse en la aplicación desde el momento de iniciar la ventana, la implementación de los gestos no estaba clara y esta disposición no permite visualizar todos los comandos a la vez, por lo que era muy engorroso utilizarlo.

3.6.1.2. Segunda propuesta

Una vez delimitado el programa y los módulos que queríamos implementar, pudimos volver a la interfaz para una actualización, con algunos cambios. Teniendo siempre la intención de mantener los botones de los módulos como lo principal, que se pudieran utilizar los gestos desde el primer momento y que fuera ordenado y atractivo para el usuario, se decidió probar una propuesta adicional.

En la siguiente imagen, se puede ver una ventana de aplicación de Windows. Ahora los 4 módulos están distribuidos en una ruleta o rueda de selección, los cuales son seleccionables por medio de gestos. Se incluye ahora un cuadro a la izquierda que mostrara información de los gestos disponibles dentro del módulo seleccionado, y dos cuadros a la derecha de la rueda de selección. El primero de arriba para abajo contiene los gestos que estan rotando con las opciones disponibles, y el de abajo de este, una descripción del módulo en sí.

Figura 97. Segunda propuesta del Menú Principal



Los módulos delimitados al final son: Media Control, Slideshow Control, Control de Hogar

y Control de Teclado. Así mismo, nos centramos en hacer una ventana a cada uno una vez que estuvieran terminados e implementados correctamente.

Originalmente un gesto sería para elegir la opción de la ruleta, y otro gesto sería usado luego de ese para “Confirmar” la selección. Esto permitiría tener más control sobre el gesto que seguía a continuación en el cuadro superior derecho, además de poder mostrar correctamente la descripción en el cuadro inferior y mostrar una lista de los gestos utilizados dentro de cada módulo en la izquierda.

A pesar de que era una mejora respecto al anterior, de igual forma presentamentamos unos problemas nuevos además de no presentar un orden amigable al primer vistazo, dado que mostraría demasiada información enseguida sin un orden de qué ver primero.

Entre los otros detalles con este prototipo, se muestra lo siguiente:

- A pesar de que la ruleta es el punto principal de la ventana, desplaza todos los demás cuadros de una forma irregular que no es agradable a la vista ni amigable a la experiencia del usuario.

- Uno de los cuadros de gesto es innecesario dado que los gestos no están listos para ejecutarse antes de elegir alguna opción, por lo que podría confundir al usuario y pensar que el gesto ya está listo para funcionar.

3.6.1.3. Tercera propuesta

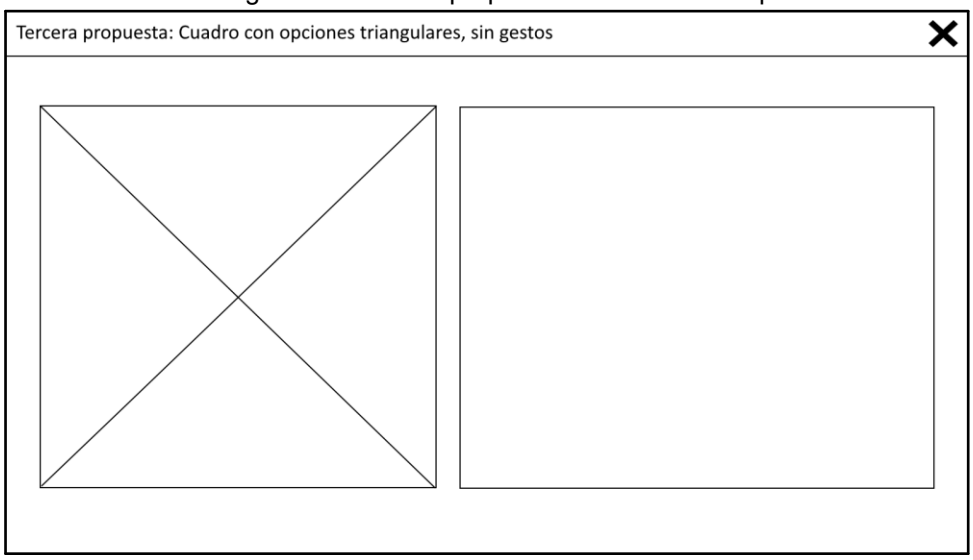
Finalmente, al mover y jugar un poco con diferentes diseños, al final se descartó la idea de elegir opciones del menú utilizando gestos. Las razones para ello son de que una vez terminado el desarrollo de todos los módulos, vimos que utilizar gestos para el menú principal también, a pesar de que era totalmente posible, iba a resultar en llenar el menú principal de muchos más gestos y manejar algún posible conflicto entre los mismos o la molestia del usuario, por lo que la ventana principal es la única en la que utiliza el mouse para abrir los demás módulos.

Además permite entrar mucho más fácilmente al contenido de la aplicación al no necesitar un gesto de inmediato, y permitir al usuario utilizar el mouse mientras se acostumbra al movimiento dentro de la aplicación.

La siguiente imagen describe el croquis final de la ventana principal: Cuatro botones con forma de triángulos isosceles rectángulos acomodados en forma de un cuadrado en la parte izquierda de la ventana, con un cuadro grande a la derecha que muestra una descripción del módulo al colocar el mouse encima de alguna de las opciones.

Finalmente este es el croquis del modelo que se utilizó para la aplicación final. Esta disposición requiere de utilizar el Mouse solo en esta ventana, pero una vez que cada módulo es abierto, queda listo para utilizarse sin necesidad de utilizar esta ventana más que para volver a abrir el módulo nuevamente.

Figura 98. Tercera propuesta del Menú Principal



3.6.2. Interfaz de Usuario Final.

Figura 99. Interfaz por defecto del Menú Principal

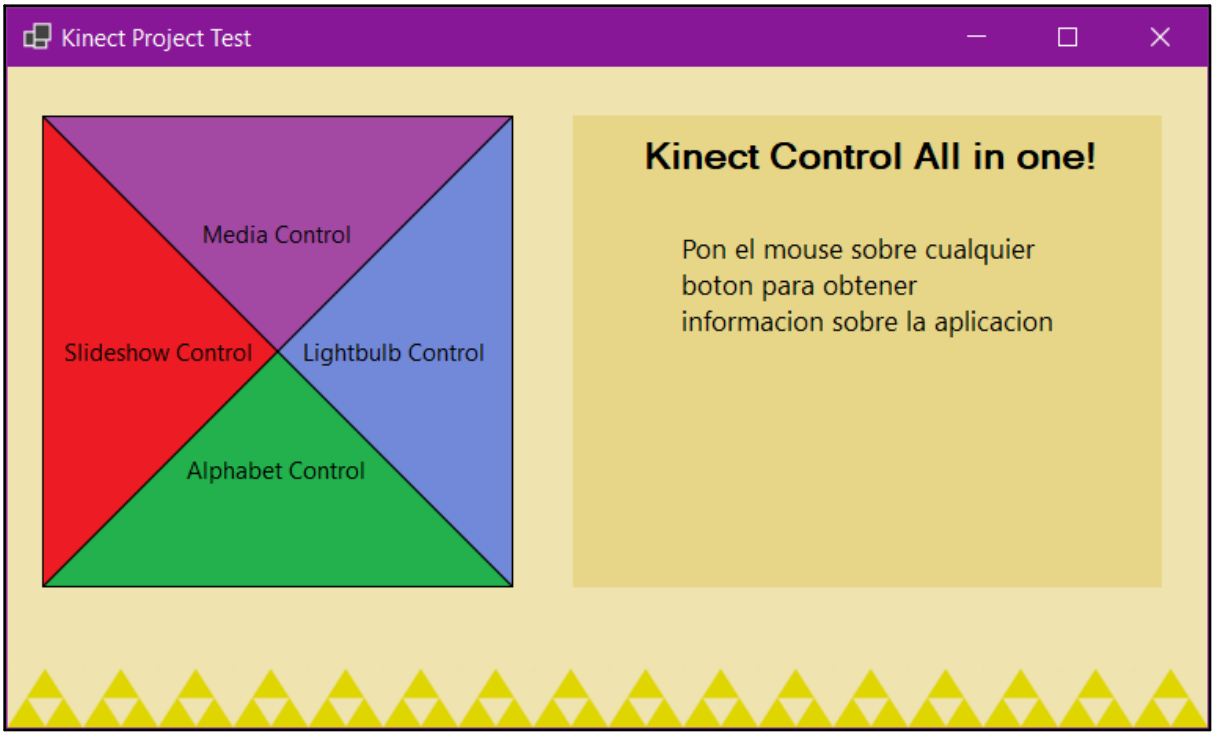


Figura 100. Interfaz sobre Music Control del Menú Principal

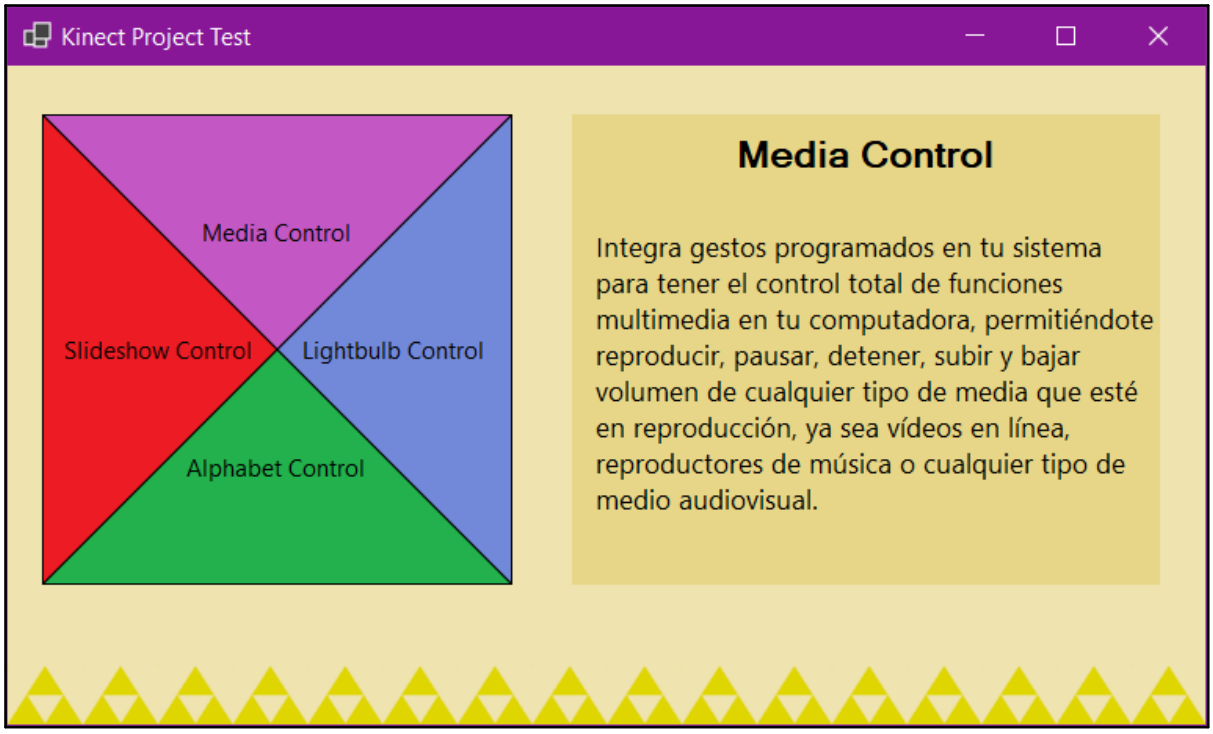


Figura 101. Interfaz sobre Slideshow Control del Menú Principal

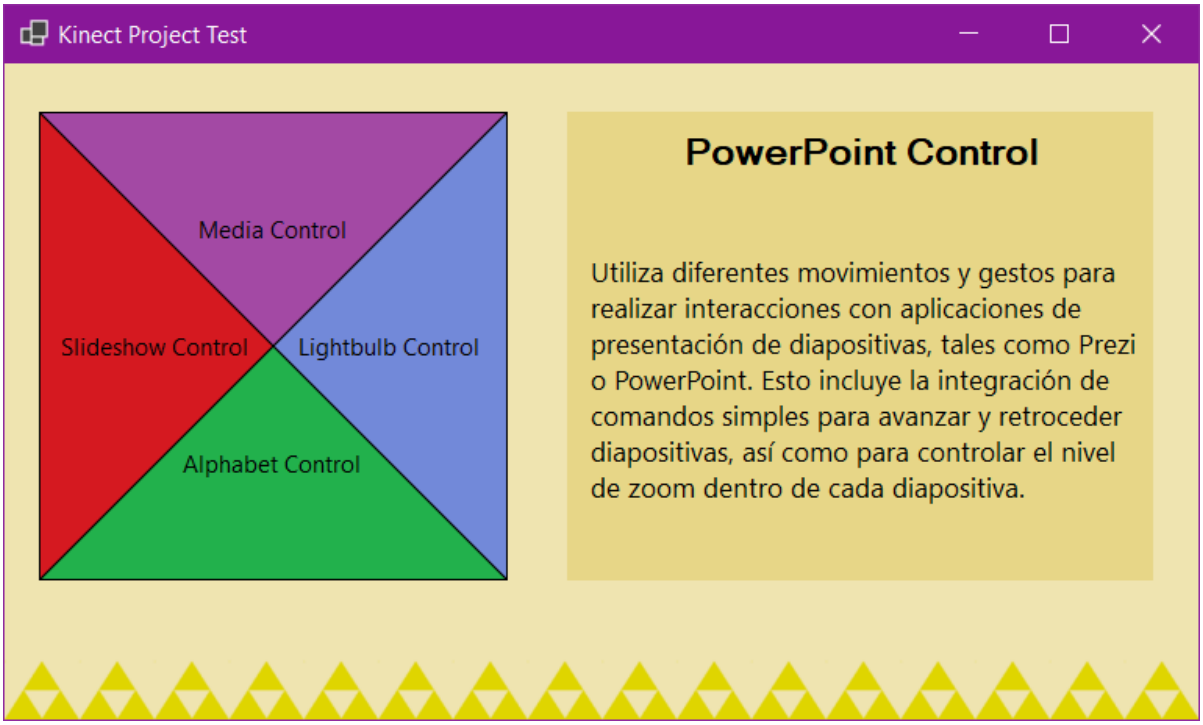


Figura 102. Interfaz sobre Alphabet Control del Menú Principal

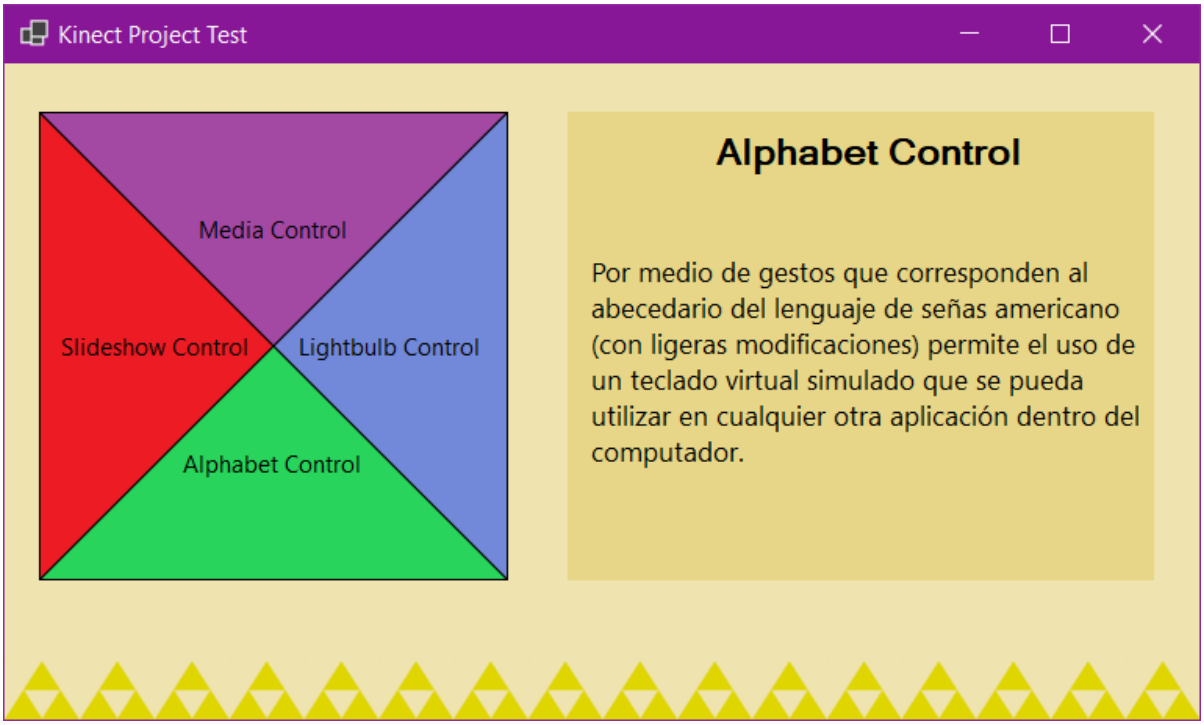
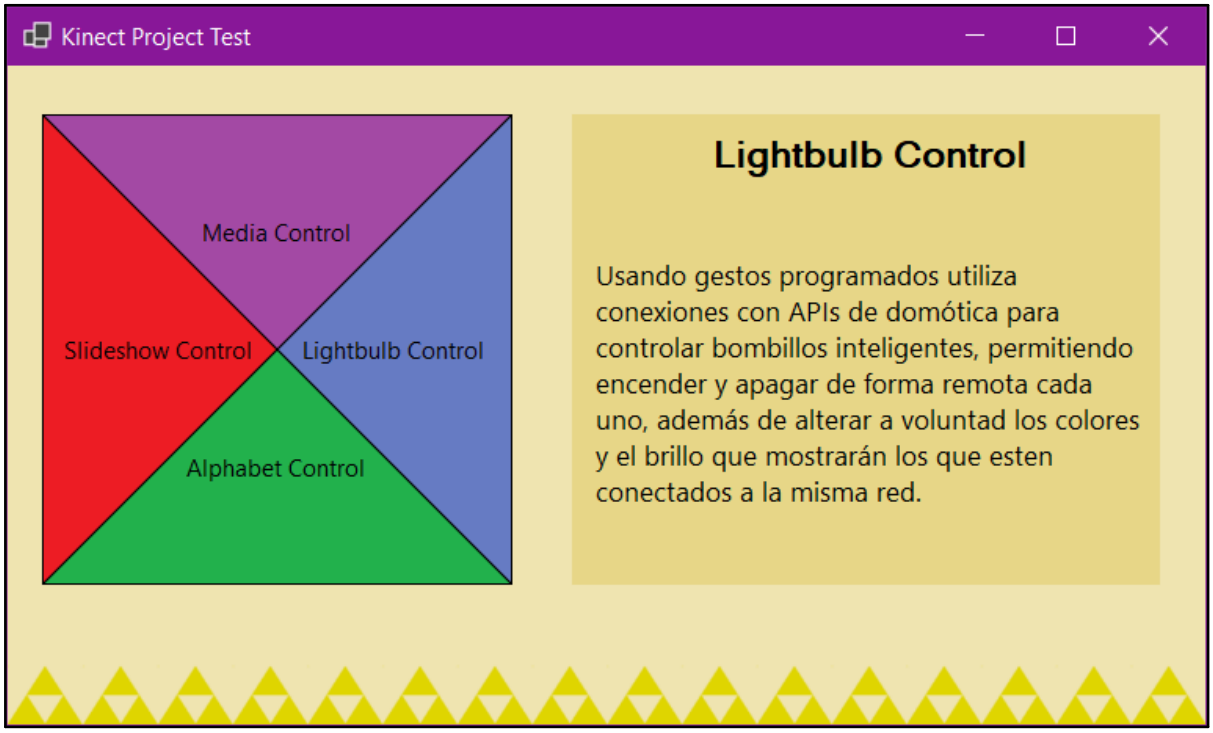


Figura 103. Interfaz sobre Lightbulb Control del Menú Principal



3.7. Gestos

3.7.1. Creación de los gestos.

Durante el proceso de desarrollo de nuestra aplicación, nos encontramos con un desafío significativo que impactó directamente en la eficiencia de nuestro equipo. Aproximadamente a la mitad del desarrollo, nos percatamos de que Visual Studio 2019 carecía de soporte para GitHub. Esta limitación se tradujo en la imposibilidad de realizar operaciones esenciales como la extracción y carga de código a través de la plataforma GitHub, lo cual era fundamental para nuestra colaboración y gestión del proyecto.

La solución que encontramos fue migrar toda la estructura de desarrollo de la aplicación a Visual Studio 2022, una versión más actualizada que ofrecía la compatibilidad necesaria con GitHub. Este detalle afectó el desarrollo de todo el proyecto, dado que se tuvo que actualizar a su vez los gestos que llevábamos realizados y actualizar gran parte de la librería que utilizaba el Microsoft.Gesture.

El proyecto Praga engloba diversas aplicaciones que se instalan por defecto durante el proceso de instalación de Microsoft Gestures. Estas aplicaciones fueron originalmente desarrolladas por el equipo de proyecto Praga mientras estaba en vigencia dentro de Microsoft. Sin embargo, cabe señalar que estas aplicaciones presentan ciertas limitaciones, como interfaces de usuario poco intuitivas y en general una experiencia en estado de acceso anticipado que no estaba destinada a ser visualizada por el público en ese momento. Inspirados por el trabajo previo de este equipo, decidimos tomar como referencia lo que habían desarrollado pero que se encontraba inconcluso. Este fue el punto de partida para nuestro propio desarrollo de programas.

En el contexto del proyecto Praga, también se incluían instrucciones detalladas sobre cómo implementar y reconocer nuevos gestos. Estas instrucciones proporcionaban indicaciones específicas sobre qué acciones debían realizar cada dedo para identificar correctamente un gesto. Entre las indicaciones se incluían aspectos como la dirección hacia la cual estaba orientada la palma, la disposición de los dedos (ya sea estirados o enrollados), la relación espacial entre ellos, como si alguno estaba encima de otro o si se tocaban entre sí.

Otro problema presentado en el tiempo de desarrollo, fue el de crear y administrar los gestos en sí. Todos los gestos utilizados son únicamente con la mano derecha, utilizando la serie de comandos explicados anteriormente. Sin embargo, a pesar de la gran cantidad de opciones disponibles con dichos márgenes, de igual manera era una limitación debido a que no es una mano suspendida sin nada atado, tiene que estar limitado a lo que puede hacer con la muñeca y el brazo del usuario, por lo que a pesar de que se podían registrar gestos únicos, es imposible de realizar un gesto con la palma de la mano derecha viendo hacia la derecha, o hacer que el dedo índice tocara la parte posterior del dedo meñique, por nombrar algunas situaciones.

Con eso dicho, utilizando gestos universales como el Me Gusta o el Rock, gestos de lenguajes de señas de diferentes idiomas, o incluso gestos que se generan al usar números binarios con la mano, logramos realizar 43 gestos para ser utilizados en la

aplicación, muchos de ellos relacionados entre sí por contexto, algunos con traducciones literales de funciones en la vida real y algunos otros que encajaban bien, de esta forma garantizando que no se fueran a repetir o entrelazar unos con otros.

Fue necesaria una manera clara de mostrar los gestos para el usuario final, tanto como si conoce el gesto al que se refiere, como si lo está viendo por primera vez, necesitábamos mostrar qué movimiento está registrando la cámara de profundidad para que el gesto se ejecute de forma correcta.

Se utilizó el modelo 3D de mano realizado por el usuario "3dhaupt" para licencia de uso personal. Este era el único modelo gratuito que pudimos encontrar para utilizar en Blender, que es el programa que se aprendió a utilizar en la carrera. El nombre del archico es "Rigged Hands 3D Model", sin embargo los huesos de dicho modelo no estan definidos para Blender, por lo que a pesar de que el modelo y la textura fueron realizados por el usuario 3dhaupt, realizar el esqueleto de la mano derecha, eliminar el vínculo que existía con la mano izquierda, definir cada uno de los dedos individualmente para que realizaran todos los gestos necesarios y renderizar los 43 gestos a mano fue realizado por nosotros para este proyecto.

A continuación se muestran unas capturas del proyecto y sus pasos:

Figura 104. Captura de pantalla de todos los archivos de Blender del modelo

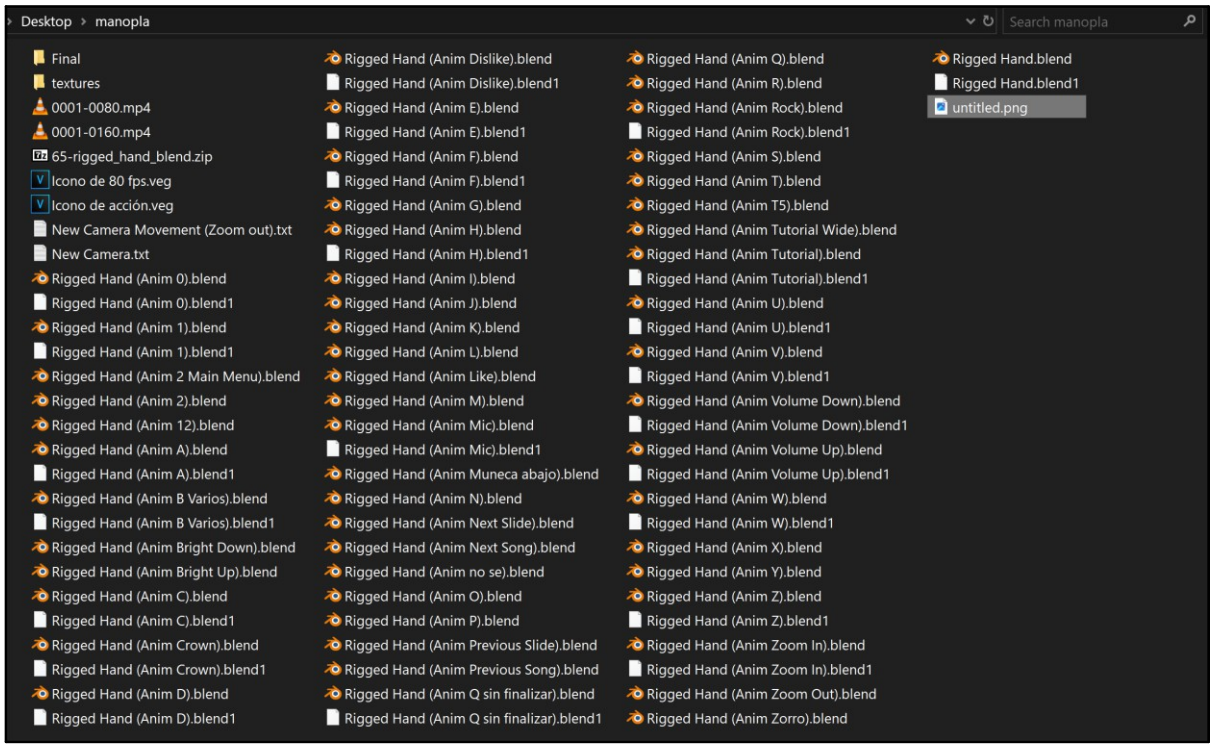


Figura 105. Captura de pantalla de todos los archivos MP4 del modelo

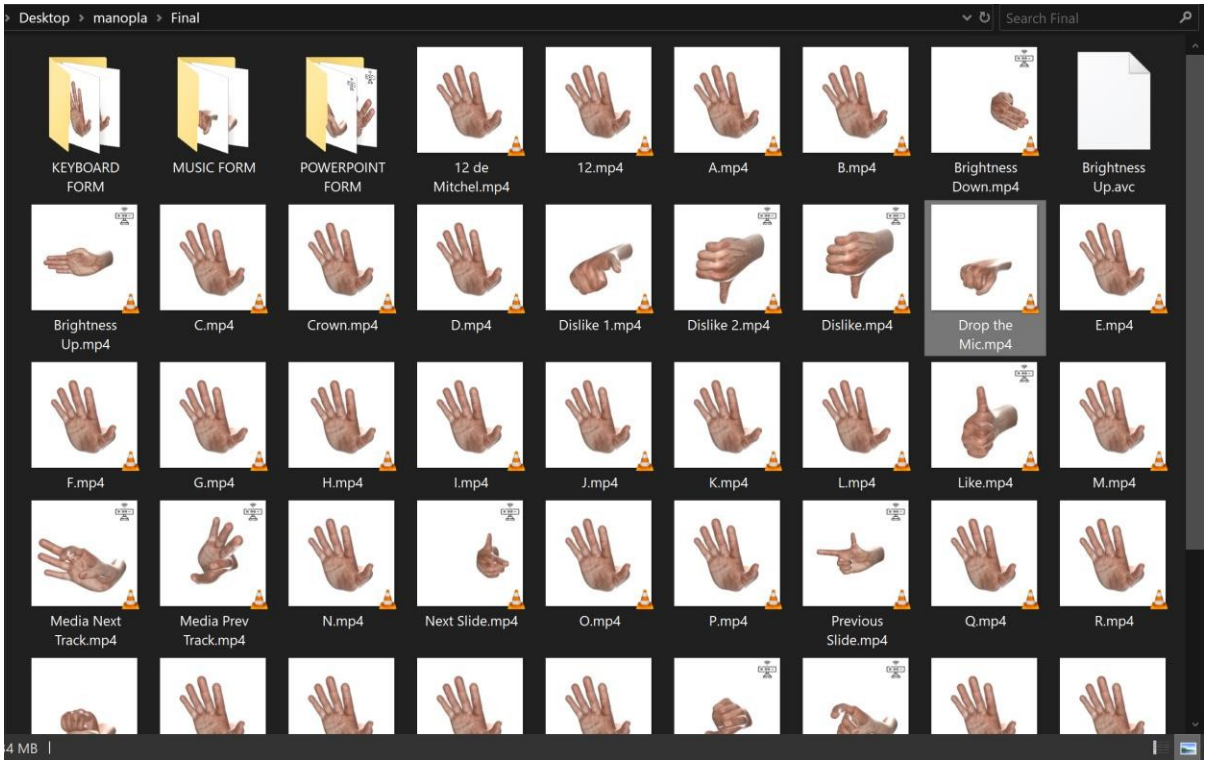


Figura 106. Captura de pantalla del modelo con los huesos colocados por nosotros

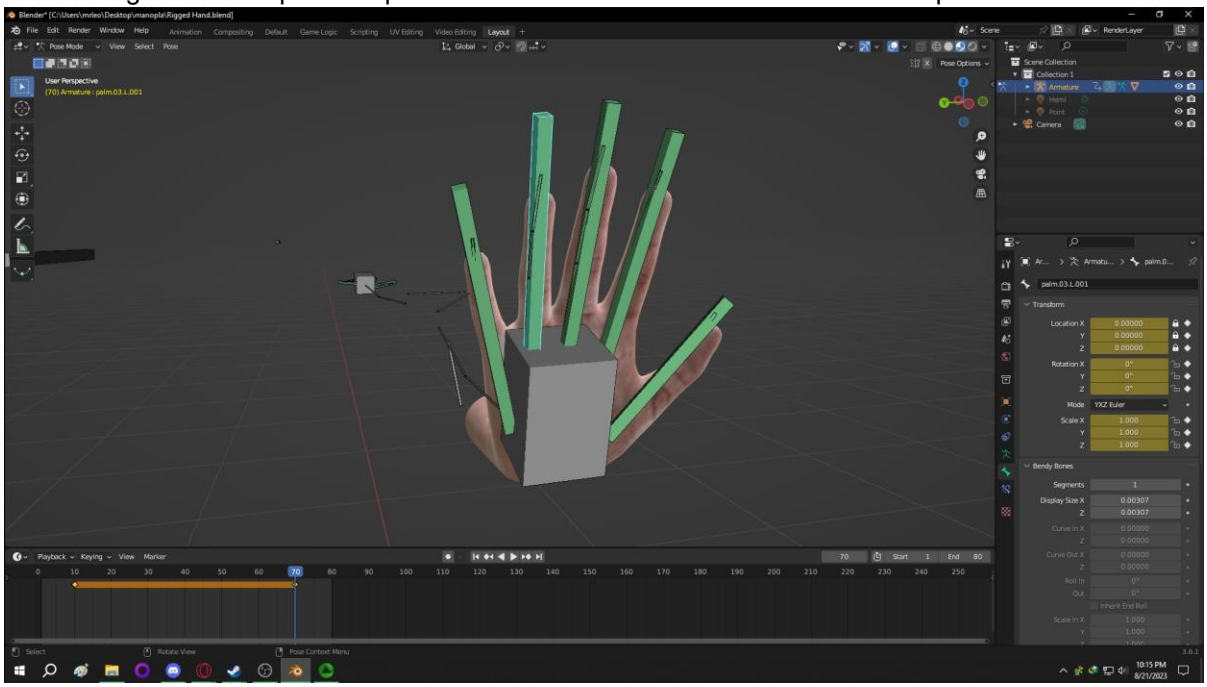
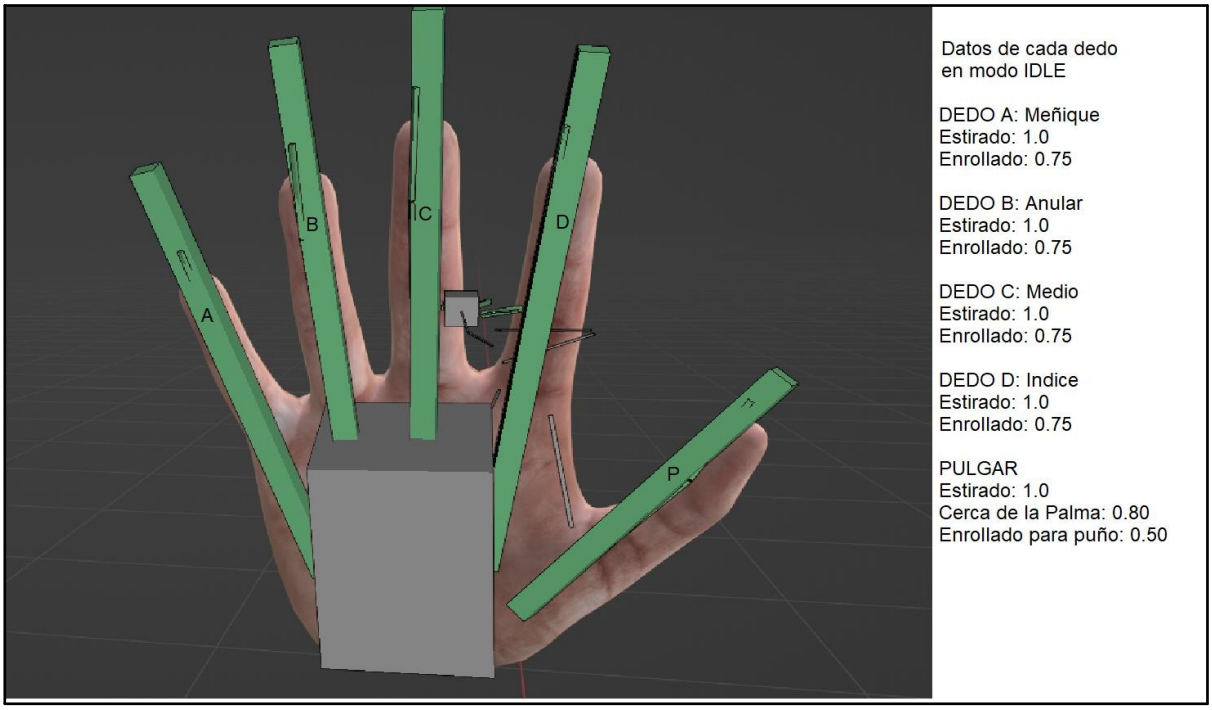


Figura 107. Captura de pantalla del modelo con las medidas de los huesos



Los gestos fueron creados utilizando una gran cantidad de inspiración de gestos utilizados en la cultura general. Además, se incluyeron gestos utilizados en diferentes lenguajes de señas de muchos países, conocimientos de computación básica para utilizar los gestos en binario, y varios otros gestos que se utilizan en ciertas regiones del mundo.

Cada gesto fue luego renderizado en un video MP4, y luego convertido al formato GIF para ser utilizado dentro de la aplicación. Todos los GIF tienen una duración de 3 segundos los gestos sencillos y de 6 segundos los gestos con dos posiciones. A todos los gestos se les agregó un ícono de un Kinect en la parte superior derecha para indicar en qué posición empieza a registrar el gesto correcto la cámara de profundidad.

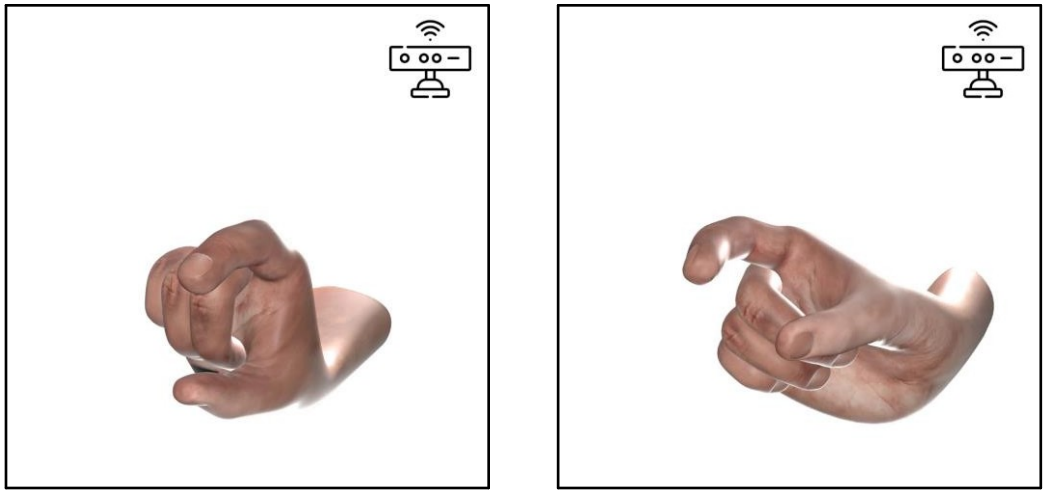
A continuación, mostraremos una lista completa de cada gesto utilizado, compilados por módulo, junto con una descripción del origen de dicho gesto. En caso de que el gesto sea sencillo se incluye una sola figura. Si el gesto es compuesto y tiene 2 posiciones para hacer el comando, se incluyen dos imágenes.

3.7.2. Music Control

3.7.2.1. Volume Up

Usa el gesto de subir volumen con una perilla de radio, similar que Volume Down.

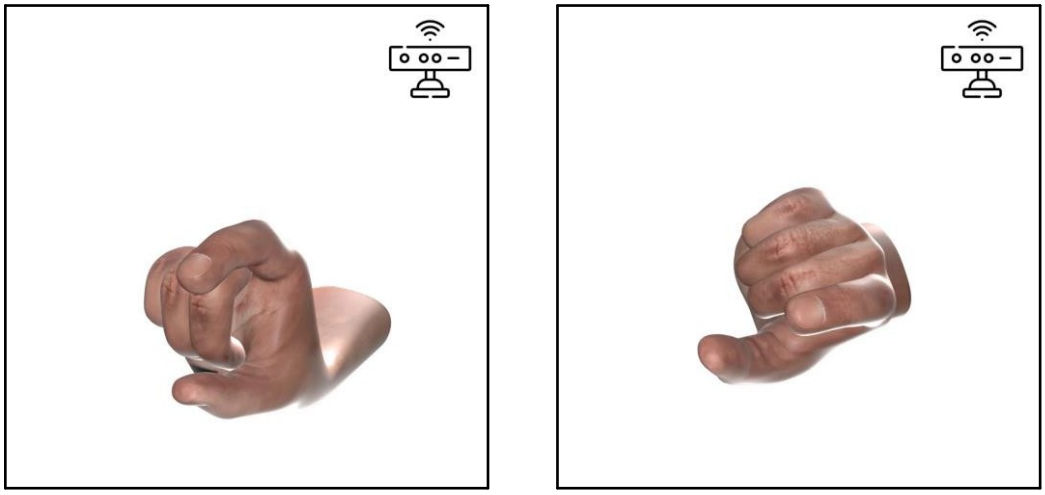
Figura 108. Gesto Volume Up



3.7.2.2. Volume Down

Usa el gesto de bajar volumen con una perilla de radio, similar que Volume Up.

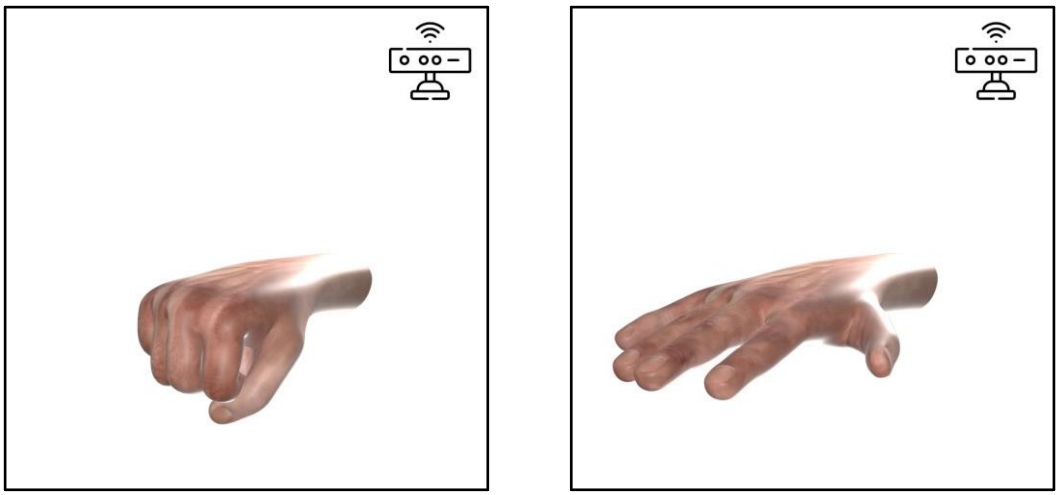
Figura 109. Gesto Volume Down



3.7.2.3. Volume Mute

También llamada como «Drop the Mic» en el código por la semejanza en dejar caer un micrófono de aire para detener la música.

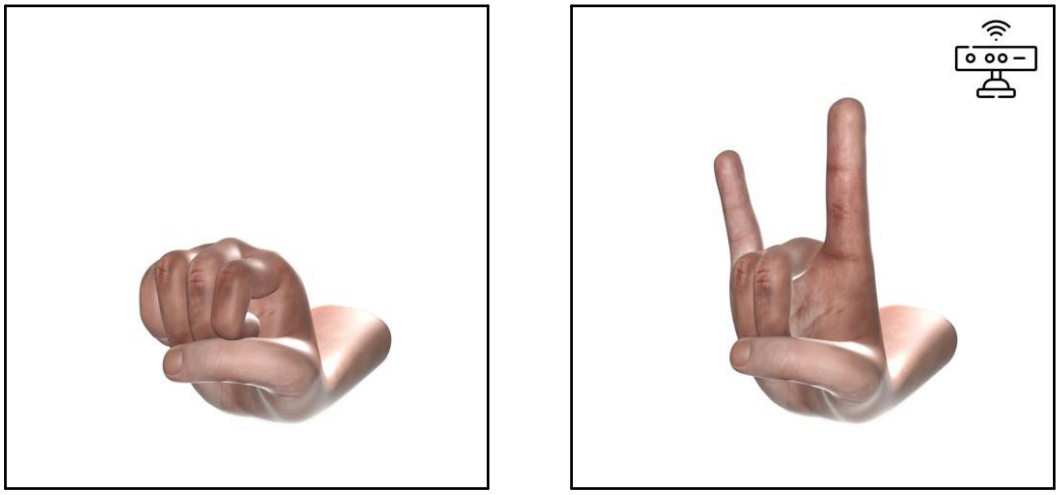
Figura 110. Gesto Volume Mute



3.7.2.4. Media Play/Pause

También catalogado como «Rock On». Así aparece en el código original de Proyecto Praga y no hacía falta alterarlo.

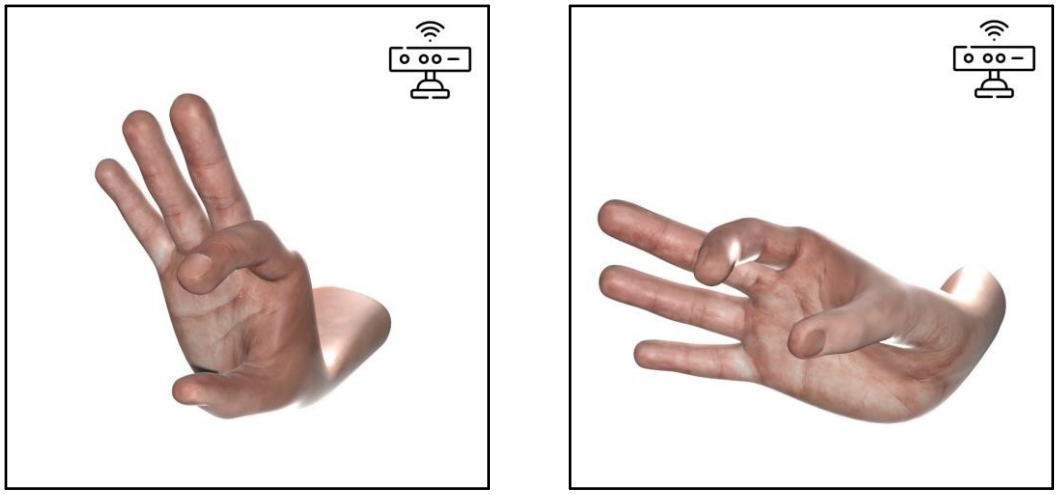
Figura 111. Gesto Media Play/Pause



3.7.2.5. Media Next

Simula mover una perilla analógica con el añadido de hacer una pinza con el índice y el pulgar. Similar al Volumen.

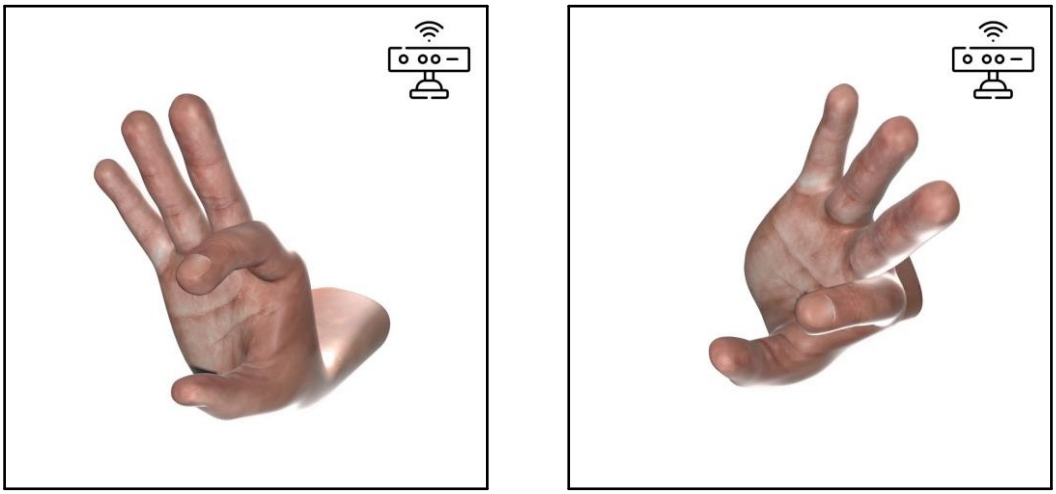
Figura 112. Gesto Media Next



3.7.2.6. Media Previous

Simula mover una perilla analógica con el añadido de hacer una pinza con el índice y el pulgar. Similar al Volumen.

Figura 113. Gesto Media Previous

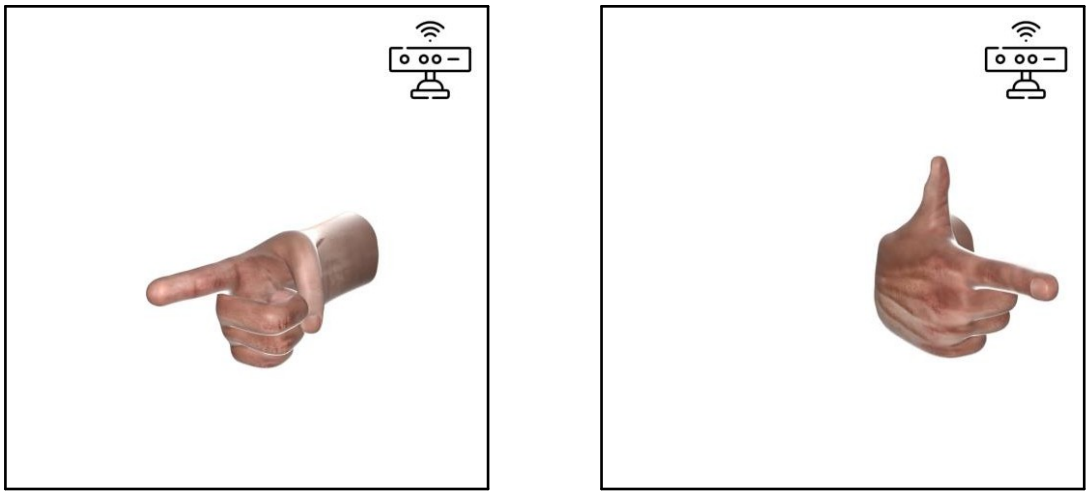


3.7.3. Slideshow Control

3.7.3.1. Next Slide

Simula el gesto de pasar página de un libro hacia adelante usando un solo dedo.

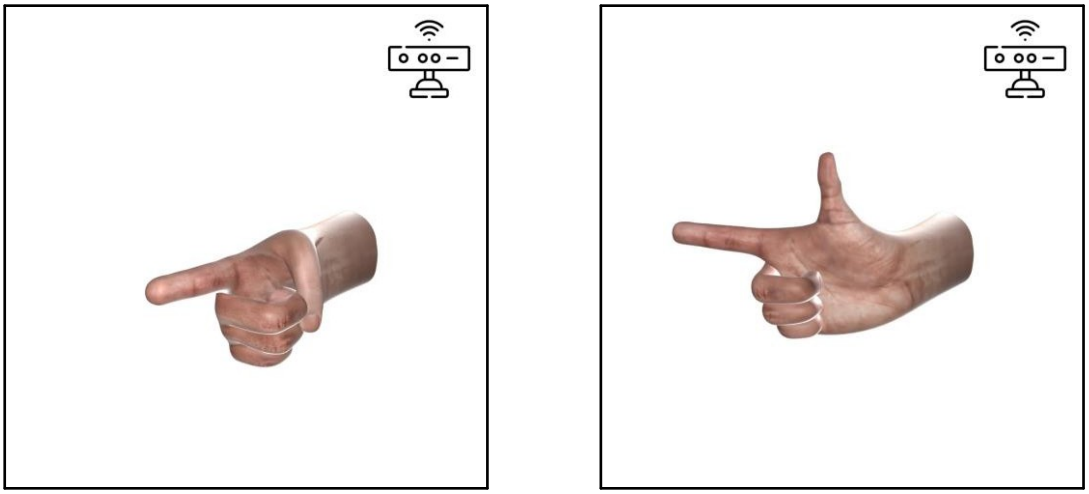
Figura 114. Gesto Next Slide



3.7.3.2. Previous Slide

Simula el gesto de pasar página de un libro hacia atrás usando un solo dedo.

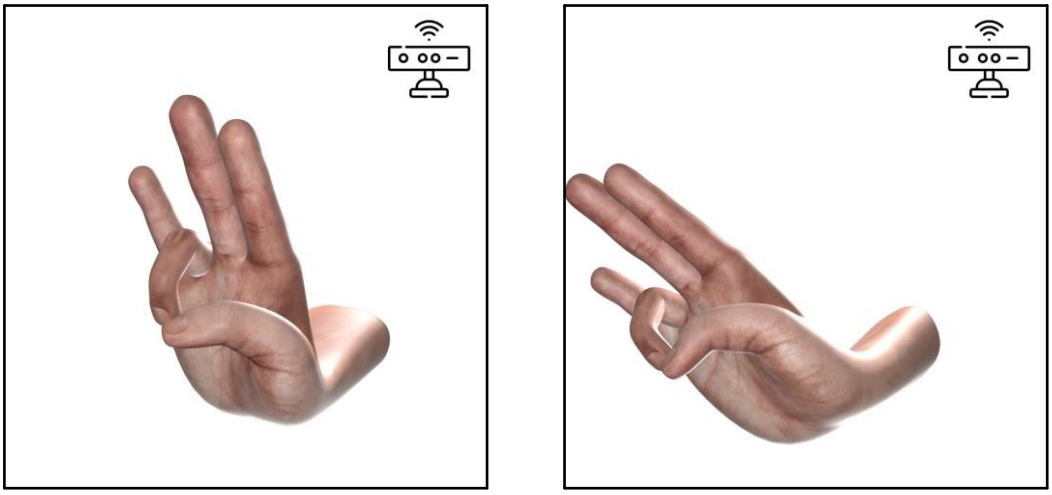
Figura 115. Gesto Previous Slide



3.7.3.3. Zoom In

Usa el gesto de “22” en binario, sin ninguna referencia específica. Hacia la derecha para aclarar que aumenta.

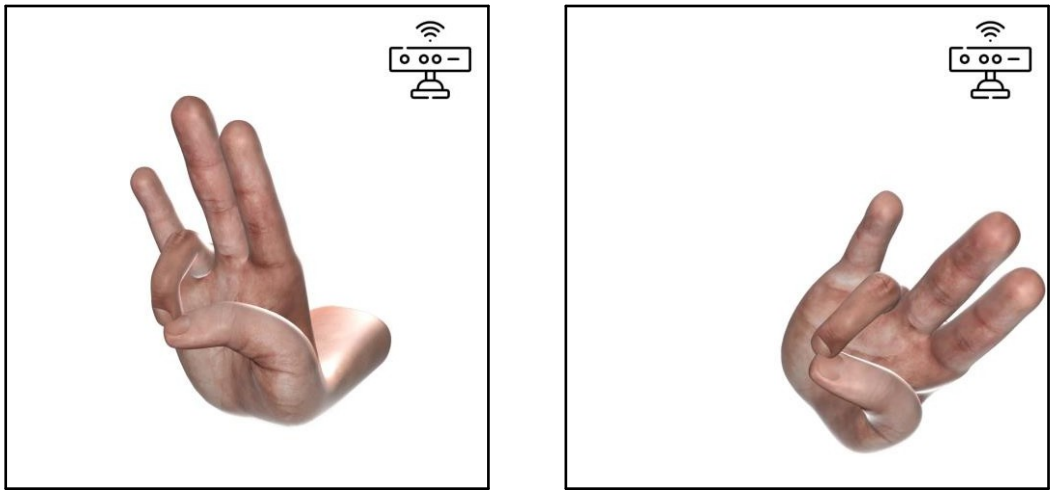
Figura 116. Gesto Zoom In



3.7.3.4. Zoom Out

Usa el gesto de “22” en binario, sin ninguna referencia específica. Hacia la derecha para aclarar que disminuye.

Figura 117. Gesto Zoom Out



3.7.3.3. Zoom to Fit

Gesto del “zorro” japonés (狐). Se utilizó para simular una pinza que ajusta la pantalla.

Figura 118. Gesto Zoom to Fit

3.7.4. Alphabet Control

Nota: Se han utilizado gestos de diferentes países y regiones que son utilizados en lenguajes de seña alrededor del mundo.

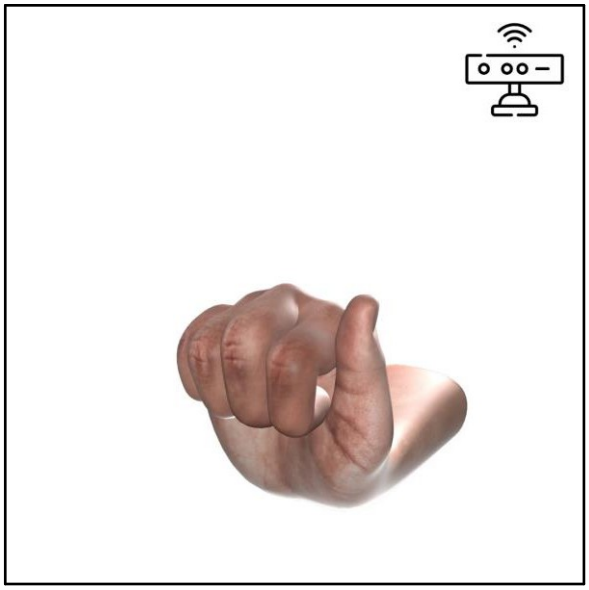
Puede resultar un poco confuso, principalmente para los usuarios que conocen dichos lenguajes de seña, pero se ha hecho de esta manera para que el Kinect no tuviera problemas en identificar algunos gestos (algunos ocultan por completo algunos dedos, u otros gestos utilizan movimientos que serían muy complicados para el usuario)

101

3.7.4.1. A

Utiliza la “A” del lenguaje de señas de Estados Unidos.

Figura 119. Gesto A



3.7.4.2. B

Utiliza la “B” del lenguaje de señas de Estados Unidos.

Figura 120. Gesto B

102

3.7.4.3. C

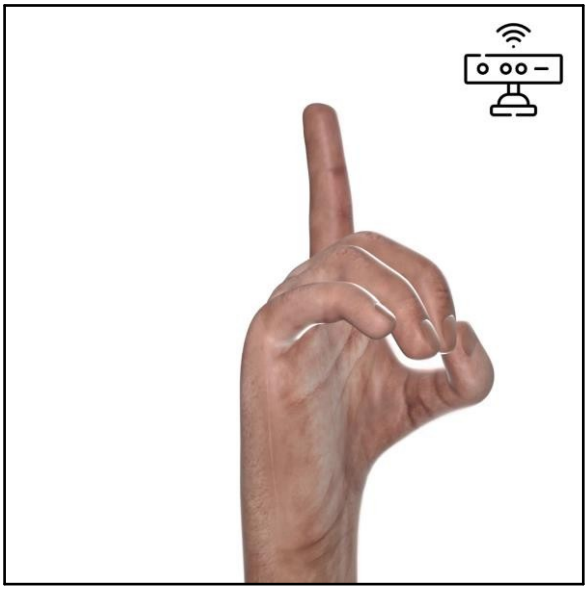
Utiliza la “C” del lenguaje de señas de Estados Unidos.

Figura 121. Gesto C

3.7.4.4. D

Utiliza la “D” del lenguaje de señas de Estados Unidos.

Figura 122. Gesto D



103

3.7.4.5. E

Utiliza la “E” del lenguaje de señas de Estados Unidos.

Figura 123. Gesto E

3.7.4.6. F

Utiliza la “F” del lenguaje de señas de Estados Unidos.

Figura 124. Gesto F

104

3.7.4.7. G

Utiliza la “G” del lenguaje de señas de Nicaragua.

125. Gesto G

3.7.4.8. H

Utiliza la “H” del lenguaje de señas de Estados Unidos.

Figura 126. Gesto H

105

3.7.4.9. I

Utiliza la “I” del lenguaje de señas de Estados Unidos.

127.Gesto I

3.7.4.10. J

Utiliza la “J” del lenguaje de señas de Estados Unidos.

Figura 128. Gesto J

106

3.7.4.11. K

Gesto inventado. Debido a fallas en el reconocimiento se decidió crear este gesto para la K. No pertenece a ningún lenguaje de señas.

129. Gesto K

3.7.4.12. L

Utiliza la “L” del lenguaje de señas de Estados Unidos.

Figura 130. Gesto L

107

3.7.4.13. M

Utiliza la “M” del lenguaje de señas de Nicaragua.

131. Gesto M

3.7.4.14. N

Utiliza la “N” del lenguaje de señas de Nicaragua.

Figura 132. Gesto N

108

3.7.4.15. O

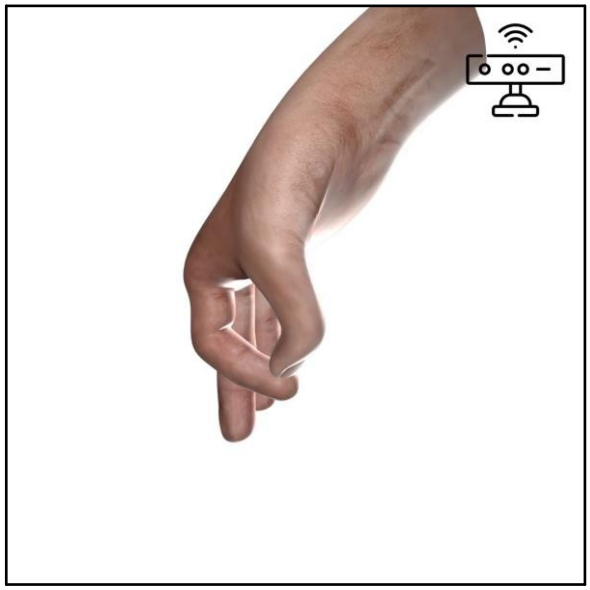
Utiliza la “O” del lenguaje de señas de Estados Unidos.

Figura 133. Gesto O

3.7.4.16. P

Utiliza la “P” del lenguaje de señas de Portugal.

Figura 134. Gesto P



109

3.7.4.17. Q

Utiliza la “Q” del lenguaje de señas de Francia.

135. Gesto Q

3.7.4.18. R

Utiliza la “R” del lenguaje de señas de Estados Unidos.

Figura 136. Gesto R

110

3.7.4.19. S

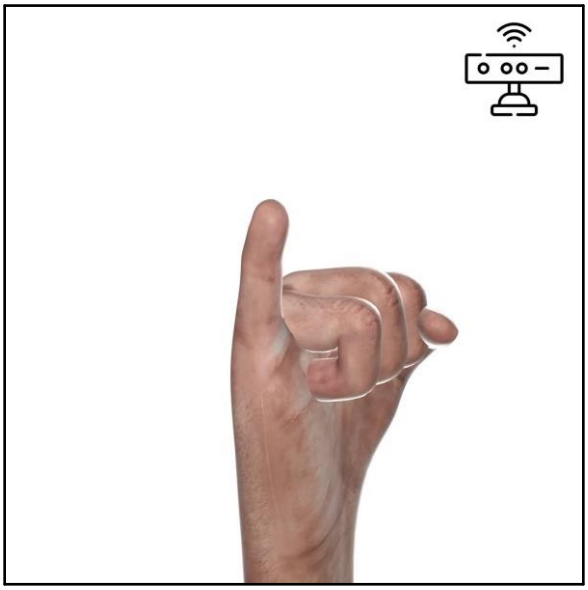
Gesto inventado. Debido a fallas en el reconocimiento se decidió crear este gesto para la S. No pertenece a ningún lenguaje de señas.

Figura 137. Gesto S

3.7.4.20. T

Gesto inventado. Debido a fallas en el reconocimiento se decidió crear este gesto para la T. No pertenece a ningún lenguaje de señas.

Figura 138. Gesto T



111

3.7.4.21. U

Utiliza la “U” del lenguaje de señas de Estados Unidos.

Figura 139. Gesto U

3.7.4.22. V

Utiliza la “V” del lenguaje de señas de Estados Unidos.

Figura 140. Gesto V

112

3.7.4.23. W

Utiliza la “W” del lenguaje de señas de Estados Unidos.

Figura 141. Gesto W

3.7.4.24. X

Utiliza la “X” del lenguaje de señas de Estados Unidos.

Figura 142. Gesto X

113

3.7.4.25. Y

Utiliza la “Y” del lenguaje de señas de Estados Unidos.

Figura 143. Gesto Y

3.7.4.26. Z

Gesto inventado. Debido a fallas en el reconocimiento se decidió crear este gesto para la Z. No pertenece a ningún lenguaje de señas.

Figura 144. Gesto Z

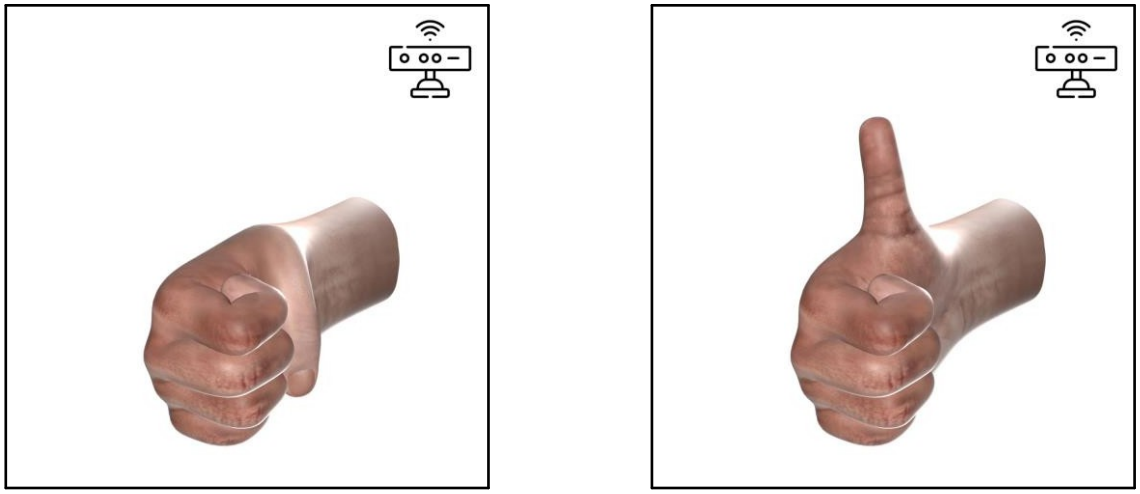
114

3.7.5. Lightbulb Control

3.7.5.1. Encender/Apagar

Pulgar Arriba. Gesto Universal ya incluido en el Gesture Sample.

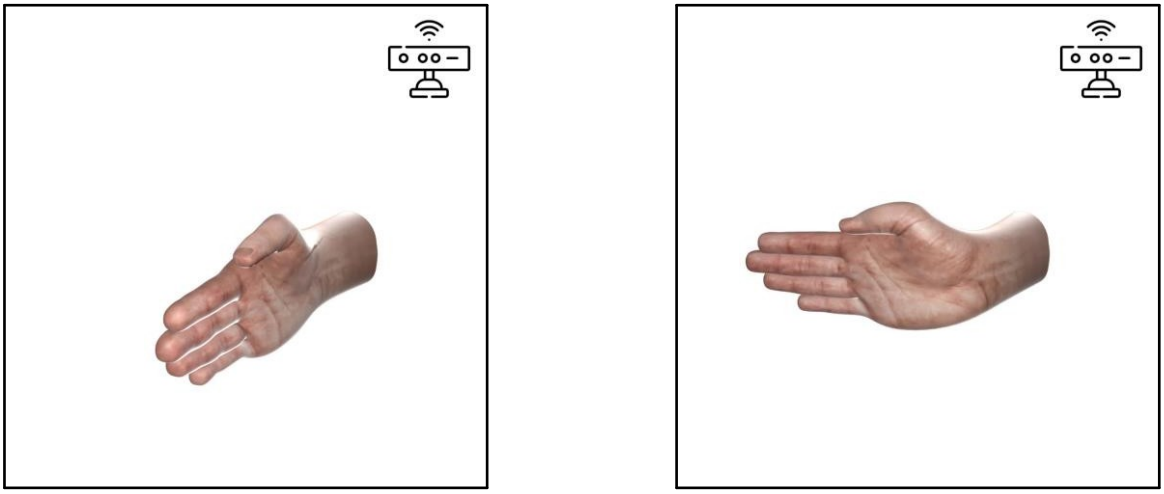
Figura 145. Gesto Encender/Apagar



3.7.5.2. Subir Brillo

Simula mover una perilla analógica con el añadido de hacer una pinza con el índice y el pulgar. Similar al Volumen.

Figura 146. Gesto Subir Brillo

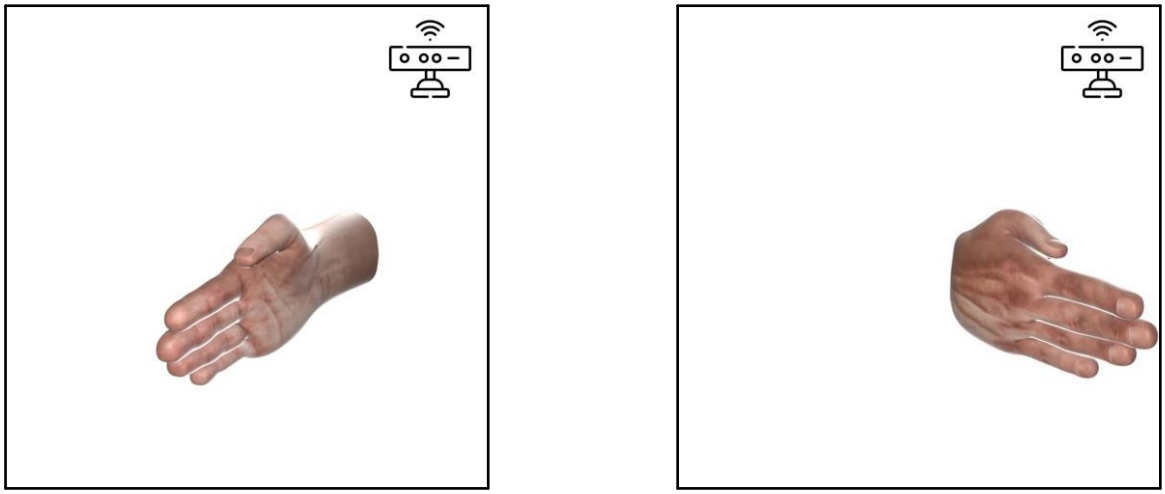


115

3.7.5.3. Bajar Brillo

Simula mover una perilla analógica con el añadido de hacer una pinza con el índice y el pulgar. Similar al Volumen.

Figura 147. Gesto Bajar Brillo



3.7.5.4. Rotar Colores

Doce Binario. No tiene ninguna otra referencia y sólo sirve para rotar en una dirección.

Figura 148. Gesto Rotar Colores

116

3.7.5.5. Reiniciar Colores

Gesto llamado Crown. Unicamente utilizado para reiniciar la ruleta de colores.

Figura 149. Gesto Reiniciar Colores

3.7.5.6. Guardar Cambios

Pulgar hacia abajo. Un poco contradictorio en función, pero hace el recordarlo mucho más sencillo

Figura 150. Gesto Guardar Cambios

117

VI. CONCLUSIONES

El desarrollo de la aplicación de escritorio para controlar funciones mediante gestos con el Kinect ha sido exitoso y cumple con los objetivos planteados inicialmente. La aplicación permite reproducir música y videos, así como controlar dispositivos de domótica inteligente en el hogar, y mostrar una forma de escritura no tan practicada por las personas, todo ello mediante gestos realizados con la mano derecha. Esta funcionalidad ha sido lograda gracias a la integración efectiva del Kinect, que ha demostrado ser una herramienta versátil y eficiente para capturar y reconocer gestos de forma precisa.

Además, la aplicación se ha desarrollado siguiendo la Metodología de Desarrollo Rápido de Aplicaciones, lo que ha permitido implementar funcionalidades de manera incremental, facilitando la detección temprana de errores y la incorporación de mejoras de forma ágil. Esta metodología ha demostrado ser efectiva para garantizar la calidad y eficiencia del desarrollo del proyecto.

Las posibilidades de esta aplicación son prometedoras y abren la puerta a futuros desarrollos en el campo de la interacción humano-computadora. Se podrían explorar nuevas funcionalidades y aplicaciones para esta tecnología, como la integración con sistemas de inteligencia artificial para una mayor automatización en el hogar o la adaptación de la aplicación para su uso en otros contextos, como la salud o la educación. En resumen, este proyecto sienta las bases para futuras investigaciones y desarrollos en el campo de la tecnología de interacción gestual.

Además, el desarrollo de esta aplicación no solo ha sido un logro técnico, sino también un paso significativo en la mejora de la calidad de vida de las personas. Al ofrecer una forma más intuitiva y natural de interactuar con la tecnología, especialmente para aquellos con discapacidades o limitaciones físicas, se ha promovido la inclusión y accesibilidad en el ámbito tecnológico. Esta aplicación tiene el potencial de impactar positivamente en la vida diaria de las personas, facilitando tareas cotidianas y mejorando su independencia.

Por otro lado, el éxito de este proyecto también ha demostrado la importancia de la investigación y la innovación en el campo de la tecnología. El uso creativo de herramientas como el Kinect y la Metodología de Desarrollo Rápido de Aplicaciones ha permitido desarrollar una solución efectiva y eficiente para un problema específico. Este enfoque innovador puede servir de inspiración para futuros proyectos en áreas similares, fomentando así el avance continuo en el campo de la interacción humano-computadora. Asimismo, el trabajo realizado en esta tesis deja un legado importante para la comunidad académica y profesional, brindando un marco sólido para futuras investigaciones y desarrollos en el campo de la tecnología de interacción gestual y la domótica inteligente.

118

VII. RECOMENDACIONES

Los gestos creados para este proyecto fueron creados considerando su factibilidad humana, asegurando que fueran ejecutables por cualquier adulto con la mano derecha. Con el tiempo, descubrimos numerosos atajos y variantes de los gestos, lo que amplió significativamente las posibilidades. Aunque no pudimos explorar todas las potenciales aplicaciones debido a las limitaciones temporales, al menos incluimos un gesto para cada función preexistente, abarcando diversas orientaciones y movimientos simples de la mano.

La documentación original que proporcionamos incluye información detallada sobre cómo crear nuevos gestos. Esto no solo sirve como guía para futuros desarrolladores, sino que también fomenta la experimentación con la tecnología. Invitamos a la imaginación y la exploración de nuevas formas de interacción gestual, dando a aquellos que trabajen con nuestro código la flexibilidad para expandir y mejorar el proyecto.

En relación con la amplitud del proyecto, creemos que tiene un potencial sustancial para mejoras. Inicialmente, constaba de ocho módulos, pero solo concebimos cuatro, dejando deliberadamente otros sin explorar ni explicar en la investigación. Esta decisión se tomó para permitir que el próximo desarrollador despliegue su creatividad y explore libremente cómo utilizar estos controles.

Con la aplicación desarrollada en un plazo de tiempo gestionado eficientemente utilizando C# en Visual Studio, cualquiera con acceso a nuestro código puede crear gestos personalizados y experimentar con este tipo de control único, que rara vez se encuentra en el panorama tecnológico actual.

Proponemos como sugerencia de este proyecto el de continuar desarrollando módulos para el mismo proyecto, pero siempre tratando de enfocar una temática a la vez. Tomamos en cuenta la referencia de que haber realizado 4 módulos para el mismo, fue mejor al momento de presentar la aplicación, pero notamos que se podría haber desarrollado mucho mejor enfocando nuestro esfuerzo en un sólo módulo.

Entre algunos módulos que no logramos desarrollar y pueden seguir esta línea está la aplicación de aprendizaje del lenguaje de señas (enfocado a un país o región específico), juegos varios que utilicen el Kinect, comandos o atajos del computador, entre muchas otras ideas.

VIII) BIBLIOGRAFÍA

[1] S. Benford; H. Schnadelbach; B. Koleva; B. Gaver; A. Schmidt; A. Boucher; A. Steed; R. Anastasi; C. Greenhalgh; T. Rodden; H. Gellersen (2003). "Sensible, sensable and desirable: a framework for designing physical interfaces" (PDF). Archivado para la posteridad y actualmente disponible en https://web.archive.org/web/20060126085052/http://www.equator.ac.uk/var/uploads/benfordTech2003.pdf

[2] M. Bastian 2021, "Kinect y cámaras de profundidad y tracking compatibles con Isadora 3". Disponible en https://hybridart.net/kinect-camaras-de-profundidad-y-tracking-compatibles-con-isadora-3/

[3] T. Moes 2014, "¿Qué es Windows? Definición e historia". Disponible en https://softwarelab.org/es/windows-historia/

[4] Communications BBVA, Autores invitados, "Domótica e inmótica, ¿son las mismas tecnologías inteligentes?", Act. 03 sep 2021 [En línea]. Disponible en https://www.bbva.com/es/sostenibilidad/domotica-e-inmotica-son-las-mismas-tecnologias-inteligentes/

[5] Dietrich Kammer, Mandy Keck, Georg Freitag, Markus Wacker, "Taxonomy and Overview of Multi-touch Frameworks: Architecture, Scope and Features" Archivado para la posteridad y actualmente disponible en: https://web.archive.org/web/20110125014444/http://vi-c.de/vic/sites/default/files/Taxonomy_and_Overview_of_Multi-touch_Frameworks_Revised.pdf

[6] R. Ibáñez 2019. "Contando con los dedos de la mano, al estilo binario". Disponible en https://aprenderapensar.net/2019/03/13/contando-con-los-dedos-de-la-mano-al-estilo-binario/

[7] R. Gacitúa Bustos, "Métodos de desarrollo de software: El desafío pendiente de la estandarización," Theoria, vol. 12, no. 1, pp. 23-42, 2003. [En línea]. Disponible en https://www.redalyc.org/pdf/299/29901203.pdf

[8] M. Castro 2019, Metodología RAD o DRA. El Desarrollo Rápido de Aplicaciones (Incentro) Disponible en: https://www.incentro.com/es-ES/blog/metodologia-rad-desarrollo-rapido-aplicaciones

[9] Microsoft 2014, "Project Prague – Hand Gestures SDK" Disponible en: https://www.microsoft.com/en-us/research/project/project-prague/

[10] L . Rosencrance 2019, "software development kit (SDK), definition". Disponible en https://www.techtarget.com/whatis/definition/software-developers-kit-SDK

ANEXO 1

DOCUMENTACIóN DE PROYECTO ORIGINAL (PROJECT PRAGUE)

OVERVIEW

Artículo escrito el día 17 de agosto de 2022. Contribuyen 6 personas: YoniSmolin, DhurataJ, v-pegao, mairaw, prnadago, sankethka

¿Qué es Proyecto Prague?

Project Prague es un SDK (kit de desarrollo de software) que le permite crear experiencias NUI (interfaz de usuario natural) basadas en la entrada de gestos con las manos. Proporcionamos API (interfaces de programación de aplicaciones) para C#, C++ (incluidas las variantes UWP y .NET Core), lo que le permite diseñar e implementar fácilmente sus propios gestos manuales personalizados e integrarlos en sus aplicaciones.

Los componentes básicos de un gesto son las poses y los movimientos de las manos. Usando restricciones simples especificadas en un lenguaje intuitivo, le permitimos definir cualquier pose de mano y cualquier movimiento de mano que desee. Puede encadenar una secuencia de posturas y movimientos de manos para especificar un gesto. Una vez definido y registrado tu gesto con nuestro runtime, te avisaremos cada vez que detectemos que tu usuario ha realizado el gesto con la mano. En este punto puede ejecutar la lógica deseada para responder al gesto detectado.

Con Project Prague, puede proporcionar a sus usuarios gestos con las manos para controlar de forma intuitiva la reproducción de música y video, marcar y dar me gusta al contenido web, enviar un emoji en aplicaciones de IM (mensajería instantánea), interactuar con un asistente digital, crear y ejecutar presentaciones de diapositivas de PowerPoint, manipular objetos tridimensionales, jugar juegos usando solo sus manos y mucho más.

Primeros pasos con Project Prague

Cámaras de profundidad soportadas

Se necesita tener una cámara de profundidad para ejecutar Project Prague en su máquina. Actualmente admitimos las siguientes marcas de cámaras:

Tabla 2.
Tipos de cámara de profundidad, Project Prague

Marca	Rango de detección	Desempeño
Intel® RealSense™ SR300 camera	20-60 [cm]	La mejor
Intel® RealSense™ F200 camera	20-60 [cm]	Bueno
Kinect for Windows v2	60-110 [cm]	Bueno

Fuente: Project Prague, 2017, https://learn.microsoft.com/en-us/gestures/

Requisitos de hardware y software

Asegúrese de que su sistema cumpla con los siguientes requisitos antes de proceder a configurar Project Prague:

Tabla 3. Requerimientos mínimos para la aplicación, Project Prague

Categoría	Recomendado	Mínimo
CPU	Intel® Core™ i7 series, 8 logical cores	Intel® Core™ i5 series, 4 logical cores
RAM	2GB o más	1GB
SISTEMA OPERATIVO	Windows 10 con la Creator Update Instalada	Windows 10

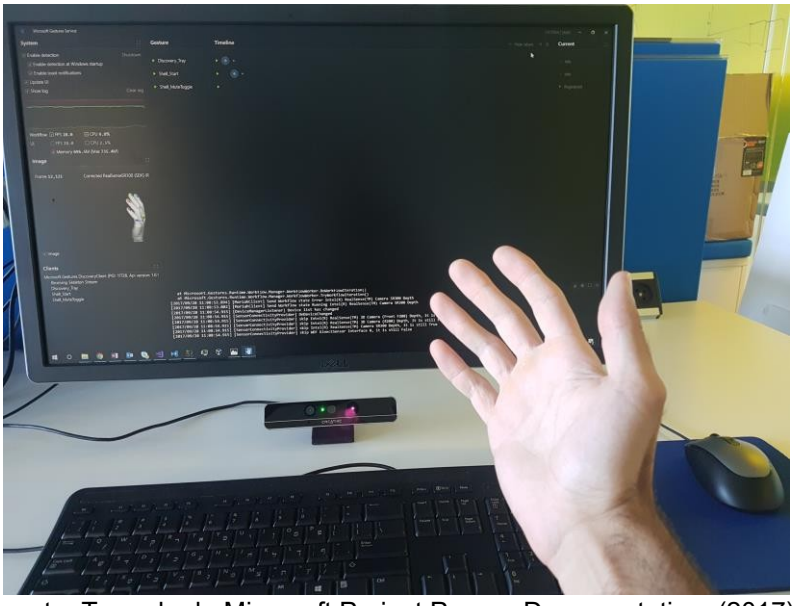
Fuente: Project Prague, 2017, https://learn.microsoft.com/en-us/gestures/

Configurando Project Prague en su máquina

Para que Project Prague se ejecute en su máquina, deberá:

1. Conecte su cámara de profundidad a un puerto USB 3.0 y colóquela debajo del monitor de su computadora, como se ilustra en la imagen a continuación:

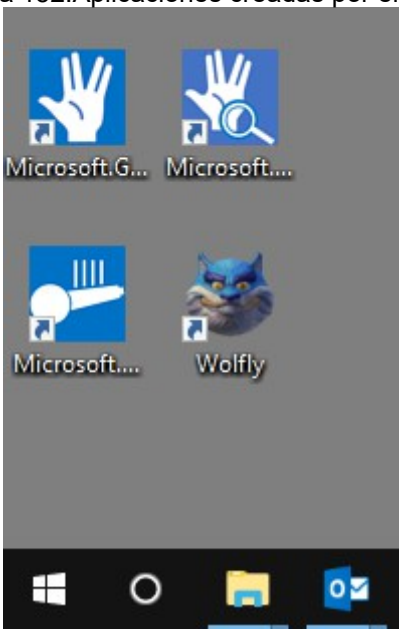
Figura 151. Gesture Services



Fuente: Tomado de Microsoft Project Prague Documentation. (2017)

2. Descargue e instale Project Prague runtime de https://aka.ms/gestures/download. La instalación colocará accesos directos en su escritorio que apuntan a nuestras aplicaciones de demostración compiladas:

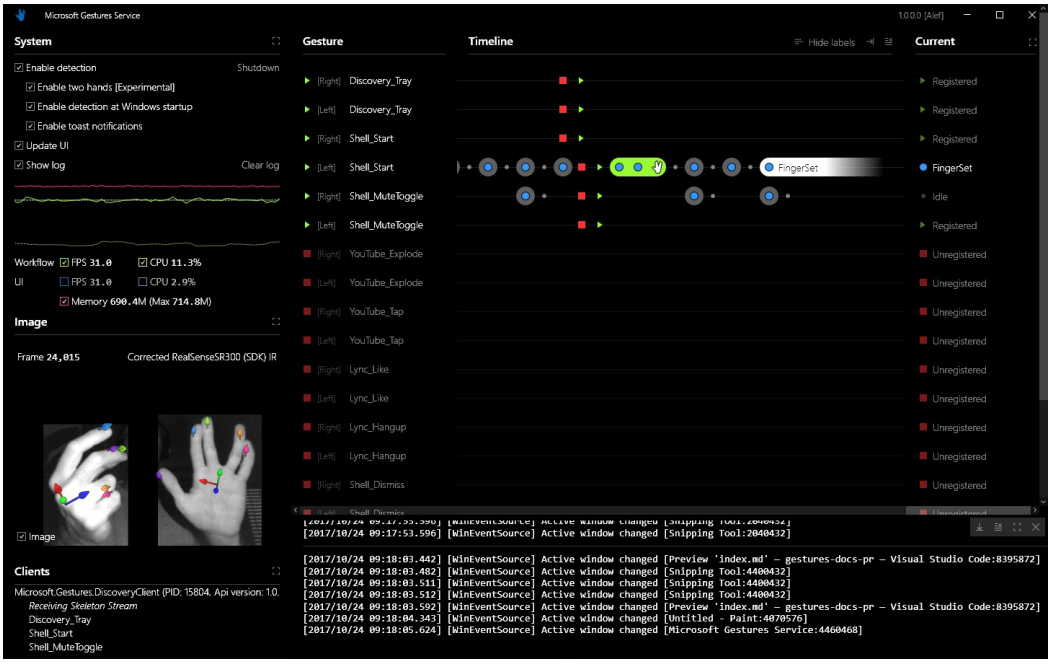
Figura 152.Aplicaciones creadas por el equipo



Fuente: Tomado de Microsoft Project Prague Documentation. (2017)

3. Cuando se complete la instalación, aparecerá una ventana titulada Microsoft Gestures Service se pondrá en marcha. Este es nuestro panel de detección de poses y gestos en tiempo real. Asegúrese de que sus dedos se detecten con éxito, como se muestra a continuación:

Figura 153.Vistazo amplio a Gesture.Services



Fuente: Tomado de Microsoft Project Prague Documentation. (2017)

Tenga en cuenta que el programa de instalación de Project Prague ha configurado las siguientes dos aplicaciones para que se inicien al iniciar Windows y se ejecuten en segundo plano:

- Microsoft.Gestures.Sync - mantiene el Gestures service en funcionamiento y extrae actualizaciones cuando las publicamos
- Microsoft.Gestures.DiscoveryClient - proporciona integración de gestos para varios contextos: shell de Windows, PowerPoint, Skype, YouTube, Fotos y Visual Studio.

Puede modificar esta configuración en cualquier momento en la pestaña Inicio del Administrador de tareas.

Gestures Service

Artículo escrito el día 19 de diciembre de 2022. Contribuyen 3 personas: YoniSmolin, alexbuckgit
, v-pegao

Gestures Service proporciona detección de gestos como un servicio local para aplicaciones cliente.

Todas las aplicaciones se basan en Gestures Service para la detección de gestos. Puede utilizar la clase GesturesServiceEndpoint para comunicarse con Gestures Service desde su aplicación.

Después de instalar Project Prague, Gestures Service se iniciará cada vez que se inicie su máquina. Para deshabilitar esto, desmarque la casilla Habilitar detección al inicio de Windows en la sección Sistema de la ventana IU (interfaz de usuario) del Servicio de gestos.

Gestures Service UI

El Gestures Service tiene una interfaz de usuario que muestra información sobre la detección de gestos y poses en tiempo real.

Ejecución y terminación del Servicio Gestos

Para iniciar **Gestures Service** , haga doble clic en el acceso directo  **de Microsoft.Gestures.Service** en su escritorio.

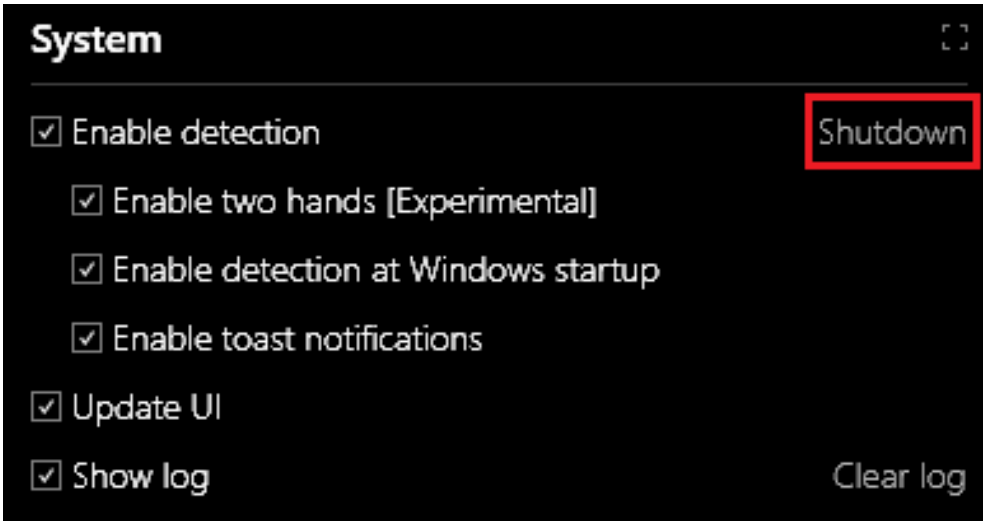
Para ocultar la **interfaz de usuario del servicio de gestos,** simplemente cierre su ventana con el botón **X** en la esquina superior derecha.

Nota

Cerrar la interfaz de usuario no finalizará el proceso **del Servicio de gestos** , es decir, la detección de gestos seguirá estando disponible. Para volver a mostrar la **interfaz de usuario de Gestures Service** , después de ocultarla, haga doble clic en el acceso directo **de Microsoft.Gestures.Service** en su escritorio.

Para finalizar el proceso del Servicio de gestos, haga clic en el enlace **Shutdown en la sección Sistema** de la interfaz de usuario del **Servicio de gestos** :

Figura 154.Cerrando el Servicio de Gestos



Fuente: Tomado de Microsoft Project Prague Documentation. (2017)

Funciones de la interfaz de usuario del servicio de gestos

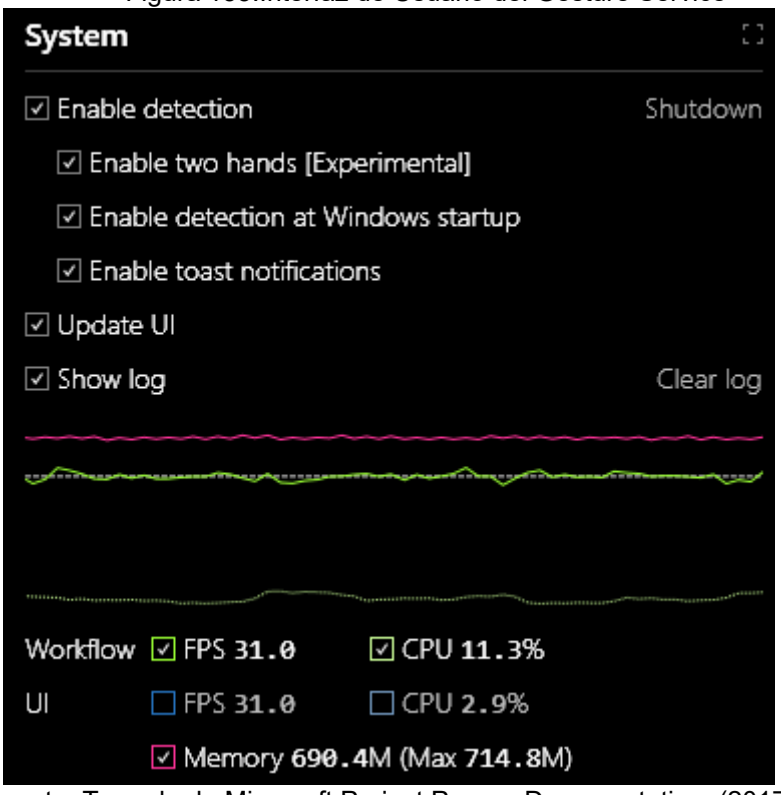
Ahora cubriremos en detalle las distintas secciones de la interfaz de usuario del Servicio Gestures.

Tenga en cuenta que cada sección tiene un **botón de maximización** en la esquina superior derecha, lo que le permite extender la sección para ocupar toda la extensión de la ventana de la interfaz de usuario del Servicio de gestos:

Sección del sistema

La sección **Sistema** es el centro de comando y control de la **interfaz de usuario de Gestures Service** :

Figura 155.Interfaz de Usuario del Gesture Service



Fuente: Tomado de Microsoft Project Prague Documentation. (2017)

Esta sección se divide en dos partes: la parte superior contiene casillas de verificación para controlar diferentes funciones del **Servicio Gestures** y la parte inferior ilustra métricas de rendimiento relevantes a medida que evolucionan con el tiempo.

Las casillas de verificación y su funcionalidad correspondiente se enumeran en la siguiente tabla:

Tabla 4. Procesos realizados por Gesture.Sample

Caja	Función
Permitir la detección	Detener o reanudar el **servicio Gestures**
Habilitar dos manos [Experimental]	Habilite la detección de la mano izquierda además de la detección de la mano derecha (que siempre está activada). Habilitar la detección de mano izquierda genera un pequeño aumento en el consumo de CPU.
Habilitar la detección al inicio de Windows	Especifique si **Gestures Service** se iniciará la próxima vez que se inicie Windows
Habilitar notificaciones	Especificar si el **Servicio de gestos** mostrará notificaciones
Actualizar interfaz de usuario	Detener o reanudar la **interfaz de usuario**

	del servicio Gestures
Mostrar registro	Alternar la visibilidad de la sección **Registro** en la **interfaz de usuario del servicio Gestures**

Fuente: Project Prague, 2017, https://www.microsoft.com/en-us/research/project/project-prague/

Tenga en cuenta que cuando la casilla de verificación **Mostrar registro** está habilitada, puede borrar el contenido del registro presionando el enlace **Borrar registro .**

El gráfico en la parte inferior de la sección **Sistema** muestra

- **FPS** : velocidad de fotogramas momentánea, en unidades de fotogramas por segundo.
- **CPU** : consumo de recursos de procesamiento momentáneo, como porcentaje de la potencia de procesamiento total disponible.
- **Memoria** : consumo momentáneo total de RAM (memoria de acceso aleatorio), en unidades de megabytes.

Las métricas **de FPS** y **CPU** se calculan y muestran tanto para el **flujo de trabajo** como para los subprocesos de la **interfaz de usuario** . El subproceso **de flujo de trabajo** es nuestro principal subproceso de trabajo.

Puede controlar qué métricas se trazan en el gráfico activando las casillas de verificación en la parte inferior de la sección **Sistema**.

Sección de Imagen
Esta sección de la **interfaz de usuario del servicio Gestures** muestra el flujo de video IR producido por la cámara de profundidad . Como puede ver, filtramos y mostramos solo los píxeles dentro de un rectángulo alrededor de la mano:

Figura 156. Foto de la mano en el servicio

Fuente: Tomado de Microsoft Project Prague Documentation. (2017)

En la parte superior del marco IR, dibujamos vectores que indican la dirección estimada

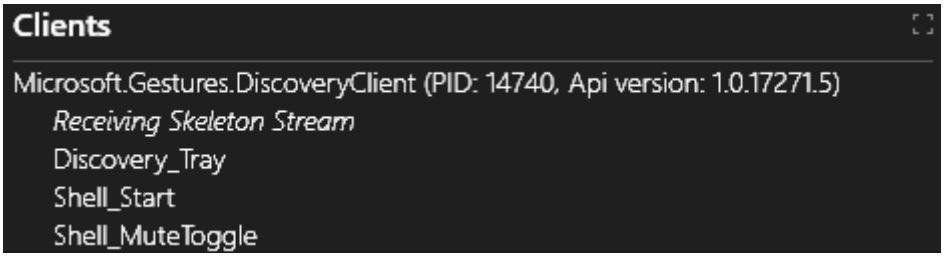
de cada dedo y la ubicación de cada punta del dedo. También dibujamos un sistema de coordenadas local, centrado en la posición de la palma de la mano y compuesto por los siguientes ejes:

- **Azul** : la dirección **perpendicular al plano de la palma** . En nuestra API, nos referimos a este eje como la dirección de la palma .
- **Verde** : la dirección que estaría **alineada con el dedo medio si se hubiera estirado** . En nuestra API, nos referimos a este eje como la orientación de la palma .
- **Rojo** : el producto cruzado de los ejes de dirección de la palma (azul) y orientación de la palma (verde).

Sección de clientes

Esta sección muestra la lista de clientes actualmente conectados al **Servicio Gestures**. Por ejemplo:

Figura 157. Lista de clientes del servicio



Fuente: Tomado de Microsoft Project Prague Documentation. (2017)

Aquí solo hay un único cliente y se muestra la siguiente información para él:

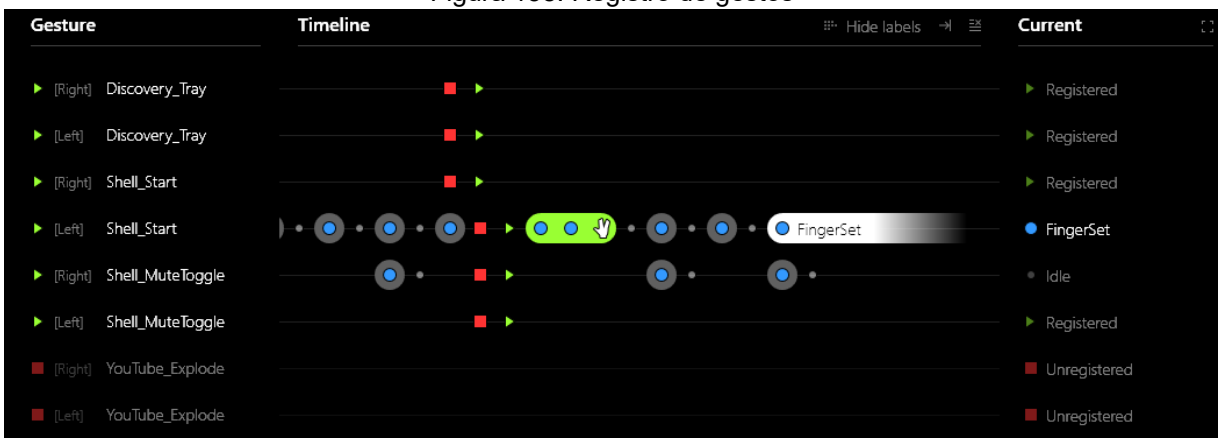
- **El nombre del proceso** del cliente : Microsoft.Gestures.DiscoveryClient en nuestro ejemplo.
- El **Id. de proceso** del cliente : 14740 en nuestro ejemplo.
- La versión de la API del **GesturesServiceEndpoint del cliente** : 1.0.17271.5 en nuestro ejemplo.
- Una indicación de si el cliente **está recibiendo una transmisión esquelética** : nuestro cliente está *recibiendo una transmisión esquelética* .
- **Una lista de todos los gestos que el cliente ha registrado** con el servicio: "Discovery_Tray", "Shell_MuteToggle" y "Shell_Start" en el ejemplo.

Sección de detección de gestos

Detección de gestos muestra el historial de detección de todos los gestos registrados. Recuérdese que todo gesto se representa como una máquina de estados . Una ejecución correcta de un gesto dado corresponde a una secuencia de estados en su máquina de estados, de modo que el primer estado de la secuencia es un estado inicial y el estado final es un estado de recepción.

A continuación se muestra una instantánea de la sección **Detección de gestos** . Cada gesto que se registró en algún momento corresponde a un par de líneas en la interfaz de usuario: una línea para cada mano. Las líneas resaltadas en la parte superior corresponden a gestos actualmente registrados y las líneas atenuadas en la parte inferior corresponden a gestos que actualmente no están registrados.

Figura 158. Registro de gestos



Fuente: Tomado de Microsoft Project Prague Documentation. (2017)

Dependiendo de su definición, un gesto puede ser ejecutado por una mano designada específica (derecha o izquierda) o por cualquiera de las manos. Puede especificar esto al definir HandPoses asociados con el gesto, asignándoles una restricción PalmPose construida usando SingleHandContext o AnyHandContext . Para un gesto dado, la línea **[Derecha]** (**Izquierda**) detalla el historial de ejecución de la mano derecha (izquierda). Solo cuando la casilla **Habilitar dos manos [Experimental]** en la sección **Sistema** de la interfaz de usuario está marcada, la detección de la mano izquierda está habilitada y las líneas **[Izquierda]** se resaltarán.

Hay tres columnas en la sección **Detección de gestos** de la interfaz de usuario:

- **Gesto** : muestra el nombre del gesto y su estado.
- - indica que el gesto está registrado actualmente.
- - indica que el gesto está dado de baja actualmente.
-
- **Línea de tiempo** : muestra el historial de eventos asociados con el gesto correspondiente.
- - indica un evento de registro de gestos.
- - indica un evento de cancelación de registro de gestos.
- - indica el estado de la máquina de gestos avanzado debido a una detección exitosa del siguiente segmento de gesto (estado).
- - indica que la máquina de estado del gesto se restableció al estado inactivo , lo que significa que el gesto se realizó hasta el final o se abandonó.
- - indica que todo el gesto se detectó con éxito.

- **Actual** : especifica el nombre del segmento de gesto actual (estado) dentro de la máquina de estado de gesto.

Sección de registro

La sección Registro muestra un flujo de mensajes producidos por **Gestures Service** en tiempo real. Aquí encontrará diversa información sobre el estado interno del proceso del **Servicio de Gestos** . Cuando **Gestures Service** no se comporta como se esperaba,

intente buscar una indicación de error en el registro.

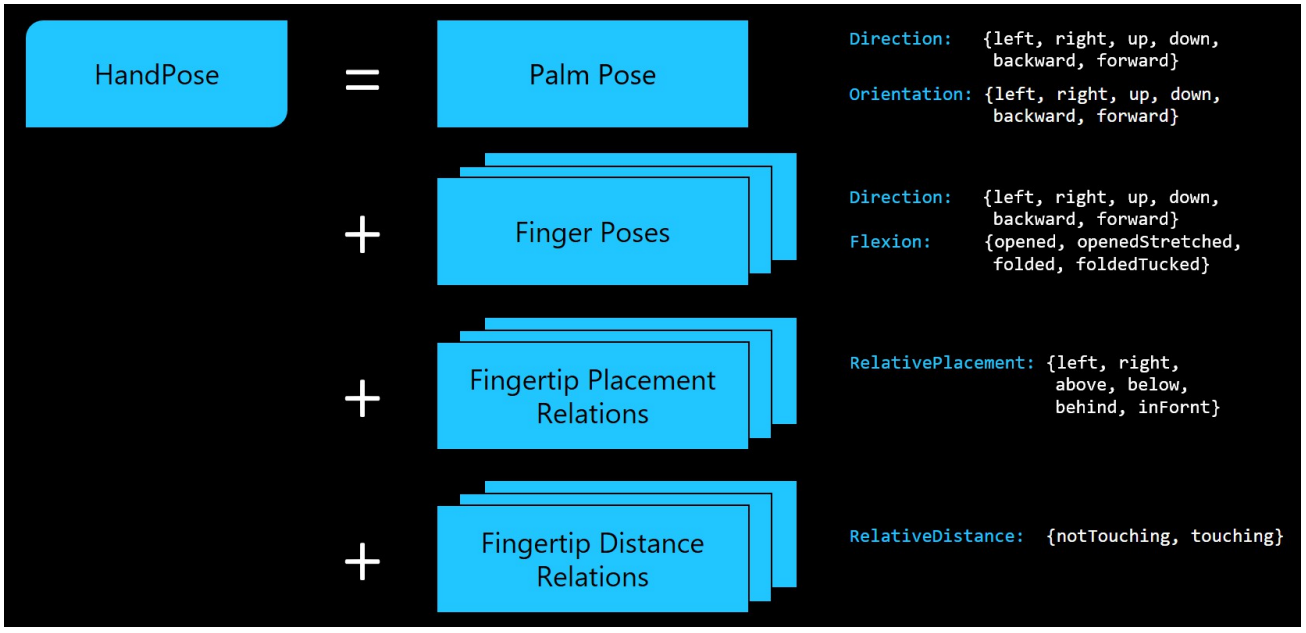
Comprender los gestos en Project Prague

Antes de comenzar a escribir gestos, debe familiarizarse con los **componentes básicos de un gesto: posturas y movimientos de las manos** .

Postura de la mano

Una pose de la mano se refiere a **una instantánea de la mano en un momento dado** . La pose de la mano contiene una descripción completa del estado de la palma y los dedos en esa instantánea. En nuestra API, una pose de mano está representada por la clase HandPose , que se **compone de varias restricciones,** como se ilustra a continuación:

Figura 159. Hand Posture para los gestos



Fuente: Tomado de Microsoft Project Prague Documentation. (2017)

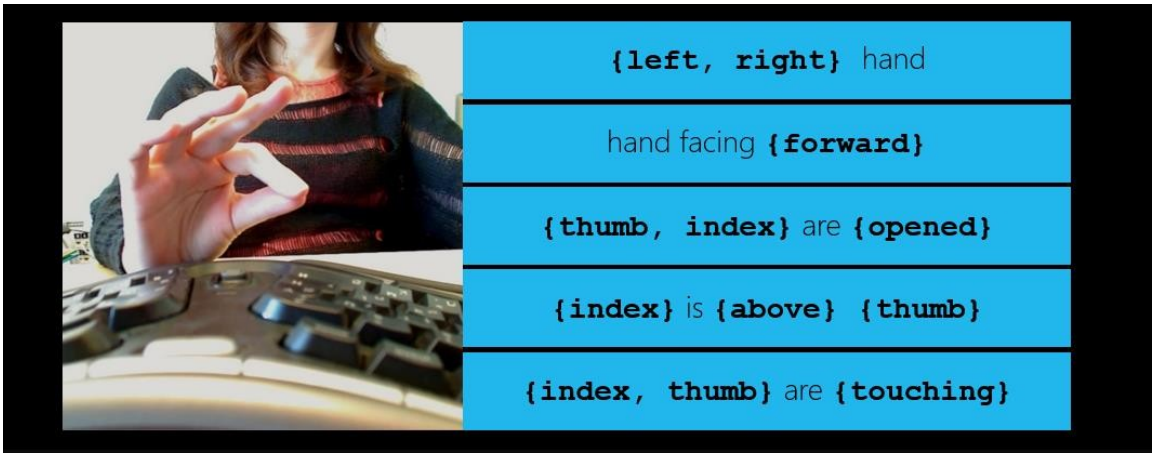
Puede expresar cualquier pose de la mano al caracterizar todas las restricciones involucradas en esa pose, como se ilustra a continuación:

Figura 160. Forma compuesta de gesto

Fuente: Tomado de Microsoft Project Prague Documentation. (2017)

El ejemplo anterior demuestra todas las restricciones que puede asociar razonablemente con la instantánea de la izquierda. Este ejemplo se proporciona únicamente con fines educativos, para mostrarle el significado de los diferentes términos utilizados para especificar restricciones en el idioma del Proyecto Praga. En la práctica, nunca usaría una cantidad tan grande de restricciones para especificar una pose de mano. En su lugar, debe intentar encontrar el número mínimo de restricciones que capturen la esencia de la pose de la mano. El siguiente ejemplo es **una forma práctica de describir la misma pose** :

Figura 161. Forma simple de gesto



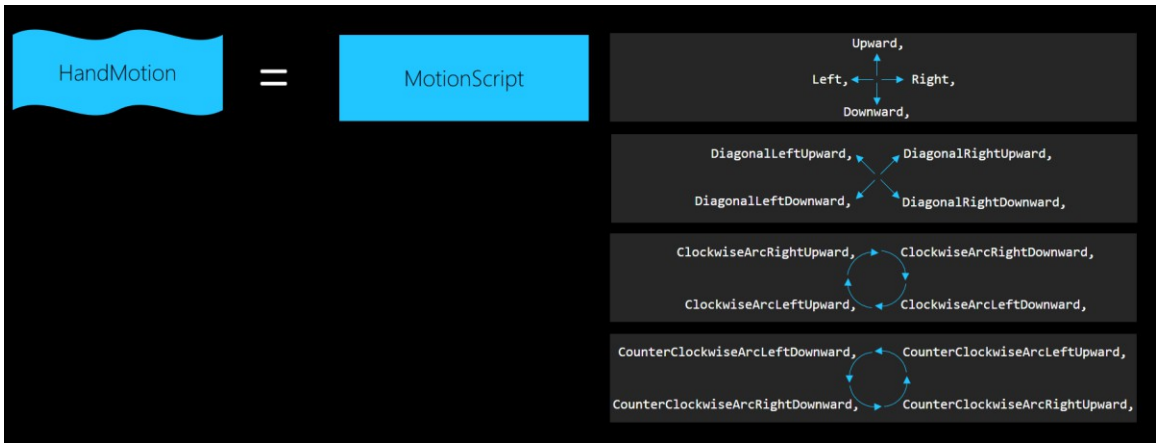
Fuente: Tomado de Microsoft Project Prague Documentation. (2017)

Este último ejemplo capta bien la esencia de la pose en la instantánea: la acción de pellizcar realizada por los dedos pulgar e índice. Tenga en cuenta que los dedos medio, anular y meñique están ausentes de la descripción de la pose de la mano, ya que no participan en la acción de pellizcar y, por lo tanto, no son necesarios para expresar la esencia de la pose.

El uso de demasiadas restricciones al definir una pose de mano (es decir, un ajuste excesivo) **puede producir una pose que es difícil de detectar** . Esto se debe a que sus

usuarios tendrán que ajustar su mano a una postura muy específica para satisfacer todas las restricciones. **El uso de muy pocas restricciones** cuando define una pose de mano (es decir, ajuste insuficiente) puede hacer que sea demasiado fácil detectar la pose, y su **usuario puede realizar la pose sin querer**.

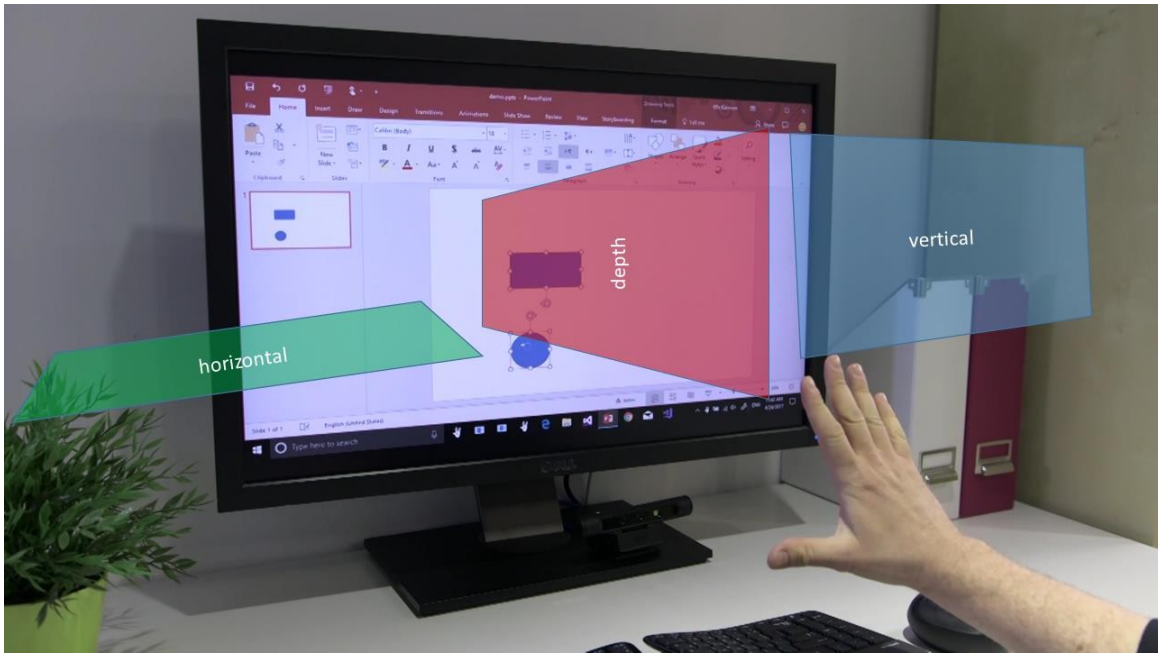
Figura 162. Explicación del Hand Motion



Fuente: Tomado de Microsoft Project Prague Documentation. (2017)

Como puede ver, todos **los componentes básicos describen un movimiento bidimensional** . En otras palabras, todo el movimiento está contenido en un solo plano. Hemos introducido esta limitación intencionalmente para brindar una mejor experiencia al usuario, ya que el movimiento tridimensional de la mano sin restricciones puede ser difícil de ejecutar con precisión. Puede **elegir uno de los tres planos disponibles para contener el movimiento de su mano** , como se ilustra a continuación:

Figura 163. Vista 3D del monitor de la cámara de profundidad



Fuente: Tomado de Microsoft Project Prague Documentation. (2017)

ANEXO 2
RESULTADOS DE LA INVESTIGACIÓN

El objetivo principal de la investigación fue desarrollar un sistema de reconocimiento de gestos altamente adaptable y versátil utilizando el framework Proyecto Praga como base. Este sistema se concibió con la intención de permitir interacciones gestuales intuitivas y precisas en una amplia variedad de aplicaciones, desde entretenimiento hasta entornos médicos y laborales.

Para lograr este objetivo, se exploraron y evaluaron diversas tecnologías y herramientas disponibles en el mercado. Se decidió utilizar Kinect como el dispositivo principal para la detección de gestos debido a su capacidad para capturar movimientos corporales en tiempo real de forma precisa y eficiente. Además, se empleó el InputSimulator para simular acciones de entrada en el sistema operativo, lo que permitió controlar aplicaciones y realizar acciones específicas en respuesta a los gestos detectados.

El sistema se diseñó para reconocer una amplia gama de gestos, desde simples movimientos de mano hasta gestos más complejos como el saludo, el movimiento de rotación o gestos específicos asociados con aplicaciones concretas. Para lograr esto, se implementaron algoritmos de procesamiento de imágenes y técnicas de aprendizaje automático, como redes neuronales convolucionales, para mejorar la precisión del reconocimiento de gestos.

Además de la implementación técnica del sistema, se realizó una integración personalizada con componentes adicionales y lógica específica para adaptar el sistema a los requerimientos del proyecto. Esto incluyó la creación de interfaces de usuario intuitivas que permitieran a los usuarios interactuar de manera natural con el sistema y la configuración de acciones personalizadas en respuesta a diferentes gestos detectados.

Los resultados de la investigación demostraron la efectividad del sistema en la detección y respuesta a gestos en una variedad de escenarios y aplicaciones. Se realizaron pruebas exhaustivas para evaluar la precisión y la fiabilidad del sistema, y los resultados fueron consistentemente positivos. Sin embargo, también se identificaron áreas de mejora potencial, como la optimización del rendimiento para manejar grandes volúmenes de datos y la expansión de las capacidades de reconocimiento de gestos para incluir gestos más complejos y sutiles.

En resumen, la investigación sobre el sistema de reconocimiento de gestos proporcionó una sólida base para el desarrollo de aplicaciones interactivas y manos libres en una variedad de campos. Los resultados obtenidos destacan el potencial de esta tecnología para mejorar la forma en que interactuamos con la tecnología en nuestra vida cotidiana y sugieren numerosas oportunidades para futuras investigaciones y desarrollos en este campo.

Figura 164. Probando la aplicacion

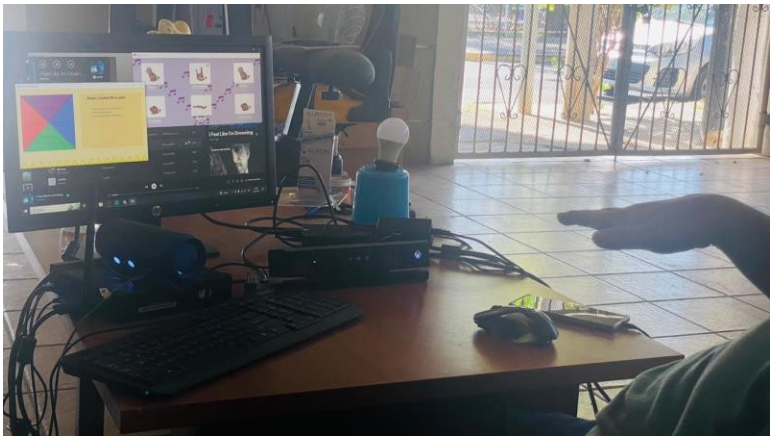


Figura 165. Probando la aplicacion

Figura 166. Probando la aplicacion

Figura 167. Probando la aplicacion

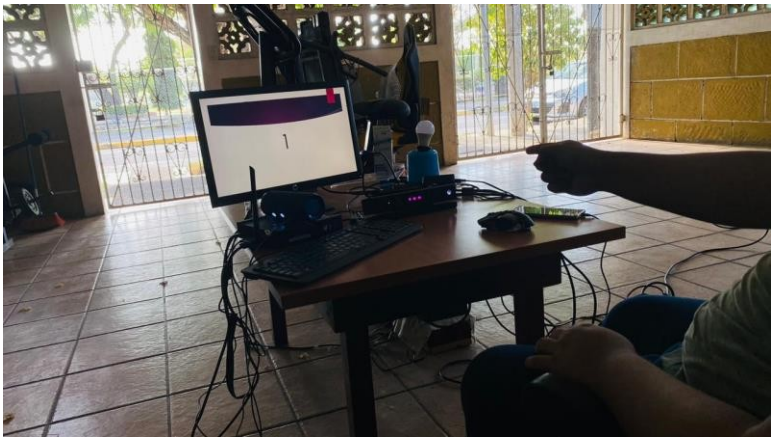


Figura 168. Probando la aplicacion

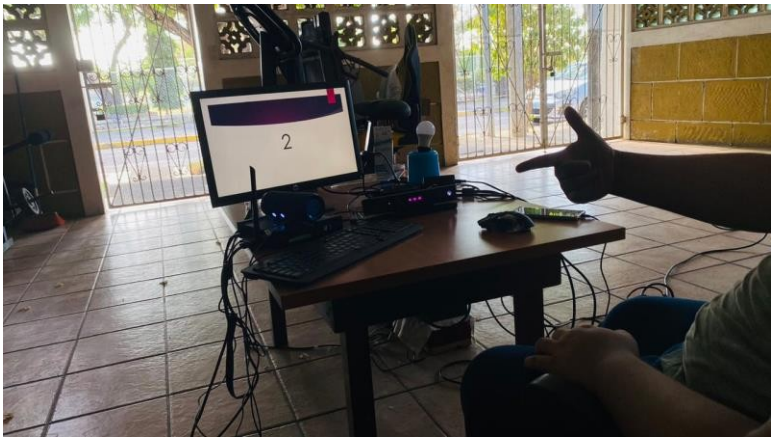


Figura 169. Probando la aplicacion

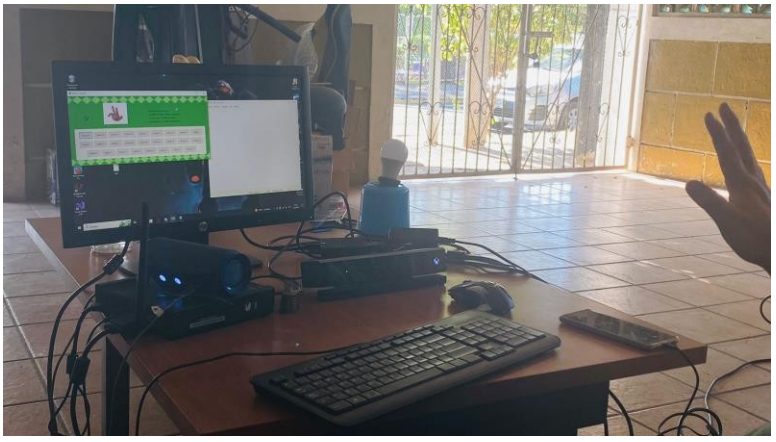
Figura 170. Probando la aplicacion

Figura 171. Probando la aplicacion

Figura 172. Probando la aplicación

Figura 173. Probando la aplicacion

Figura 174. Probando la aplicacion



ANEXO 3
COSTO Y PRESUPUESTO

A continuación se muestra un resumen del costo de montar el proyecto. Dichos precios son costo total, incluso si algo se ha tenido que comprar en Amazon se ha incluido el precio de envío a Nicaragua:

Tabla 5. Presupuesto del Proyecto

Artículo	Función	Precio USD$	Precio en $COR
Kinect V2	Dispositivo de profundidad para manejar los comandos recibidos.	$120.00	C$4416.00
Adaptador Kinect Hyperkin	Adaptador de terceros para conectar el Kinect v2 a Windows	$54.00	C$1987.20
Bombillos Inteligentes TP-Link	Bombillos con API modificable que permiten controles desde el computador	$32.00	C$1177.60
lenovo ThinkCentre M73	Computador sencillo de gama baja para mostrar la aplicación	$72.00	C$2649.60
Mouse Top Game Weapon	Mouse usado para el proyecto.	$8.00	C$294.40
Parlante JBL Flip 4	Parlante usado en la presentación por el módulo de música	$118.32	C$4354.20
Huion Tablet	Pen Tablet utilizada para la creación del UI de la aplicación	$60.00	C$2208.00
Teclado Mecánico	Teclado usado para el proyecto.	$24.00	C$883.20
Otros costos	Resto de periféricos usados como USB, teclados o pantallas	$40.00	C$1472.00