UNIVERSIDAD INTERNACIONAL SEK

FACULTAD DE ARQUITECTURA E INGENIERÍAS

Trabajo de fin de carrera titulado:

“SISTEMA TRADUCTOR SIMULTÁNEO DE SEÑAS A VOZ MEDIANTE UNA INTERFAZ NATURAL DE USUARIO PARA PERSONAS CON DISCAPACIDAD”

Realizado por:

SANTIAGO ALEXANDER CONSTANTE EGAS

ANDRÉS SEBASTIÁN YÉPEZ PASQUEL

Director del proyecto:

ING. JUAN SEBASTIÁN GRIJALVA, MSC.

Como requisito para la obtención del título de:

INGENIERO EN SISTEMAS EN DISEÑO Y MULTIMEDIA

Quito, Julio del 2016

DECLARACIÓN JURAMENTADA

Nosotros, SANTIAGO ALEXANDER CONSTANTE EGAS, con cédula de identidad #171596083-5, y ANDRÉS SEBASTIÁN YÉPEZ PASQUEL, con cédula de identidad #172401035-8 declaramos bajo juramento que el trabajo aquí desarrollado es de nuestra autoría, que no ha sido previamente presentado para ningún grado a calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración, se cede los derechos de propiedad intelectual correspondientes a este trabajo, a la UNIVERSIDAD INTERNACIONAL SEK, según lo establecido por la Ley de Propiedad Intelectual, por su reglamento y por la normativa institucional vigente.

Santiago Alexander Constante Egas

C.C.: 171596083-5

Andrés Sebastián Yépez Pasquel

C.C.: 172401035-8

DECLARATORIA

El presente trabajo de investigación titulado:

“SISTEMA TRADUCTOR SIMULTÁNEO DE SEÑAS A VOZ MEDIANTE UNA INTERFAZ NATURAL DE USUARIO PARA PESONAS CON DISCAPACIDAD”

Realizado por:

SANTIAGO ALEXANDER CONSTANTE EGAS

ANDRÉS SEBASTIÁN YÉPEZ PASQUEL

Como requisito para la obtención del título de:

INGENIERO EN SISTEMAS EN DISEÑO Y MULTIMEDIA

Ha sido dirigido por el docente

ING. JUAN SEBASTIÁN GRIJALVA, MSC.

Quien considera que constituye un trabajo original de su autor

Ing. Juan Sebastián Grijalva, Msc.

DIRECTOR

PROFESOR INFORMANTE

ING. VERÓNICA RODRÍGUEZ, MBA.

Después de revisar el trabajo presentado, lo han calificado

como apto para su defensa oral ante el tribunal

examinador

Ing. Verónica Rodríguez, MBA

Quito, Julio de 2016

PROFESOR INFORMANTE

ING. DANIEL RIPALDA

Después de revisar el trabajo presentado, lo han calificado

como apto para su defensa oral ante el tribunal

examinador

Ing. Daniel Ripalda. MSC.

Quito, Julio de 2016

DEDICATORIA

Dedicamos el presente trabajo de investigación a nuestras familias, por todo el apoyo brindado en el largo de esta carrera universitaria, ya que gracias a ellos se ha logrado a culminar una etapa más en nuestras vidas, dando como resultado un proyecto de gran importancia y de ayuda para la sociedad.

AGRADECIMIENTO

El agradecimiento primordial es a Dios por permitirnos llegar al punto en el que estamos. A nuestras familias, por insistirnos en ponerle empeño al desarrollo del proyecto. A la Ingeniera Verónica Rodríguez, que fue nuestra profesora desde el inicio de la carrera y que siempre podíamos contar con ella si se presentaba algún problema. Al Ingeniero Juan Sebastián Grijalva, por todo el apoyo y dirección en la tesis, por todo el conocimiento que nos brindó y que gracias a ello pudimos crecer profesionalmente. Al Ingeniero Daniel Ripalda por ser una persona que además de enseñarnos Diseño, nos enseñó valores importantes que se deben tomar en cuenta día a día para ser una mejor persona. A la Universidad Internacional SEK, por formarnos como profesionales íntegros.

ÍNDICE GENERAL DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

RESUMEN

El presente documento es el detalle del desarrollo del proyecto denominado Sistema Traductor de Gestos. El proyecto se viene realizando desde hace más de un año y tiene como finalidad la comunicación con personas con discapacidad de habla por medio de una interfaz NUI que permite el reconocimiento de gestos para luego traducirlos a voz. El proyecto en sí cuenta con varias etapas, de las cuales la primera etapa fue cumplida por los estudiantes de la Universidad Internacional SEK, Mario Mauricio Silva Granda y José Luis Amador Abedrabo quienes desarrollaron el prototipo del sistema. Para el desarrollo del presente trabajo se tomó como base dicho prototipo y se buscó mejorar el sistema en todos los aspectos además de cubrir las falencias presentadas en la primera etapa. El sistema se desarrolló con varias características que permitieron alcanzar este fin como por ejemplo con la implementación de detección de manos y dedos, el reconocimiento de movimiento, y la interpretación de varios puntos del cuerpo que antes no se detectaban. Esto se consiguió con la ayuda del sensor Kinect v2 junto con su respectivo SDK y programando un nuevo código y diseño del sistema en lenguaje C# por medio de la API Visual Studio 2012 de Microsoft. Las mejoras en el sistema representaron una significativa mejora en eficiencia y velocidad del sistema en relación con su prototipo.

Palabras clave: NUI, Microsoft Kinect.

ABSTRACT

The present document describes in detail the development of the project called Sistema Traductor de Gestos (Gesture Translating System). The project is an in-development work that seeks to facilitate the communication with speech-impaired people through a NUI (Natural User Interface) which enables gesture recognition and voiced translation. The proyect itself consists of various development stages. The first of these stages was developed by Universidad Internacional SEK students Mauricio Silva Grand and José Luis Amador Abedrabbo who succesfully built the prototype of the system. This prototype was used as a base for the development of the present work with the finality to improve the system from the base up and to fix the current existent flaws found in the prototype. The system was developed with various characteristics that allowed this idea to succeed, for example the implementation of hand and finger detection, the recoding of movement recognition and the interpretation of various body parts which were not detectable in the previous version. All of these changes were achieved with the aid of the sensor Kinect V2 with its corresponding SDK as well as programming a new base code and design for the system using C# programming language in Window's Visual Studio 2012. The improvements developed proved to be a significant advance in efficiency in the system compared to its prototype.

Key Words: NUI, Microsoft Kinect.

CAPÍTULO I

INTRODUCCIÓN

1.1 El Problema de Investigación

1.1.1 Planteamiento del problema

La comunicación es una de las actividades más elementales de la humanidad, sin ella la humanidad no hubiera logrado progresar en ninguna área. Cuando una persona es impedida por alguna causa de la capacidad de hablar, la comunicación se ve severamente afectada ya que la única forma de hacerse entender es usar su cuerpo por medio de gestos y señas que demuestren lo que quieren decir y les permite comunicarse con su entorno. Sin embargo la comunicación es mucho más lenta e ineficiente que la comunicación hablada. El prototipo del Sistema Traductor de Gestos presentó una herramienta que permite traducir ciertos gestos y señas a voz facilitando la comunicación, sin embargo al ser un prototipo presento varias limitaciones como por ejemplo su lentitud, la confusión de movimientos similares y los limitados gestos capaces de ser recreados. Siendo una herramienta de comunicación, el sistema necesita poder captar más rápida y fielmente los gestos realizados por el usuario, así como también tener la capacidad de ser más flexible a la hora de programar gestos que puedan ser adaptados a las necesidades del usuario.

1.1.2 Objetivo General.

Desarrollar el Sistema Traductor Simultáneo de Señas a Voz por medio de la utilización del sensor Kinect v2 con su respectivo SDK y el diseño y elaboración de un nuevo código fuente para la mejora del sistema en todos sus aspectos en relación con el prototipo.

1.1.3 Objetivo Específicos.

- Analizar el prototipo para la obtención de información sobre el estado actual del sistema para el levantamiento de requerimientos y la identificación de las fallas y posibles mejoras en el sistema.
- Diseñar la interfaz de usuario y la arquitectura del sistema por medio de Visual Studio 2010 de manera que provean una herramienta de interacción entre el usuario, el sistema y el sensor.
- Programar el código del sistema mediante el lenguaje C# y el diseño planteado para la elaboración de la etapa de desarrollo.
- Verificar el funcionamiento del sistema por medio de pruebas de software para la corrección de errores presentes.

1.1.4 Justificación

El presente trabajo comprende el desarrollo del proyecto denominado “Sistema de Traducción Simultánea de Señas a Voz” cuya finalidad es ayudar a que la comunicación con personas impedidas del habla sea más fácil, en el sentido de proveer una interfaz de

comunicación por voz permitiendo así que estas personas puedan utilizar una mayor cantidad de servicios de una forma más eficiente, por el hecho de que el Sistema de Traducción Simultánea de Señas a Voz actuaría como un intermediario entre la persona impedida de habla con las personas que ofrecen algún servicio.

El prototipo del STDG fue creado en la etapa anterior del proyecto, demostrando que el sistema funciona y puede potencialmente ser una herramienta muy útil para la inclusión social de personas impedidas de habla. Sin embargo, la tecnología utilizada en el prototipo presentaba algunas limitaciones que debían ser mejoradas. El sistema no permitía una detección rápida y fiel y su campo de visión, resolución y movimientos impedían proveer una herramienta eficaz de comunicación. El cambio de tecnología y la reprogramación y rediseño del sistema presentado en este proyecto mejoran significativamente el sistema en todas las debilidades expuestas anteriormente, así como también presenta nuevas fortalezas que antes carecía.

1.2 Marco Teórico

1.2.1 Ingeniería de Software

Es un proceso que está compuesto por todos los pasos para el desarrollo un producto enfocado a la utilización de software. Los pasos mencionados abarcan desde la planificación hasta el mantenimiento del software, es decir, el proceso empieza desde antes de desarrollar el software hasta después de haberlo finalizado. *"En general los ingenieros de software adoptan un enfoque sistemático y organizado en su trabajo, ya que es la forma efectiva de producir software de calidad"* (Summerville, 2005).

1.2.2 Proceso de Desarrollo de Software

Un proceso de desarrollo de software es un conjunto de procedimientos, componentes de software, metodologías, y herramientas utilizadas para desarrollar un servicio o extender un producto de software.

Cuando se habla de desarrollo se necesita de un modelo de proceso de software cuya funcionalidad esté probada en la práctica, y personalizarlo para que cumpla con necesidades específicas. Es decir, Una organización podría variar su modelo de proceso para cada proyecto, según la naturaleza del proyecto, a naturaleza de la aplicación, los controles y entregas requeridas y las herramientas que se van a utilizar.

Los modelos genéricos de desarrollo de software son aquellos que son los más comúnmente aceptados, y se describen a continuación:

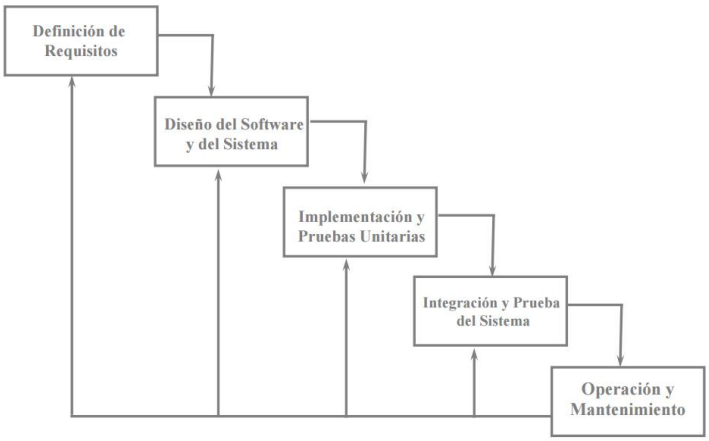
1.2.2.1 Modelo en Cascada

Según Méndez (2009), el modelo de cascada define las siguientes etapas que deben cumplirse de forma sucesiva:

1. Especificación de requisitos
2. Diseño del software
3. Construcción o Implementación del software
4. Integración
5. Pruebas (o validación)

6. Despliegue (o instalación)
7. Mantenimiento

Figura 1. Fase del Modelo en Cascada.
Fuente: Méndez, G., 2009 Proceso de software y ciclo de vida



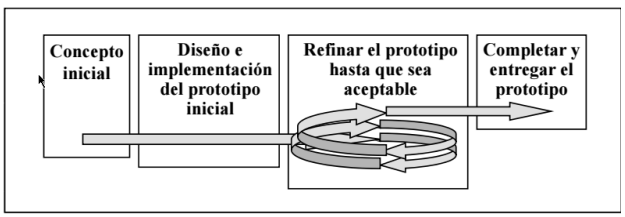
Es un ciclo de vida en sentido amplio, que incluye no sólo las etapas de ingeniería sino toda la vida del producto: las pruebas, el uso (la vida útil del software) y el mantenimiento.

1.2.2.2 Modelo de Construcción de Prototipos

La idea detrás de este modelo según explica Moreno (2009), es el desarrollo de un prototipo del sistema inicial, exponerlo a los comentarios del usuario, pulirlo en un número de versiones hasta que se desarrolle el sistema requerido.

Una ventaja de este modelo es que se obtiene una rápida realimentación del usuario, ya que las actividades de especificación, desarrollo y pruebas se ejecutan en cada iteración.

Figura 2. Proceso del Modelo de Construcción de Prototipos.
Fuente: Moreno, M., 2009, Modelos de proceso del software.



1.2.2.3 Modelo Espiral

El ciclo de desarrollo se representa como una espiral, en lugar de una serie de actividades sucesivas con retrospectiva de una actividad a otra.

Méndez (2009) nos dice que cada ciclo de desarrollo se divide en cuatro fases:

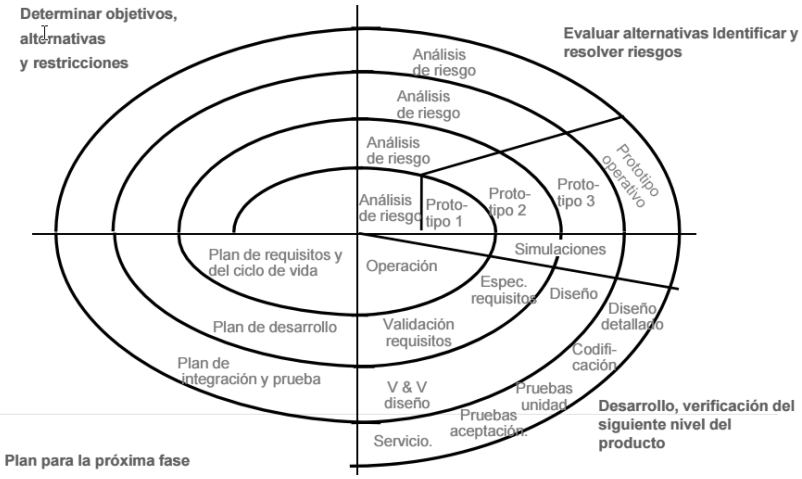
1. Definición de objetivos: Se definen los objetivos. Se definen las restricciones del proceso y del producto. Se realiza un diseño detallado del plan administrativo. Se identifican los riesgos y se elaboran estrategias alternativas dependiendo de estos.

2. Evaluación y reducción de riesgos: Se realiza un análisis detallado de cada riesgo identificado. Pueden desarrollarse prototipos para disminuir el riesgo de requisitos dudosos. Se llevan a cabo los pasos para reducir los riesgos.

3. Desarrollo y validación: Se escoge el modelo de desarrollo después de la evaluación del riesgo. El modelo que se utilizará (cascada, sistemas formales, evolutivo, etc.) depende del riesgo identificado para esa fase.

4. Planificación: Se determina si continuar con otro ciclo. Se planea la siguiente fase del proyecto.

Figura 3. Modelo de espiral
Fuente: Méndez, G., 2009 Proceso de software y ciclo de vida



Este modelo a diferencia de los otros toma en consideración explícitamente el riesgo, esta es una actividad importante en la administración del proyecto.

Aparte de los modelos genéricos otros nuevos modelos de desarrollo de software han sido creados para diferentes situaciones, como por ejemplo el modelo RAD.

1.2.2.4 Modelo RAD (Rapid Application Development)

El modelo de desarrollo rápido de aplicaciones o RAD fue creado por James Martin en 1980. El método comprende el desarrollo interactivo, la construcción de prototipos y el uso de utilidades como por ejemplo las interfaces gráficas. El método se divide en cuatro fases las cuales son:

Planificación de requerimientos: Combina elementos de las fases de planificación de requerimientos y análisis de requerimientos del método de la cascada. Tanto usuarios como desarrolladores se reúnen llegan a un acuerdo sobre el alcance y los resultados esperados del sistema.

Fase de Diseño: Durante esta fase los desarrolladores crean modelos y diseñan el Sistema de acuerdo a los requerimientos discutidos y se los presentan a los usuarios para obtener retroalimentación.

Fase de Construcción: Se enfoca en el desarrollo de la aplicación. Las tareas encontradas en esta fase incluyen desarrollo de interfaces, programación de código y validación.

Fase de Pruebas y Finalización: En esta fase se encuentran las tareas de clausura como son la realización de pruebas, implementación, y capacitación.

Si se comprenden bien los requisitos y se limita el ámbito del proyecto, el proceso DRA permite al equipo de desarrollo crear un sistema completamente funcional dentro de periodos cortos de tiempo. Martel (2002)

1.2.3 Adopción de una perspectiva teórica

Para el desarrollo de la segunda etapa del Sistema Traductor de Gestos se utilizó el modelo de desarrollo rápido de aplicaciones. Se eligió este modelo de proceso de software por las siguientes razones:

1. El modelo pone mayor énfasis en el desarrollo de la aplicación y menor énfasis en las etapas de planificación y especificación. Debido a la naturaleza del presente trabajo (por ejemplo no ser un sistema a ser implementado en una empresa) las rigurosas fases de análisis de requerimientos de usuario y requerimientos de implementación presentados en el modelo de cascada no eran aplicables a este trabajo.

2. Al tener características del modelo de proceso de Construcción de Prototipos, el Modelo de Desarrollo Rápido también se basa en la creación de versiones del sistema sistemáticas hasta alcanzar el resultado deseado. Esto permite que se desarrolle de forma tanto intuitiva como flexible, así como también permite detectar de forma rápida cualquier error que podría ser más complicado detectar usando otro modelo de desarrollo.

3. Al tener características del modelo de proceso en Cascada, el Modelo de Desarrollo Rápido denota un proceso lineal de programación que añade control al proceso ya que promueve el desarrollo por módulos o etapas.

1.2.4 Visual Studio 2012

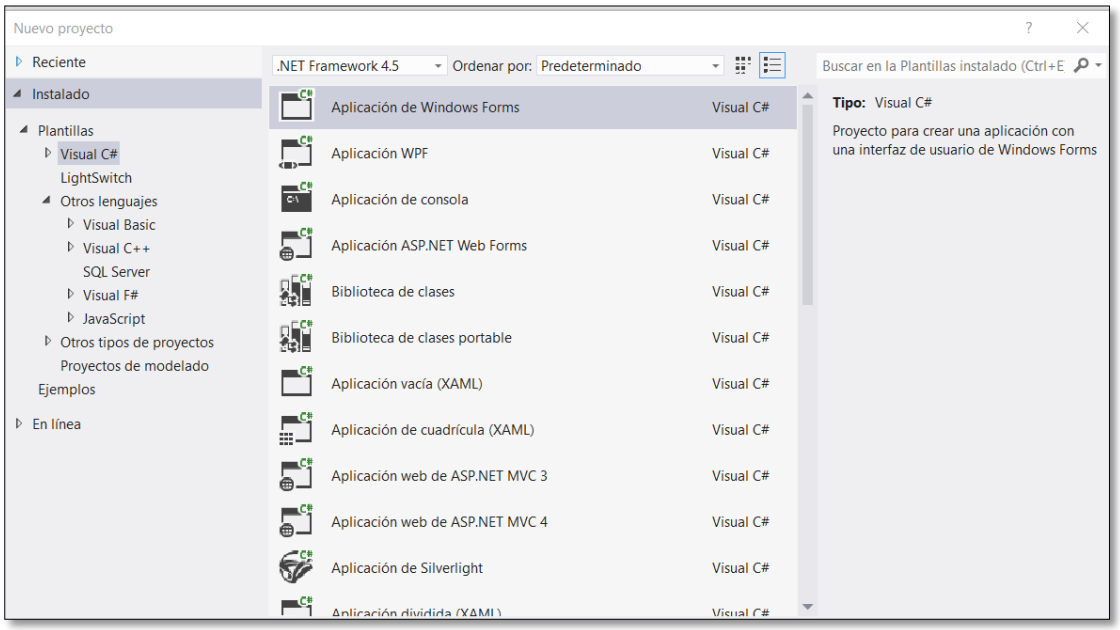
Es un programa dedicado a al desarrollo de aplicaciones, donde abarca varios tipos de lenguaje para crear una gran diversidad de aplicaciones que sean compatibles con otros programas.

Primeramente para empezar a desarrollar una aplicación, se debe elegir el entorno en el que se va a trabajar, es decir, seleccionar el lenguaje de programación que se va a utilizar, ya que dependiendo de la elección que se haga, van a variar los tipos de aplicaciones que se puedan realizar este proceso se le conoce como selección del IDE.

Para comenzara con el desarrollo del código, se debe crear un espacio que contendrá toda la información del programa, este lugar se denomina solución. Las soluciones se crean

automáticamente cuando se crea un proyecto. Los proyectos los espacios en donde se compila y depura el código del programa.

Figura 4. Captura de pantalla de Visual Studio 2012: Creación de proyecto.
Elaborado por: Santiago Constante, Andrés Yépez.



1.2.4.1 C#

Es un lenguaje de programación utilizado para la creación de aplicaciones. Presenta la función de ensamblado la cual sirve para tomar varios archivos y del programa y agruparlos para posteriormente permitir instalar las aplicaciones. Una definición muy clara de lo que es C# dada por el autor Luis Joyaes es *"C# es un idioma diseñado para proporcionar una combinación optima entre sencillez, claridad y rendimiento y es el lenguaje en el que se han escrito los servicios de la plataforma."* (Joyanes, 2002).

El proceso para la creación de un programa consta de varias etapas:

Primero tenemos a la etapa de análisis, consiste en definir qué es exactamente lo que el programa debe hacer, es una etapa fundamental debido a que si está mal planteada la definición, el programa no llegara a realizar las actividades que debería hacer, por ende, especificar detalladamente lo que se desea que haga, se podrá continuar a las siguientes etapas de la creación de un programa.

Luego continuamos a la etapa de diseño, en la cual especificamos la forma en la que se va a desarrollar el programa, es decir, si se aplicaran estructuras, matrices, clases, etc. Es decir se define todos los algoritmos que se piensa utilizar en el desarrollo del programa, como también, la ubicación de cada uno de los mismos, porque, se puede contener código en el cuerpo principal del programa o tener código en diferentes funciones que serán llamadas desde el código principal.

Posteriormente, la etapa de implementación o codificación, en donde, con todos los elementos definidos anteriormente, se procede a escribir las líneas de código necesarias para que el programa cumpla con su objetivo, haciendo de diferentes elementos como, declaración de variables, establecimiento de funciones, colocación de ciclos repetitivos, etc.

Por último se pasa a la etapa de prueba o depuración en donde se hace una simulación del funcionamiento del programa, en donde podemos encontrar diferentes errores, como falta de comas, errores de sintaxis, variables no declaradas, ciclos infinitos, etc. Si en esta etapa de pruebas se encuentra con algún tipo de error, se debe regresar a la etapa anterior para corregir

estos errores y posteriormente pasar de nuevo a la etapa de pruebas hasta que ya no obtengamos ningún error.

Dentro de C#, se puede observar que utiliza unas ciertas palabras reservadas para la programación, como por ejemplo, “public”, que vendría a ser la interpretación de que una función o variable es de carácter público, es decir, que cualquier método o variable pueden realizar interacciones con las mismas. Otra palabra reservada tenemos a “int”, que puede ser utilizado para definir una variable con valores enteros o para que las funciones devuelvan como resultado un valor entero. La palabra “static” está reservada para definir que las funciones únicamente se pueden relacionar con clases y no con objetos. Para definir al cuerpo principal del programa se coloca al inicio la palabra “main”, la cual normalmente está acompañada de un nombre, seguido de dos paréntesis, uno abierto y uno cerrado, y por ultimo unas llaves que muestran el inicio y el final del cuerpo principal, en general las llaves tienen esa función. Además de las mencionadas, existe una gran cantidad de palabas reservadas, que se deben tomar en cuenta al momento de ponerle un nombre a una variable, porque si se pone el mismo nombre que una palabra reservada, puede ocasionar que se presenten errores.

Para poder utilizar diferentes funciones en un mismo programa se tienen que tener establecido al inicio del mismo sus referencias, que se las puede interpretar como solicitudes para hacer uso de diferentes funciones, por ejemplo, se desea que cuando salga un mensaje de error aparezca una ventana flotante que indique que sucedió. Para poder crear la ventana, primero necesitamos establecer la referencia WindowsForm, después, se procede a utilizar las funciones establecidas en la referencia para hacer que la ventana aparezca.

Para llamar a una función se debe completar los siguientes parámetros: <NombreDeFuncion>.<Clase>.<Metodo> (<Parametros>). En donde se define exactamente cuál es la función y que parámetros se necesita para utilizarla, hay que tomar en cuenta de que las funciones pueden o no contener parámetros. Además, se debe verificar que no haya espacios en blanco al momento de hacer uso de una clase, se debe escribir tal como se mostró en el ejemplo anterior o utilizar "_" para separar palabras.

1.2.5 SDK (Software Development Kit)

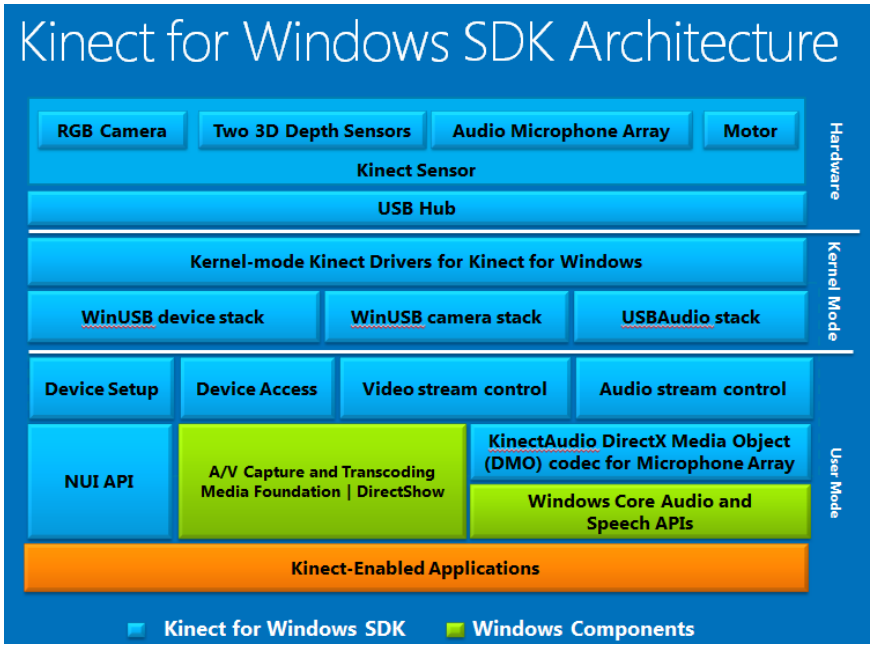
El kit de desarrollo de software o más conocido como SDK es por norma general un conjunto de herramientas de desarrollo de software que permite a un programador crear programas y aplicaciones para un sistema o plataforma concretos.

Suelen incluir un soporte para la detección de errores, un entorno de desarrollo integrado y otro tipo de utilidades. Suelen incluir también algún tipo de explicación de lo que contiene, algunos códigos de ejemplos y un manual de uso.

La mayoría de los kits de desarrollo de software son gratuitos y se distribuyen libremente por internet para así fomentar la colaboración de los desarrolladores para mejorar su producto y para que utilicen su lenguaje.

Microsoft decidió sacar el sensor "Kinect for Windows" el 20 de Noviembre del 2010 debido a que detectó un vacío en el mercado, que inicialmente no supo prever, pero que una vez detectado, rápidamente se apresuró a hacerse con él.

Figura 5. Arquitectura Windows SDK.
Fuente: Microsoft.



El Kinect SDK (Software Development Kit) se trata de una librería que facilita diferentes funciones que ayudan a interactuar con el dispositivo Kinect.

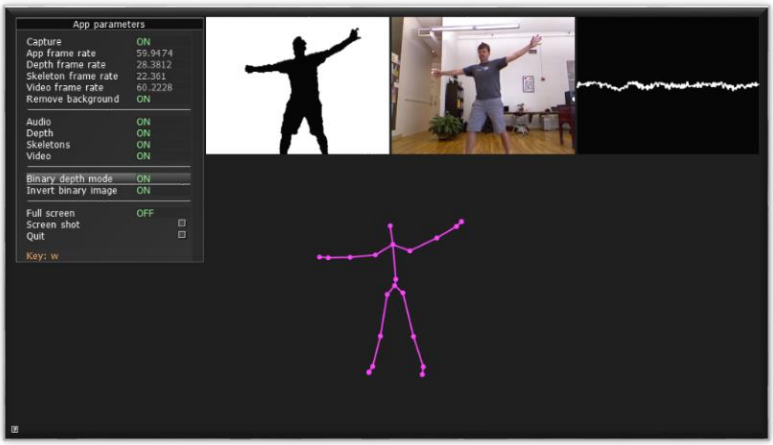
Kinect, una vez detecta el esqueleto humano, es capaz de facilitarnos información detallada de la posición exacta en el plano (X, Y, Z) de todas y cada una de las articulaciones en las que divide el esqueleto humano. Es gracias a esa información lo que permite que podamos desarrollar aplicaciones que funcionen con la interacción del cuerpo humano, sin necesidad de teclados, ratones ni touchpads.

Desde el lanzamiento de Kinect para XBOX 360, ha habido muchos desarrolladores que adaptaron el dispositivo para que pudiera ser utilizado desde un PC, gracias al adaptador a

puerto USB, para trabajar con él y darle otros usos que no fueran únicamente lúdicos para la consola de Microsoft; XBOX 360.

Rápidamente aparecieron librerías openSource que ayudaban a los distintos desarrolladores a interactuar con el dispositivo. Fue entonces cuando Microsoft decidió ponerse manos a la obra con el desarrollo del SDK para ofrecérselo a los programadores.

Figura 6. Kinect v2 muestra de esqueleto.
Fuente: Cinder



1.2.6 Kinect

Es un dispositivo que cuenta con un sensor que realiza un escaneo de una persona y obtiene la posición en la que se encuentra. Esta información la obtiene una máquina la cual la interpreta de forma de coordenadas de diferentes puntos ubicados en distint partes del cuerpo. Este dispositivo cuenta actualmente con dos versiones: que son Kinect V1 y V2.

1.2.6.1 Kinect V1

Se basa es una técnica de luz estructurada que emite haces de luz en un área específica para saber la ubicación del cuerpo. Esta versión no puede captar movimiento del cuerpo pero si comparar una posición inicial con una posterior para interpretar que la persona se ha movido.

1.2.6.2 Kinect V2

Cuenta con una cámara TOF (Time of fligth) la cual permite escanear un área determinada capturando todo lo que esté en su rango. Esta versión permite el reconocimiento de dedos y la posibilidad de captar movimiento.

A continuación se muestra las características de las 2 versiones de Kinect:

Tabla 1. Cuadro Comparativo Kinect v1 vs Kinect v2.
Elaborado por: Santiago Constante, Andrés Yépez

Características	Kinect v1	Kinect v2
Campo de visión horizontal	57º	70º
Campo de visión vertical	43º	60º
Resolución	640 x 480	1920 x 1080
Rango de profundidad mínimo	0,8 metros	0,5 metros
Rango de profundidad máximo	4 metros	4,5 metros
Puerto USB	UBS 2.0	USB 3.0
Capta movimiento en la obscuridad	No	Si
Capta estados de las manos	No	Si
Número de puntos del esqueleto	20	25
Versión de Sistema Operativo	Windows 7 o superior	Windows 8.1 o superior

CAPÍTULO II

MÉTODO

2.1 Análisis

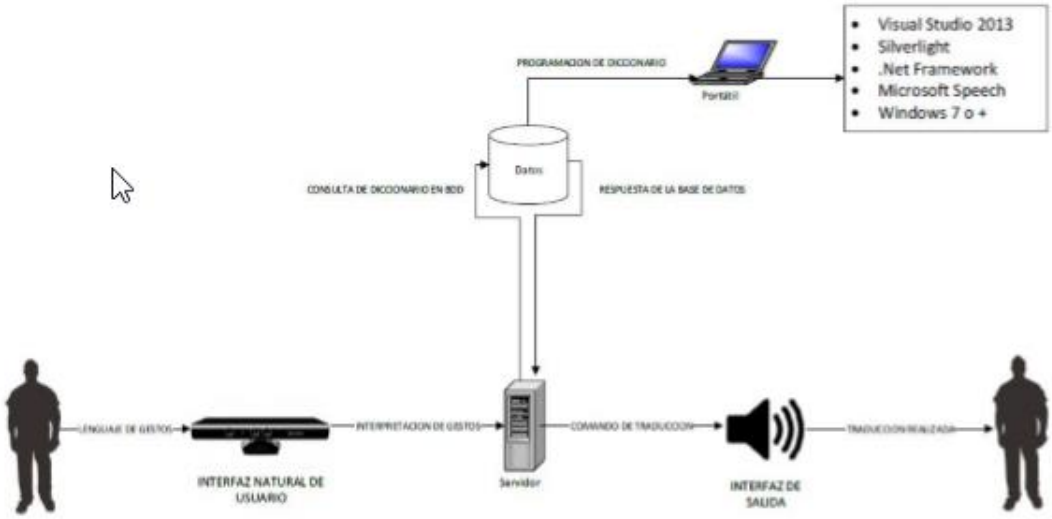
2.1.1 Estudio Preliminar

El estudio preliminar para el presente trabajo se realizó analizando el prototipo del sistema realizado en la primera etapa. Para el desarrollo del prototipo se estudiaron términos y herramientas adaptadas para la programación de la aplicación. El sistema utilizó una Interfaz Natural de Usuario (conocido normalmente como NUI por sus siglas en inglés), el MSc. Juan Sebastián Grijalva lo explicó de esta forma*: "La interfaz natural de usuario o NUI, permite interactuar con sistemas o aplicaciones a través de señas o movimientos remplazando otros dispositivos de entrada de uso habitual como son: teclado, ratón, lápiz óptico, joystick, entre otros."*(Grijalva, S., 2015*).*

Con este fin el desarrollo del sistema utilizó la herramienta de Microsoft Kinect, la misma que es utilizada en la consola de videojuegos de la misma Xbox 360 que interpreta los movimientos naturales del usuario para realizar distintos movimientos virtuales en un juego, la idea era revolucionaria *"Kinect fue creado con el objetivo de revolucionar la experiencia que tenía el usuario al momento de controlar su consola de juegos solo con gestos corporales o comandos de voz. Este dispositivo incorpora una arquitectura de varios elementos para su correcto funcionamiento"* (Zhang, Z., 2012). Para el sistema de traducción simultánea de señas a voz, se aprovechó el potencial de este intérprete de movimientos utilizando el kit de

programación de Microsoft propia del sensor Kinect, Kinect SDK. El prototipo presenta la siguiente arquitectura:

Figura 7. Arquitectura del Kinect
Fuente: Tesis Kinect UTE Jose Amador- Mario Silva



Debido a que las señas realizadas son recibidas como tramas, el sistema necesita poder interpretarlas y traducirlas de forma que el sistema pueda utilizarlas. El prototipo del sistema utiliza todas las librerías proporcionadas por el SDK de Kinect para facilitar proceso en las siguientes porciones del código:

```
KinectSensor kinectSensor;
SwipeGestureDetector swipeGestureRecognizer;
readonly ColorStreamManager colorManager = new ColorStreamManager();
SkeletonDisplayManager skeletonDisplayManager;
readonly ContextTracker contextTracker = new ContextTracker();
ParallelCombinedGestureDetector parallelCombinedGestureDetector;
readonly AlgorithmicPostureDetector algorithmicPostureRecognizer = new AlgorithmicPostureDetector();
private bool recordNextFrameForPosture;
BindableNUICamera nuiCamera;
private Skeleton[] skeletons;
```

Algunas de estas librerías, sin embargo, no tienen ninguna funcionalidad en el sistema debido a que no son necesarias para su funcionamiento, o directamente no realizan ningún proceso. Por ejemplo la librería SwipeGestureDetector cuya función es detectar el movimiento horizontal de un brazo para el otro lado, no realiza ninguna acción ya que el sensor Kinect no detecta el movimiento si se usa en conjunción con la librería AlgorithmicPostureDetector que funciona para la interpretación de posturas estáticas.

Posteriormente, el sistema del prototipo utiliza algunas funciones in-line para realizar los procesos de inicialización del Kinect, así como los procesos de la gestión de las tramas, y de visualización de los elementos de la interfaz.

```
void Kinects_StatusChanged(object sender, StatusChangedEventArgs e)...
private void Window_Loaded(object sender, RoutedEventArgs e)...
private void Initialize()...
void kinectRuntime_ColorFrameReady(object sender, ColorImageFrameReadyEventArgs e)...
void kinectRuntime_SkeletonFrameReady(object sender, SkeletonFrameReadyEventArgs e)...
void ProcessFrame(ReplaySkeletonFrame frame)...

private void comprobador(Skeleton skeleton)...
private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)...
private void Clean()...
void replay_ColorImageFrameReady(object sender, ReplayColorImageFrameReadyEventArgs e)...
void replay_SkeletonFrameReady(object sender, ReplaySkeletonFrameReadyEventArgs e)...
private void seatedMode_Checked_1(object sender, RoutedEventArgs e)...
private void seatedMode_Unchecked_1(object sender, RoutedEventArgs e)...
```

El prototipo incluyó en la función comprobador, que contiene las funciones de asignación de puntos del cuerpo para su reconocimiento así como la función que crea el esqueleto para su proyección en la interfaz, los algoritmos de comprobación de los gestos que serán traducidos. Tomando como ejemplo el gesto “Estoy fuerte” los algoritmos se veían de esta forma:

```
int contadorfuerte = 0;

if (pieizq.Position.X < hombroizq.Position.X && pieder.Position.X > hombroder.Position.X)
{
    contadorfuerte = 1;
}
if (contadorfuerte == 1)
{
    if (cododer.Position.Y > hombroder.Position.Y && codoizq.Position.Y > hombroizq.Position.Y)
    {
        contadorfuerte = 2;
    }
    else
    {
        contadorfuerte = 0;
    }
}
if (contadorfuerte == 2)
{
    if (manoder.Position.Y < cabeza.Position.Y && manoizq.Position.Y < cabeza.Position.Y)
    {
        txtHabla.Text = ("Estoy Fuerte");
        BotonHablar_Click(null, null);
        HistorialLista.Items.Add(txtHabla.Text);
    }
    else
    {
        contadorfuerte = 0;
    }
```

Finalmente el prototipo incluía una función para activar la traducción de texto a voz

```
private void BotonHablar_Click(object sender, RoutedEventArgs e)
{
    int indice;
    indice = Voz.SelectedIndex;
    String nombre = vocesInfo.ElementAt(indice).Name;
    synthesizer.SelectVoice(nombre);
    synthesizer.Speak(txtHabla.Text);

}
```

Todas las funciones utilizadas en el código del prototipo son funciones in-line incluidas en el programa principal, es decir, no se creó ninguna clase propia para el desarrollo del sistema. Es por esto que el código resulta un poco difícil de comprender y de modificar. El hardware utilizado para el desarrollo del prototipo está descrito en el siguiente gráfico realizado por los autores del mismo.

Figura 8. Hardware Utilizado
Fuente: Tesis Kinect UTE Jose Amador- Mario Silva

LISTA DE HARDWARE UTILIZADO		
Procesador		Componente electrónico donde se realizan los procesos lógicos, con el cual permite correr aplicaciones y programas.
Disco Duro		Elemento de almacenamiento de información utilizado en los computadores para archivar documentos y programas.
Memoria RAM		Elemento utilizado para almacenar información volátil que necesitan las aplicaciones para funcionar.
Sensor Kinect		Dispositivo de reconocimiento de gestos utilizado para detectar coordenadas del cuerpo.
Placa Madre		Hardware que permite el ensamblaje de todos los componentes necesarios para la comunicación entre elementos como: procesador, disco duro y memoria RAM.
Fuente de Poder		Dispositivo que regula y provee de alimentación al resto de componentes del computador para su correcto funcionamiento.
Adaptador PC Kinect		Elemento indispensable en la conexión del sensor Kinect hacia el computador para su correcto funcionamiento.
Mouse		Interfaz de control del computador permitiendo desplazarse en la interfaz de usuario.
Parlantes		Equipo responsable de transmitir audio del computador hacia el usuario.
Teclados		Interfaz de comunicación del computador para recibir instrucciones de parte del usuario.
Monitores		Dispositivo que sirve de interfaz visual del computador al usuario.

Como paso final del desarrollo del prototipo, se construyó un mueble el cual serviría el sistema de traducción. (Ver Anexo 1)

2.1.2 Estudio de Factibilidad

2.1.2.1 Factibilidad Operativa

Para la administración y mantenimiento del sistema traductor simultáneo se necesita una persona especializada en el área de sistemas, que tengan conocimiento en desarrollo de software para realizar cambios en el programa de tal manera que se adaptarse a diferentes ambientes. En el proceso de desarrollo de software se utilizaron los siguientes recursos:

Tabla 2. Perfiles y Competencias.
Elaborado por: Santiago Constante, Andrés Yépez.

Nombre	Descripción	Tipo
Santiago Constante	Estudiante	Trabajo
Andrés Yépez	Estudiante	Trabajo
Visual Studio 2012	Software	Material
NUI Kinect V2	Software	Material
Laptop Programador	Equipo	Material
SKD Kinect v2	Software	Material
STDG Piloto	Equipo	Material

Los recursos humanos mencionados anteriormente son los estudiantes encargados de realizar el presente proyecto. Los recursos materiales son las herramientas que fueron utilizadas para el desarrollo del programa.

2.1.2.2 Factibilidad Tecnológica

Los requerimientos mínimos que se necesita para poder desarrollar el sistema traductor son de hardware y software. En el siguiente grafico se especifica los requerimientos:

Figura 9. Requerimientos mínimos de Kinect v2.
Elaborado por: Santiago Constante, Andrés Yépez.

REQUERIMIENTOS MÍNIMOS	DESCRIPCIÓN
Sensor Kinect V2	Se necesita el dispositivo con sus respectivos adaptadores para computadoras.
Memoria Ram 4Gb	Se necesita 4BG de RAM para almacenar la información que emite el sensor Kinect v2
Directx 11	Presenta un conjunto de Drivers que permiten la compatibilidad de ciertos dispositivos como el Sensor Kinect v2
Puerto Usb 3.0	El Sensor Kinect v2 transmite grandes cantidades de información, por este motivo necesita puertos USB 3.0
Sistema Operativo Windows 8.1	El Sensor Kinect v2 tiene compatibilidad con los sistemas operativos 8.1 y las siguientes versiones.
Procesador De 64 Bits, Doble Núcleo Físico De 3,1 Ghz	Se necesita un procesador con las características mínimas mencionadas para que el sistema pueda funcionar.
SDK Kinect V2	Se necesita el SDK para poder utilizar el Kinect v2 en la computadora
Visual Studio 2012	Programa que posee las herramientas necesarias para poder programar el código del Sistema Traductor.
.Net Framework 4.5	Extensión necesaria para complementar la programación en Visual Studio.

Tomando en consideración que se cuenta con todos los elementos mencionados, la elaboración del sistema es factible desde el punto de vista tecnológico.

2.1.2.3 Factibilidad Económica

Dentro de la factibilidad económica se establece los costos en valor monetario de los componentes adquiridos para el desarrollo del proyecto. En la siguiente tabla se visualizan los costos de los componentes necesarios para el desarrollo del proyecto:

Tabla 3. Costo de Recursos
Elaborado por: Santiago Constante, Andrés Yépez

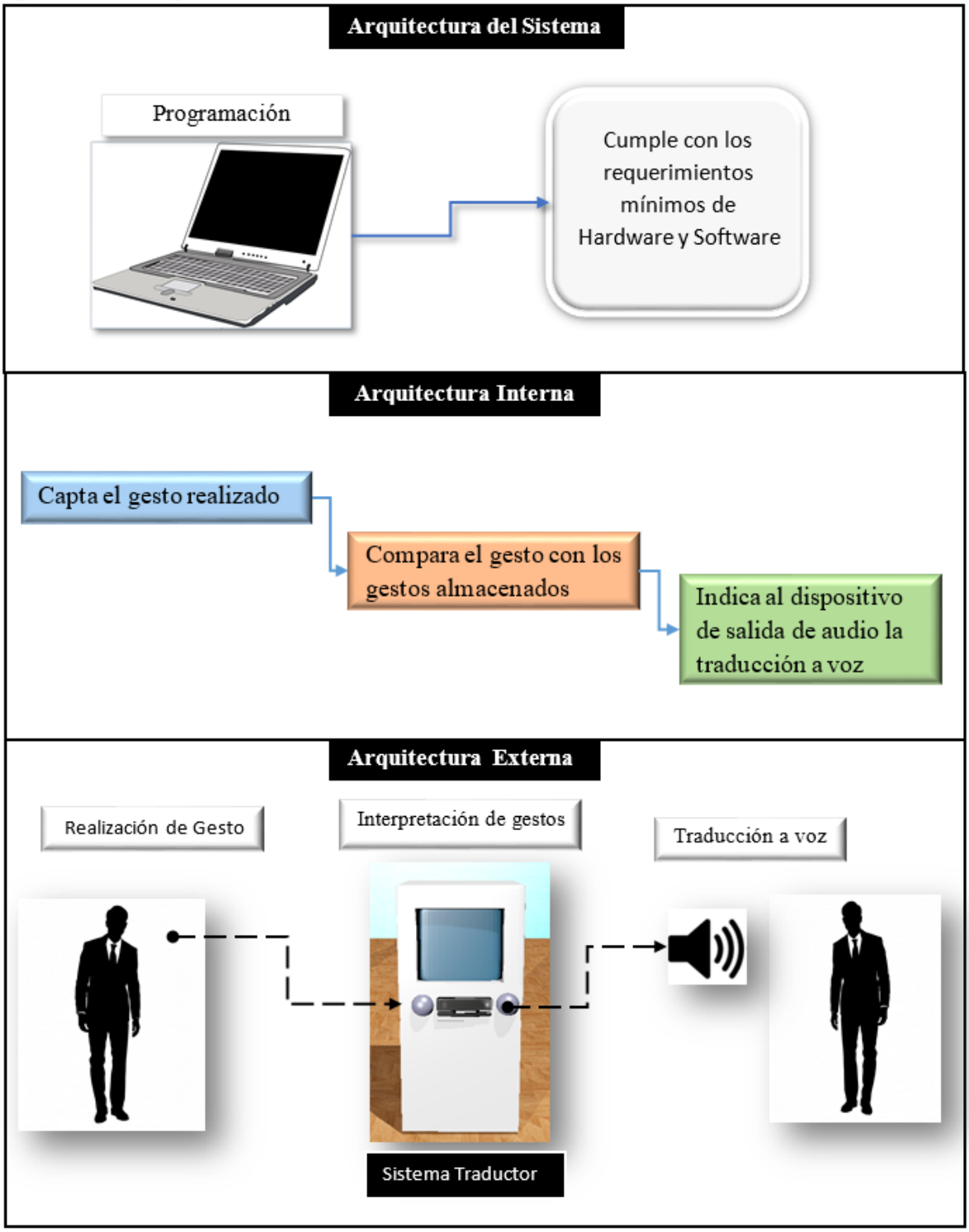
RECURSOS					
Equipos		**Materiales**		**Otros gastos**	
Descripción	Valor Unitario	Descripción	Valor Unitario	Descripción	Valor Unitario
NUI Kinect v2.0	200	Soporte Monitor	79	Transporte	95
CPU Prototipo	350	Botón de encendido	5	Extras	100
Adaptador Kinect Pc	50	Cableado	25		
Monitor	120	Materiales para la estructura física	120		
Periféricos	60				
Costo por recurso	680		195		195

Con los valores obtenidos se puede apreciar que la suma total de los costos es de 1170 dólares. Los gastos mencionados anteriormente fueron cubiertos por parte de los autores del presente trabajo.

2.2 Diseño

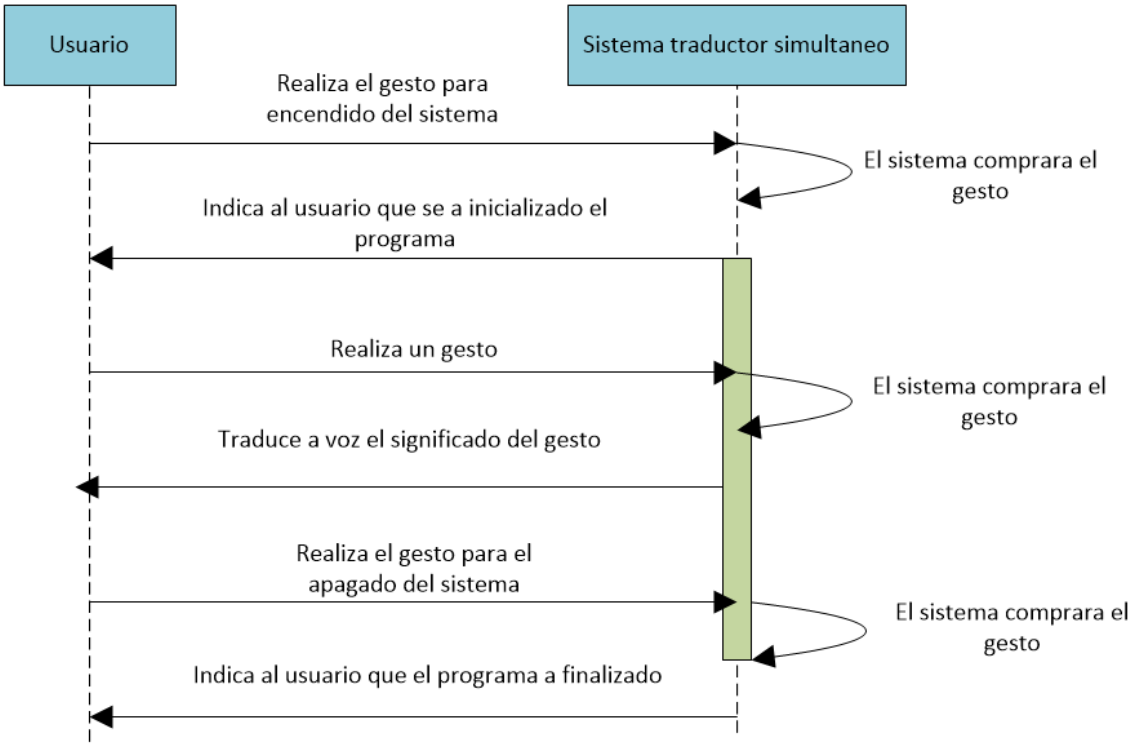
2.2.1 Esquema general de la solución técnica

Figura 10. Arquitectura del Sistema
Elaborado por: Santiago Constante, Andrés Yépez



2.2.1.1 Diagrama de Secuencia

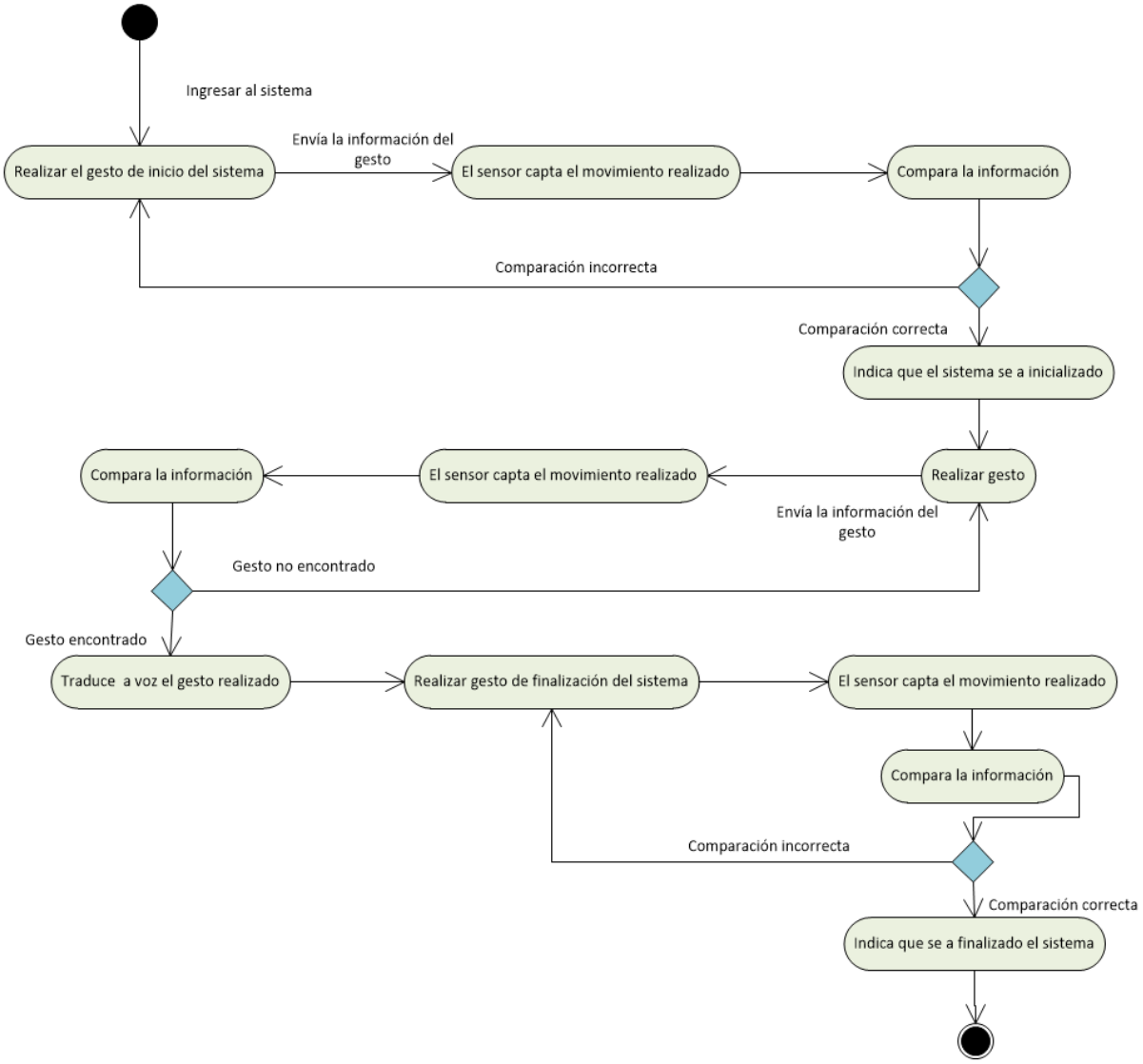
Figura 11. Diagrama de Secuencia
Elaborado por: Santiago Constante, Andrés Yépez



El diagrama de secuencia muestra la interacción entre el usuario y el traductor simultáneo, mostrando el proceso que se realiza desde que se inicializa el programa hasta que se lo apaga. Al momento de realizar la traducción el sistema capta la información de los movimientos del usuario para luego comprar sus movimientos con los gestos almacenados, si la persona realiza el gesto correctamente, el sistema traduce a voz lo que el gesto significa.

2.2.1.2 Diagrama de Actividades

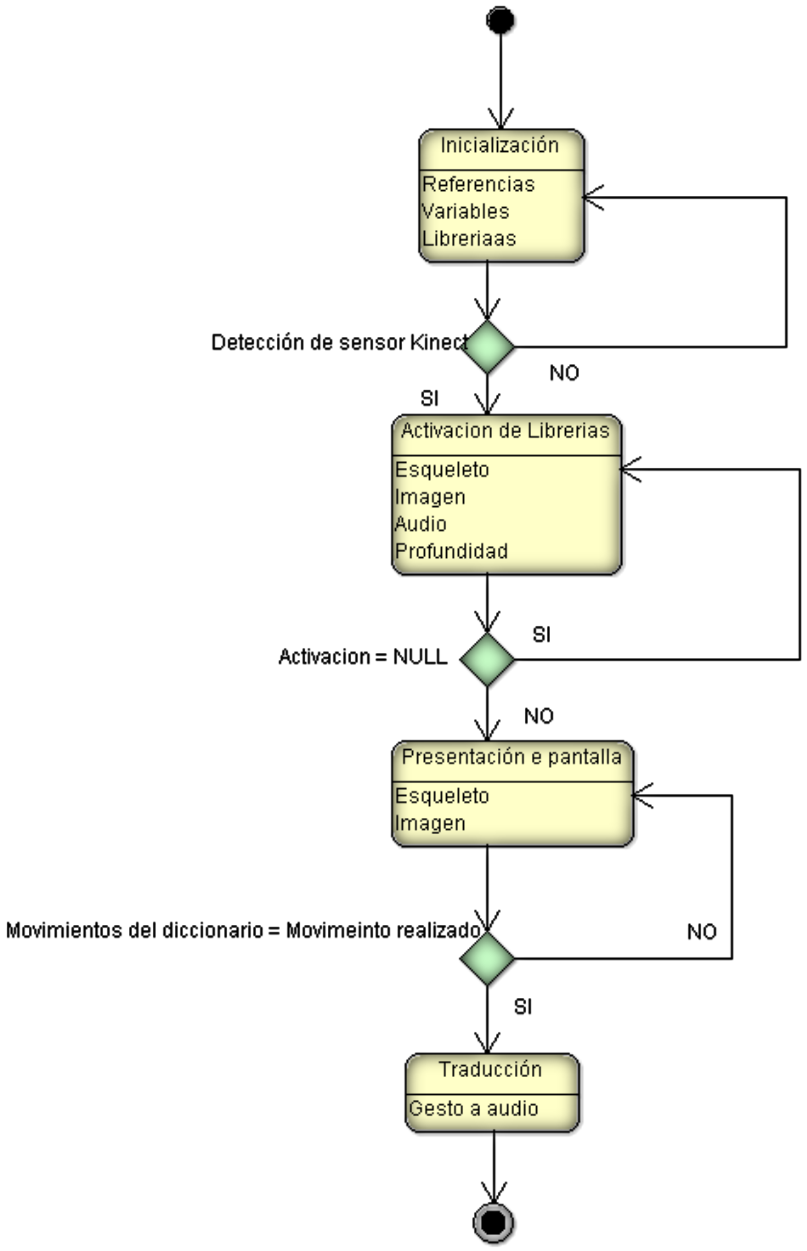
Figura 12. Diagrama de Actividades
Elaborado por: Santiago Constante, Andrés Yépez



En el diagrama de actividades se puede apreciar el proceso que el usuario sigue para utilizar el sistema traductor, indicando así las acciones que realiza el usuario y la respuesta que emite el sistema frente a la información obtenida. Cada vez que realiza un movimiento, el sistema actúa dependiendo si existe o no el gesto realizado.

2.2.1.3 Diagrama de Estados

Figura 13. Diagrama de estados
Elaborado por: Santiago Constante; Andrés Yépez



2.2.1.4 Diagrama de Casos de uso

Figura 14. Diagrama de Casos de uso Encendido del sistema
Elaborado por: Santiago Constante; Andrés Yépez

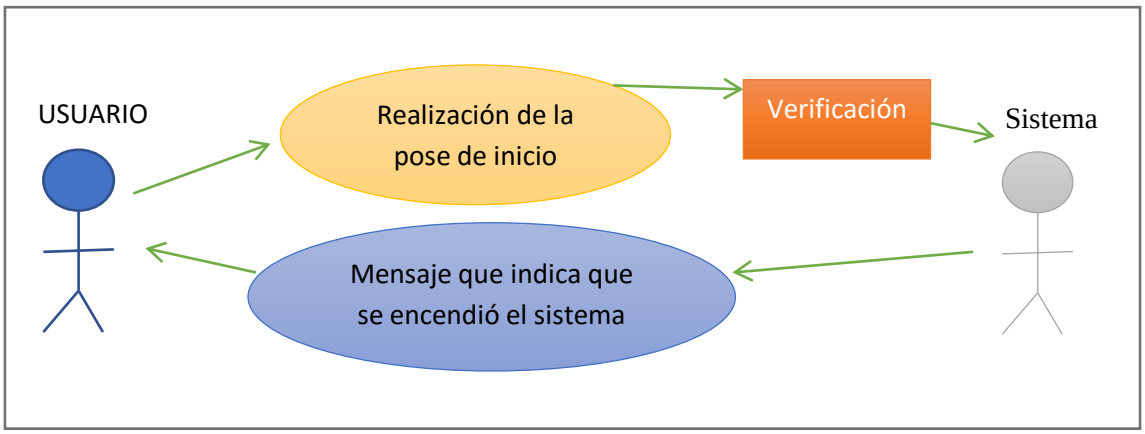


Tabla 4. Caso de uso Encendido del Sistema
Elaborado por: Santiago Constante, Andrés Yépez

Título:	Encendido del Sistema
Actor Principal:	Usuario
Actor Secundarios:	Sistema
Descripción	Iniciar el sistema mediante el gesto de encendido
Pre Condiciones:	• El programa debe estar ejecutado. • El Usuario debe estar colocado a la distancia indicada.
Frecuencia	Cada vez que un usuario quiera utilizar el traductor y se encuentre apagado
Escenario:	Curso Normal del evento: • Usuario: Realizar gesto de encendido. • Sistema: Captar el gesto mediante el sensor • Sistema: Comparar el gesto del usuario con el de encendido. • Sistema: Emitir mensaje de voz indicando que se a inicializado el programa. Curso Alternativo del evento: • Usuario: Realizar un gesto diferente al de encendido. • Sistema: Captar el gesto mediante el sensor

Título:	Encendido del Sistema
Escenario:	• Sistema: Comparar el gesto del usuario con el de encendido. • Sistema: Esperar nuevamente a la realización de un nuevo gesto. Finalización del evento: • El usuario realiza correctamente el gesto de encendido. • El usuario sale del rango de visión del Kinect

Figura 15. Diagrama de Caso de uso traducción
Elaborado por: Santiago Constante; Andrés Yépez

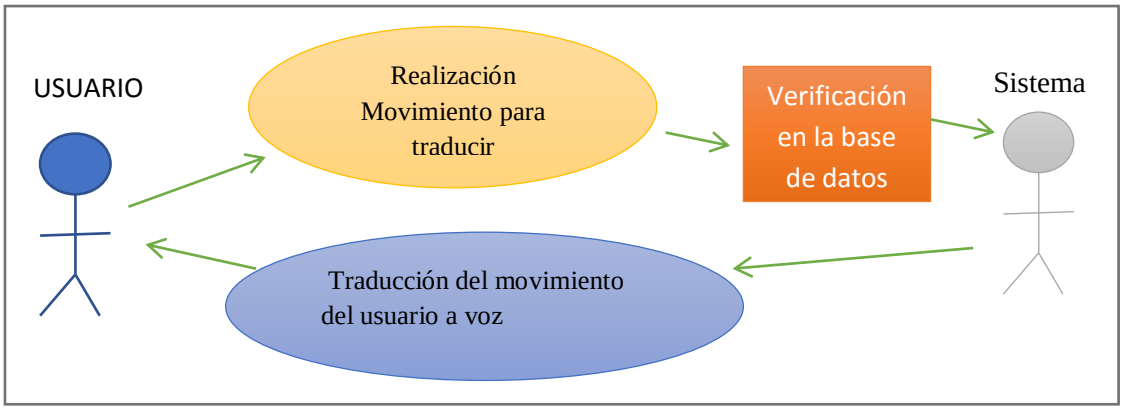


Tabla 5. Caso de uso Traducción de Gestos.
Elaborado por: Santiago Constante, Andrés Yépez.

Título:	Traducción de gestos
Actor Principal:	Usuario
Actor Secundarios:	Sistema
Descripción	Realizar un gesto para su traducción a voz
Pre Condiciones:	• El programa debe estar Inicializado. • El Usuario debe estar colocado a la distancia indicada.
Frecuencia	Cada vez que el usuario quiera traducir un gesto a voz
Escenario: **Escenario:**	Curso Normal del evento: • Usuario: Realizar un gesto. • Sistema: Captar el gesto mediante el sensor

Título:	Traducción de gestos
	• Sistema: Comparar el gesto realizado con los almacenados. • Sistema: Emitir mensaje de voz indicando el significado del gesto Curso Alternativo del evento: • Usuario: Realizar un gesto que no esté almacenado en el sistema. • Sistema: Captar el gesto mediante el sensor • Sistema: Comparar el gesto realizado con los almacenados. • Sistema: Esperar nuevamente a la realización de un nuevo gesto. Finalización del evento: • El usuario realiza correctamente el gesto. • El usuario sale del rango de visión del Kinect

CU03: Apagado del sistema

Figura 16. Diagrama de casos de uso finalización
Elaborado por: Santiago Constante; Andrés Yépez

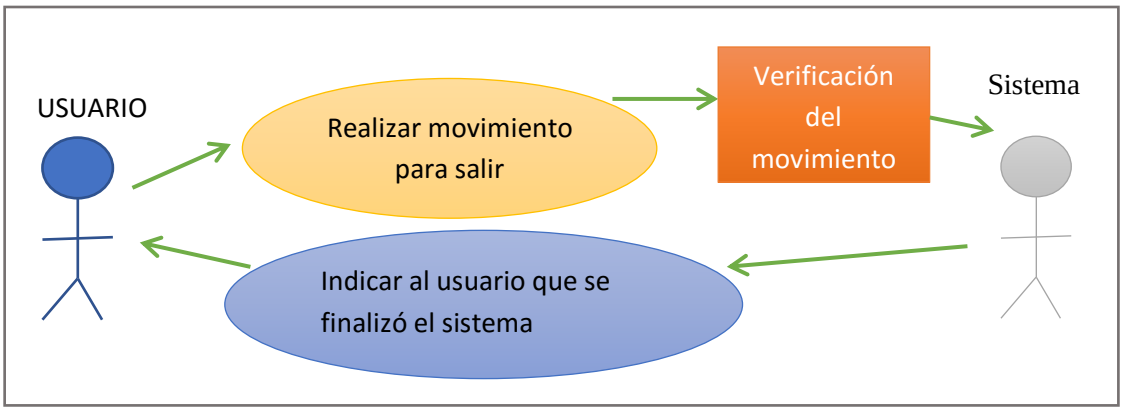
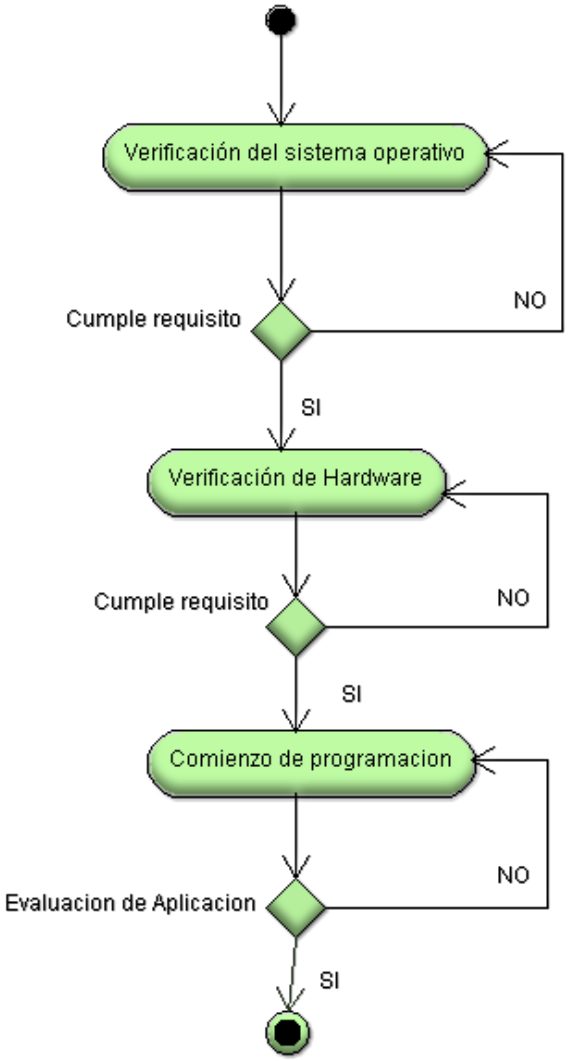


Tabla 6. Caso de uso Apagado del Sistema.
Elaborado por: Santiago Constante, Andrés Yépez.

Título:	Apagado del sistema
Actor Principal:	Usuario
Actor Secundarios:	Sistema
Descripción	Finalizar el programa mediante el gesto de apagado.
Pre Condiciones:	• El programa debe estar Inicializado. • El Usuario debe estar colocado a la distancia indicada.
Frecuencia	Cada vez que el usuario quiera apagar el sistema
Escenario:	Curso Normal del evento: • Usuario: Realizar el gesto de apagado. • Sistema: Captar el gesto mediante el sensor • Sistema: Comparar el gesto realizado con el gesto de apagado. • Sistema: Emitir mensaje de voz indicando que se ha finalizado el programa. Curso Alternativo del evento: • Usuario: Realizar un gesto diferente al de apagado. • Sistema: Captar el gesto mediante el sensor • Sistema: Comparar el gesto realizado con el de apagado. • Sistema: Esperar nuevamente a la realización de un nuevo gesto. Finalización del evento: • El usuario realiza correctamente el gesto. • El usuario sale del rango de visión del Kinect

2.2.1.5 Diagrama de flujo de plataforma

Figura 17. Diagrama de flujo
Elaborado por: Santiago Constante; Andrés Yépez



CAPÍTULO III

RESULTADOS

3.1 Construcción

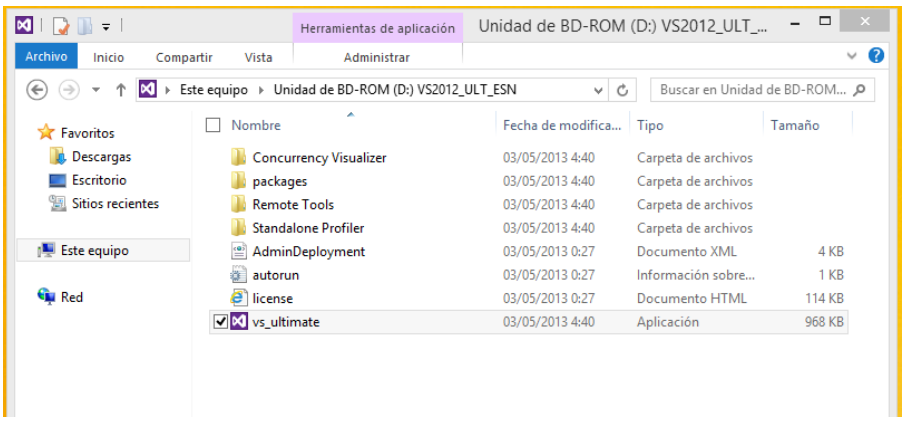
3.1.1 Instalación de software

Para poder implementar el sistema traductor primeramente se debe contar con los requisitos mínimos. Estos son:

3.1.1.1 Pasos para la instalación de Visual Studio 2012

1. Para iniciar el proceso de instalación primero se presiona doble clic sobre el icono de la aplicación del Visual Studio, en la siguiente captura se muestra seleccionado el icono.

Figura 18. Ubicación del instalador Visual Studio 2012
Elaborado por: Santiago Constante; Andrés Yépez



2. Luego de abrir el instalador aparecerá el recuadro siguiente en donde se debe especificar una ubicación para almacenar la información del programa. Se recomienda dejar la ubicación por defecto. Se debe aceptar los términos de condiciones de uso para poder continuar con la instalación.

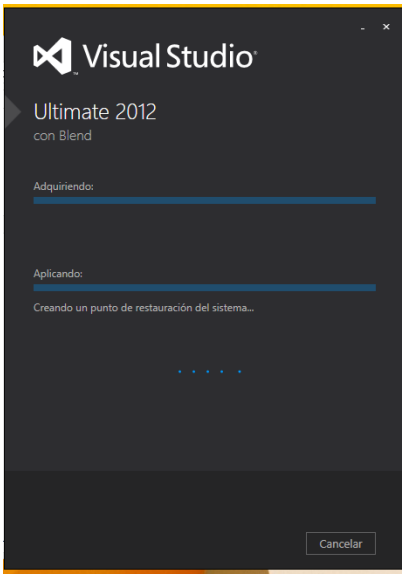
Figura 19. Selección de la ubicación en donde será almacenado el programa
Elaborado por: Santiago Constante; Andrés Yépez

3. Posteriormente el asistente de instalación muestra las características que se van a instalar. Se selecciona todas las características y se presiona el botón Instalar.

Figura 20. Selección de características a instalar.
Elaborado por: Santiago Constante; Andrés Yépez

4. Luego de confirmar las características comenzara el proceso de instalación.

Figura 21. Proceso de instalación de la aplicación.
Elaborado por: Santiago Constante; Andrés Yépez



5. Una vez finalizada la instalación saldrá el siguiente mensaje, donde se presiona el botón INICIAR para cerrar el asistente de instalación y empezar a utilizar el programa.

Figura 22. Ventana que se muestra después de acabar la instalación del programa.
Elaborado por: Santiago Constante; Andrés Yépez

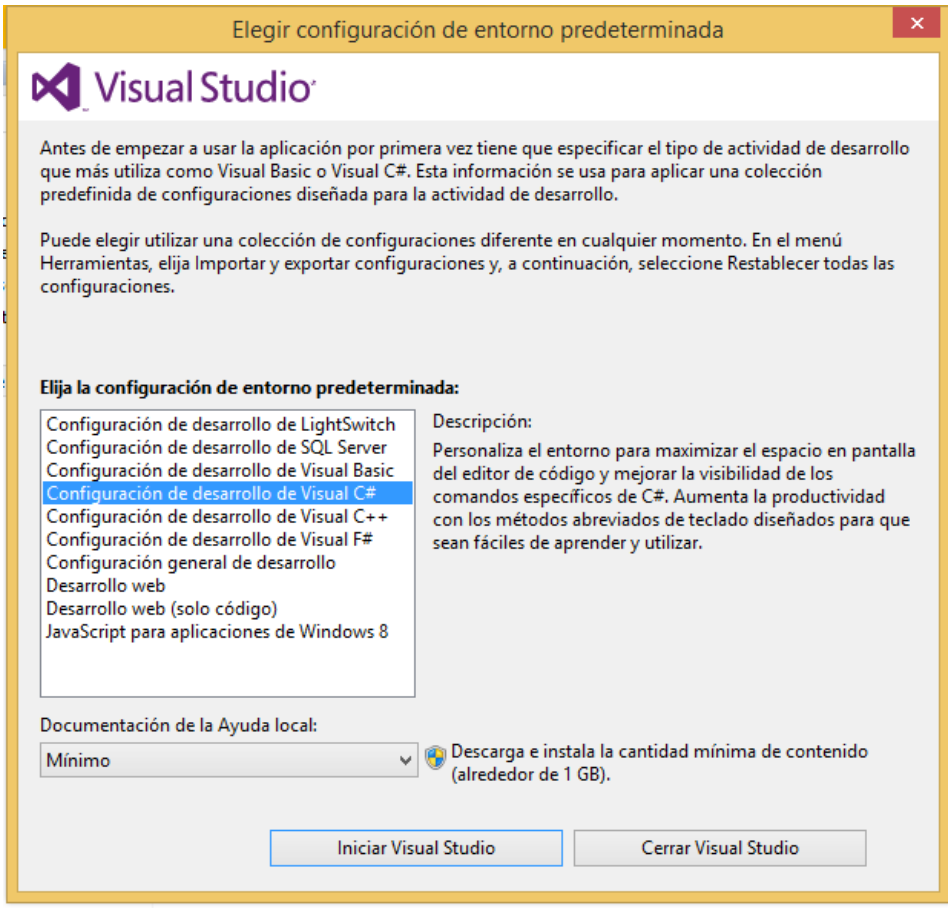
6. Se coloca la clave del producto para poder continuar. En caso de no tener la clave del producto, el software instalado pasara a ser tomado en cuenta como versión de prueba. La versión prueba dura 30 días.

Figura 23. Solicitud de la clave de instalación.
Elaborado por: Santiago Constante; Andrés Yépez

7. Se establece el entorno en el que se va a trabajar. Para este caso se selecciona el entorno "Configuración de desarrollo de Visual C#". La duración de este proceso dependerá de cuando contenido se desee descargar.

8. Por defecto se descarga los requerimientos mínimos, esto es suficiente para el desarrollo del proyecto, sin embargo si se desea que el entorno cuente con características más completas se puede cambiar la configuración presionando en la lista desplegable que se encuentra en la palabra "Mínimo"

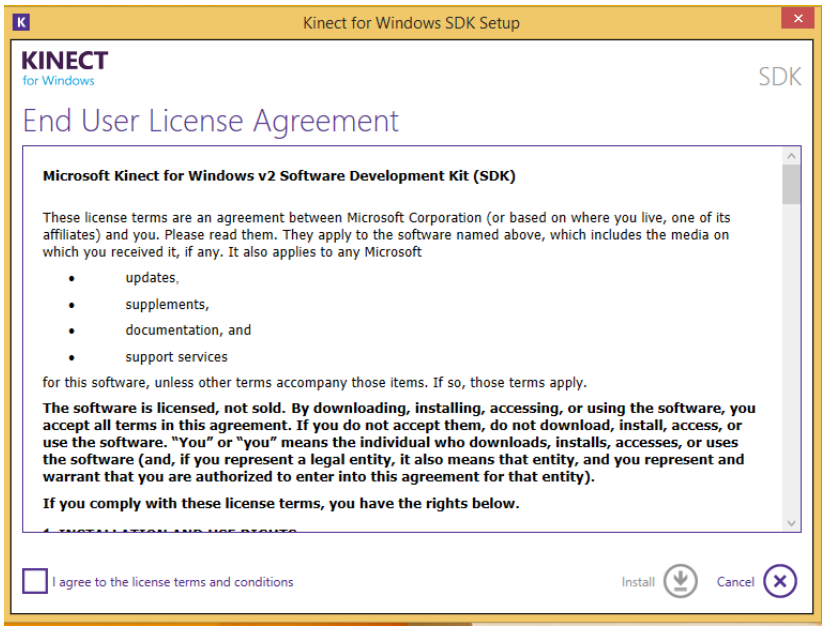
Figura 24. Inicialización del programa en un entorno c#.
Elaborado por: Santiago Constante; Andrés Yépez



3.1.1.2 SDK versión 2.

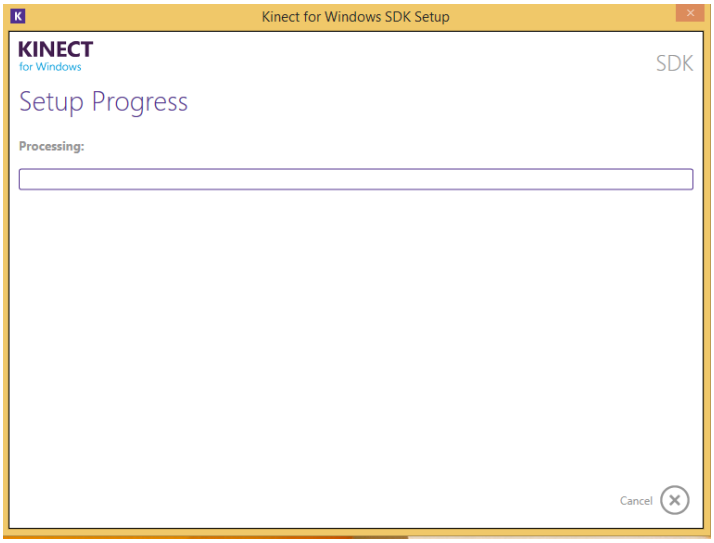
1. Para la instalación del SDK v2 primero se presiona el icono de instalación del SDK v2.
2. Aparecerá una ventana en donde se debe aceptar los términos y condiciones de uso para poder continuar. Se presiona el botón instalar luego de aceptar las condiciones.

Figura 25. Términos y condiciones de uso de SDK v2.
Elaborado por: Santiago Constante; Andrés Yépez



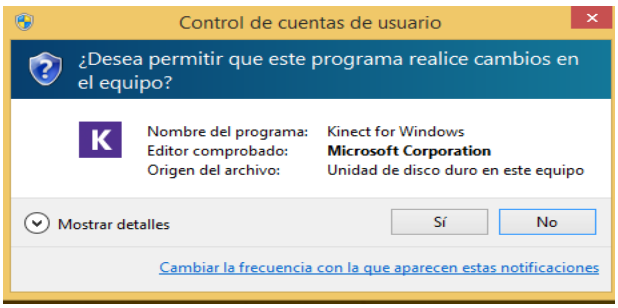
3. Aparecerá una ventana que muestra el avance de la instalación. La duración de este proceso es relativamente corta ya que el componente a ser instalado no contiene gran peso.

Figura 26. Proceso de instalación del SDK v2
Elaborado por: Santiago Constante; Andrés Yépez



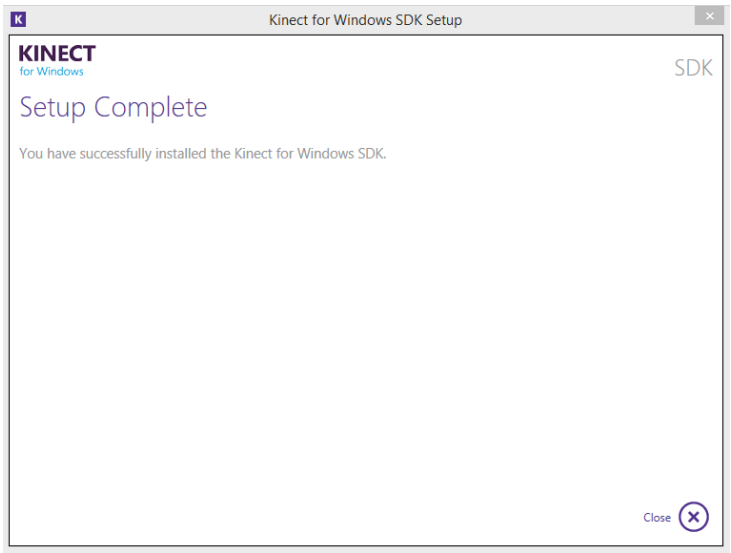
4. En el proceso de instalación aparece una ventana que solicita permiso para instalar el programa, se presiona el botón “Sí” para continuar.

Figura 27. Solicitud para que el programa realice cambios en el equipo.
Elaborado por: Santiago Constante; Andrés Yépez



5. Al finalizar la instalación aparece un mensaje indicando que se ha instalado completamente. Se presiona el botón “Close”.

Figura 28. Finalización de la instalación del SKD V2
Elaborado por: Santiago Constante; Andrés Yépez



3.1.2.1 Pasos para la comprobación de los requerimientos mínimos.

1. Una vez instalado el SKD se procede a abrir la aplicación.

2. Presionar la tecla de Windows.

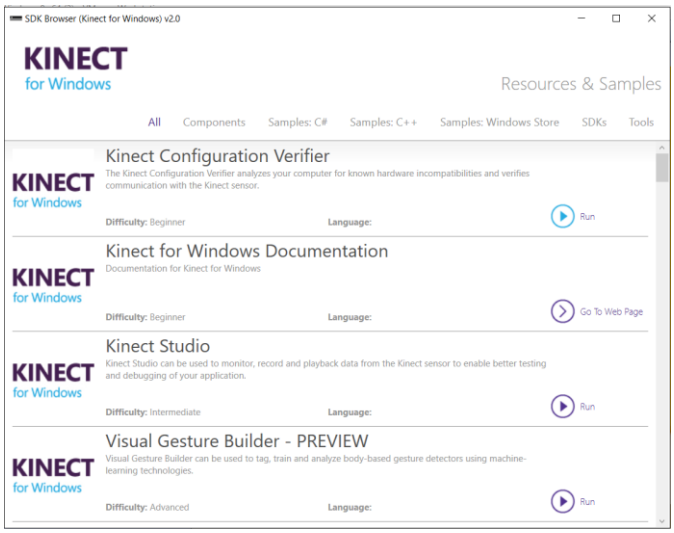
3. Escribir la palabra “SDK”. Al momento de escribir la frase, aparecerá al lado derecho de la pantalla los programas que tienen la palabra “SDK”, se selecciona el programa “SDK Browser v2.0 (Kinect for Windows)”.

Figura 29. Búsqueda de la aplicación SDK Kinect v2.
Elaborado por: Santiago Constante; Andrés Yépez

4. Dar clic en la aplicación “SDK Browser v2.0 (Kinect for Windows)”.

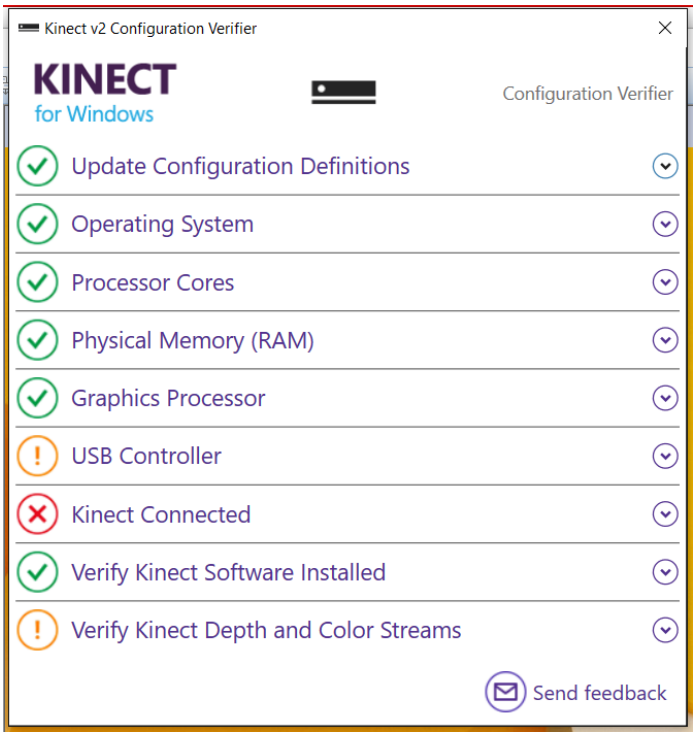
5. Aparecerá la siguiente ventana

Figura 30. Aplicaciones del SDK.
Elaborado por: Santiago Constante; Andrés Yépez



6. Presionar el botón “Run” que se encuentra en la opción “Kinect Configuration Verifier.”
7. Aparecerá la siguiente ventana la cual indicara si la computadora contiene los requerimientos mínimos para usar el Kinect v2.

Figura 31. Verificación de requerimientos para el uso de Kinect V2.
Elaborado por: Santiago Constante; Andrés Yépez



8. Los Iconos representan los siguiente:
    a. Icono verde: Cumple con los requerimientos mínimos.
    b. Icono amarillo: No cumple con los requerimientos mínimos pero puede utilizarse para el Kinect v2.
    c. Icono rojo: No cumple con los requerimientos y no se puede utilizar el Kinect.

3.1.2 Diseño de Interfaz

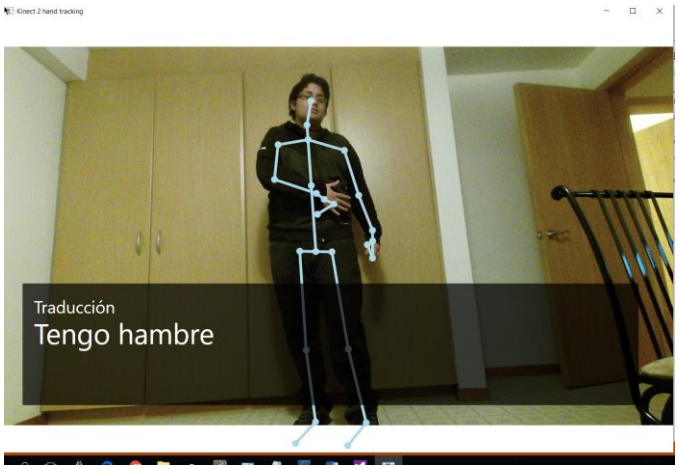
Una vez instalado el software requerido se inició con el desarrollo del sistema en sí. El primer paso a seguir era el diseño de la interfaz de usuario que brinda interacción entre la persona y el sistema a través del sensor.

La interfaz refleja los movimientos utilizando un cuadro de tipo “canvas” que interactúa directamente con el sensor Kinect v2 proporcionándole información sobre las tramas detectadas.

Figura 32. Interfaz gráfica del sistema sin ejecutar.
Elaborado por: Santiago Constante, Andrés Yépez.

Además, se agregó una barra donde también se traducen a lenguaje escrito los movimientos en la pantalla. Con esta nueva característica así como volviendo al objeto canvas más grande, la interfaz se vuelve más espaciosa permitiendo una imagen más amplia en comparación con el prototipo.

Figura 33. Interfaz gráfica del sistema al momento de ejecutar.
Elaborado por: Santiago Constante, Andrés Yépez.



3.1.3 Programación de los algoritmos

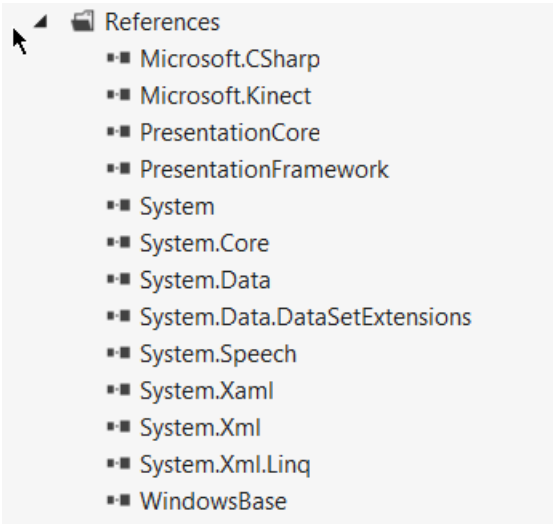
3.1.3.1 Declaración de las Librerías e Inclusión de Referencias

Para desarrollar el sistema primero se deben incluir ciertas librerías y referencias que permiten poder utilizar algunas de las funciones principales para comunicarse con el sensor. Incluyendo las librerías que permiten la utilización del lector de texto a voz de Windows, las librerías incluidas fueron:

```
using Microsoft.Kinect;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Speech.Synthesis;
```

Además se debieron incluir algunas referencias como la referencia Microsoft.Kinect (la cual es tomada del kit de herramientas SDK) para el correcto funcionamiento de los métodos de algunas librerías utilizadas. Las referencias agregadas fueron las siguientes:

Figura 34. Referencias Utilizadas
Elaborado por: Santiago Constante, Andrés Yépez



3.1.3.2 Programación de la clase extensiones

Esta clase fue creada con el fin de gestionar la interacción del sensor con el usuario. Por un lado las tramas recibidas tienen un tamaño y un formato que no es aceptado por los monitores o pantallas de una computadora. Por lo tanto, deben ser convertidas al formato BitMap para que puedan ser proyectadas por el monitor del sistema, a esta función la encerramos en una región llamada Camera. La clase también se encarga de indicarle al sistema cuales son los puntos del cuerpo que deben ser procesados. A esta función la localizamos en la región Body. Finalmente, la clase se encarga de crear el esqueleto para ser visualizado en la interfaz de usuario lo que permite verificar que movimiento se está realizando, esto se encuentra en la región Drawing. Sin abrir las regiones, el código se mostraría así:

```
namespace SistemaTraductorSimultaneo
{

    public static class Extensions
    {
        Camera

        Body

        Drawing
    }
}
```

La región Camera, contiene los siguientes algoritmos:

```
#region Camera

public static ImageSource ToBitmap(this ColorFrame frame)
{
    int width = frame.FrameDescription.Width;
    int height = frame.FrameDescription.Height;
    PixelFormat format = PixelFormats.Bgr32;

    byte[] pixels = new byte[width * height * ((format.BitsPerPixel + 7) / 8)];

    if (frame.RawColorImageFormat == ColorImageFormat.Bgra)
    {
        frame.CopyRawFrameDataToArray(pixels);
    }
    else
    {
        frame.CopyConvertedFrameDataToArray(pixels, ColorImageFormat.Bgra);
    }

    int stride = width * format.BitsPerPixel / 8;

    return BitmapSource.Create(width, height, 96, 96, format, null, pixels, stride);
}
```

La región Body está compuesta del siguiente código:

```
#region Body

public static Point Scale(this Joint joint, CoordinateMapper mapper)
{
    Point point = new Point();

    ColorSpacePoint colorPoint = mapper.MapCameraPointToColorSpace(joint.Position);
    point.X = float.IsInfinity(colorPoint.X) ? 0.0 : colorPoint.X;
    point.Y = float.IsInfinity(colorPoint.Y) ? 0.0 : colorPoint.Y;

    return point;
}

#endregion
```

Y la última región, Drawing, está compuesta de los siguientes algoritmos:

```
#region Drawing

public static void DrawSkeleton(this Canvas canvas, Body body, CoordinateMapper mapper)
{
    if (body == null) return;

    foreach (Joint joint in body.Joints.Values)
    {
        canvas.DrawPoint(joint, mapper);
    }

    canvas.DrawLine(body.Joints[JointType.Head], body.Joints[JointType.Neck], mapper);
    canvas.DrawLine(body.Joints[JointType.Neck], body.Joints[JointType.SpineShoulder], mapper);
    canvas.DrawLine(body.Joints[JointType.SpineShoulder], body.Joints[JointType.ShoulderLeft], mapper);
    canvas.DrawLine(body.Joints[JointType.SpineShoulder], body.Joints[JointType.ShoulderRight], mapper);
    canvas.DrawLine(body.Joints[JointType.SpineShoulder], body.Joints[JointType.SpineMid], mapper);
    canvas.DrawLine(body.Joints[JointType.ShoulderLeft], body.Joints[JointType.ElbowLeft], mapper);
    canvas.DrawLine(body.Joints[JointType.ShoulderRight], body.Joints[JointType.ElbowRight], mapper);
    canvas.DrawLine(body.Joints[JointType.ElbowLeft], body.Joints[JointType.WristLeft], mapper);
    canvas.DrawLine(body.Joints[JointType.ElbowRight], body.Joints[JointType.WristRight], mapper);
    canvas.DrawLine(body.Joints[JointType.WristLeft], body.Joints[JointType.HandLeft], mapper);
    canvas.DrawLine(body.Joints[JointType.WristRight], body.Joints[JointType.HandRight], mapper);
    canvas.DrawLine(body.Joints[JointType.HandLeft], body.Joints[JointType.HandTipLeft], mapper);
    canvas.DrawLine(body.Joints[JointType.HandRight], body.Joints[JointType.HandTipRight], mapper);
    canvas.DrawLine(body.Joints[JointType.HandTipLeft], body.Joints[JointType.ThumbLeft], mapper);
    canvas.DrawLine(body.Joints[JointType.HandTipRight], body.Joints[JointType.ThumbRight], mapper);
    canvas.DrawLine(body.Joints[JointType.SpineMid], body.Joints[JointType.SpineBase], mapper);
    canvas.DrawLine(body.Joints[JointType.SpineBase], body.Joints[JointType.HipLeft], mapper);
    canvas.DrawLine(body.Joints[JointType.SpineBase], body.Joints[JointType.HipRight], mapper);
    canvas.DrawLine(body.Joints[JointType.HipLeft], body.Joints[JointType.KneeLeft], mapper);
    canvas.DrawLine(body.Joints[JointType.HipRight], body.Joints[JointType.KneeRight], mapper);
    canvas.DrawLine(body.Joints[JointType.KneeLeft], body.Joints[JointType.AnkleLeft], mapper);
    canvas.DrawLine(body.Joints[JointType.KneeRight], body.Joints[JointType.AnkleRight], mapper);
    canvas.DrawLine(body.Joints[JointType.AnkleLeft], body.Joints[JointType.FootLeft], mapper);
    canvas.DrawLine(body.Joints[JointType.AnkleRight], body.Joints[JointType.FootRight], mapper);
}
```

```
public static void DrawPoint(this Canvas canvas, Joint joint, CoordinateMapper mapper)
{
    if (joint.TrackingState == TrackingState.NotTracked) return;

    Point point = joint.Scale(mapper);

    Ellipse ellipse = new Ellipse
    {
        Width = 20,
        Height = 20,
        Fill = new SolidColorBrush(Colors.LightBlue)
    };

    Canvas.SetLeft(ellipse, point.X - ellipse.Width / 2);
    Canvas.SetTop(ellipse, point.Y - ellipse.Height / 2);

    canvas.Children.Add(ellipse);
}
```

```
public static void DrawHand(this Canvas canvas, Joint hand, CoordinateMapper mapper)
{
    if (hand.TrackingState == TrackingState.NotTracked) return;

    Point point = hand.Scale(mapper);

    Ellipse ellipse = new Ellipse
    {
        Width = 100,
        Height = 100,
        Stroke = new SolidColorBrush(Colors.LightBlue),
        StrokeThickness = 4
    };

    Canvas.SetLeft(ellipse, point.X - ellipse.Width / 2);
    Canvas.SetTop(ellipse, point.Y - ellipse.Height / 2);

    canvas.Children.Add(ellipse);
}

public static void DrawThumb(this Canvas canvas, Joint thumb, CoordinateMapper mapper)
{
    if (thumb.TrackingState == TrackingState.NotTracked) return;

    Point point = thumb.Scale(mapper);

    Ellipse ellipse = new Ellipse
    {
        Width = 40,
        Height = 40,
        Fill = new SolidColorBrush(Colors.LightBlue),
        Opacity = 0.7
    };

    Canvas.SetLeft(ellipse, point.X - ellipse.Width / 2);
    Canvas.SetTop(ellipse, point.Y - ellipse.Height / 2);

    canvas.Children.Add(ellipse);
}


public static void DrawLine(this Canvas canvas, Joint first, Joint second, CoordinateMapper mapper)
{
    if (first.TrackingState == TrackingState.NotTracked || second.TrackingState == TrackingState.NotTracked) return;

    Point firstPoint = first.Scale(mapper);
    Point secondPoint = second.Scale(mapper);

    Line line = new Line
    {
        X1 = firstPoint.X,
        Y1 = firstPoint.Y,
        X2 = secondPoint.X,
        Y2 = secondPoint.Y,
        StrokeThickness = 8,
        Stroke = new SolidColorBrush(Colors.LightBlue)
    };

    canvas.Children.Add(line);
}

#endregion
}
```

3.1.3.3 Programación de la clase principal MainWindow y de la clase Activador

El código principal del sistema se encuentra en la clase principal MainWindow, la cual contiene los algoritmos de conexión con el sensor, las llamadas a los métodos de las clases

Extensiones y Activador, los algoritmos de los gestos almacenados, así como también los algoritmos de traducción a voz. Una vez más, esta clase está dividida por regiones, lo que permite su fácil manejo y lectura. El código sin abrir las regiones se ve de esta forma:

```
public partial class MainWindow : Window
{
    Members

    Constructor
    Eventos
    textohablado


}
}
```

La primera de estas regiones, la llamada Members, contiene objetos que van a funcionar para llamar a los métodos que le permiten al sistema interactuar con el sensor Kinect y procesar las tramas recibidas. La región simplemente se ve así:

```
#region Members

KinectSensor _sensor;
MultiSourceFrameReader _reader
IList<Body> _bodies;
int variable;

#endregion
```

La segunda de estas regiones es la región constructor, la que contiene el algoritmo del constructor de la clase MainWindow entre otros algoritmos esenciales como el objeto que va a llamar a las funciones de traducción de texto a voz, y la bandera de activación y desactivación del sistema. Esta región se ve así:

```
#region Constructor

public MainWindow()
{
    InitializeComponent();

}


SpeechSynthesizer synthesizer = new SpeechSynthesizer();
List<VoiceInfo> vocesinfo = new List<VoiceInfo>();
Activador ban = new Activador();
#endregion
```

La región Eventos es la región más grande del código y esto se debe a que esta región abarca todos los eventos que realiza el sistema, desde la conexión con el sensor, pasando por la selección de la voz hablada, la asignación de puntos a ser detectados y finalmente la creación y comparación de gestos a ser traducidos. Es por esto, que la región Eventos también está definida por dos subregiones, como se ve a continuación:

```
#region Eventos

Kinect

Traductor

                    }
                }
            }
        }
    }
}

#endregion
```

La región Kinect se encarga de todos los eventos relacionados directa y únicamente con el sensor como son la conexión, creación de puntos, visualización y creación del esqueleto.

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    _sensor = KinectSensor.GetDefault();

    if (_sensor != null)
    {
        _sensor.Open();

        _reader = _sensor.OpenMultiSourceFrameReader(FrameSourceTypes.Color | FrameSourceTypes.Depth |
            FrameSourceTypes.Infrared | FrameSourceTypes.Body);
        _reader.MultiSourceFrameArrived += Reader_MultiSourceFrameArrived;
    }
    foreach (InstalledVoice voice in synthesizer.GetInstalledVoices())
    {
        vocesinfo.Add(voice.VoiceInfo);
        Voz.Items.Add(voice.VoiceInfo.Name);
    }
    Voz.SelectedIndex = 2;
}


private void Window_Closed(object sender, EventArgs e)
{
    if (_reader != null)
    {
        _reader.Dispose();
    }

    if (_sensor != null)
    {
        _sensor.Close();
    }
}


void Reader_MultiSourceFrameArrived(object sender, MultiSourceFrameArrivedEventArgs e)
{
    var reference = e.FrameReference.AcquireFrame();

    // Color
    using (var frame = reference.ColorFrameReference.AcquireFrame())
    {
        if (frame != null)
        {
            camera.Source = frame.ToBitmap();
        }
    }

    // Body
    using (var frame = reference.BodyFrameReference.AcquireFrame())
    {
        if (frame != null)
        {
            canvas.Children.Clear();

            _bodies = new Body[frame.BodyFrameSource.BodyCount];

            frame.GetAndRefreshBodyData(_bodies);

            foreach (var body in _bodies)
            {
                if (body != null)
                {
                    if (body.IsTracked)
                    {
```

```
// Find the joints
Joint handRight = body.Joints[JointType.HandRight];
Joint thumbRight = body.Joints[JointType.ThumbRight];

Joint handLeft = body.Joints[JointType.HandLeft];
Joint thumbLeft = body.Joints[JointType.ThumbLeft];
AsignacionPuntos

//

  canvas.DrawSkeleton(body, _sensor.CoordinateMapper);

 #endregion
```

La región traductor es el corazón del sistema y es donde están programados los algoritmos de los gestos que van a ser comparados y traducidos. El sensor capta las tramas, compara con los algoritmos condicionales y si coinciden, el sistema traduce a través de su interfaz de salida el gesto traducido a voz. Esta región se ve de la siguiente manera:

```
#region Traductor

                    variable =ban.ValorM1();
                if (variable==0)
                {
                    if (manoder.Position.X > hombroder.Position.X &&
                        manoizq.Position.X < hombroizq.Position.X &&
                        manoder.Position.Y < hombroder.Position.Y
                        && manoizq.Position.Y < hombroizq.Position.Y &&
                        manoder.Position.Y > columna.Position.Y &&
                        manoizq.Position.Y > columna.Position.Y
                        && cabeza.Position.Z > hombroder.Position.Z)
                    {
                        tblRightHandState.Text = ("Bienvenidos al Sistema de Traducción Integrada de Señas a Voz");
                        BotonHablar_Click(null, null);
                        ban.valorP();
                       // variable = ban.ValorM1();

                        ////.Items.Add(tblRightHandState.Text);
                    }
                }
                if (variable == 1)
                {
                    Gestos


                }
                  #endregion
```

En donde si abrimos la subregión gestos encontramos todos los algoritmos de comparación de los que están compuestos los gestos, tomando como ejemplo el gesto tengo hambre los algoritmos están compuestos así:

```
#region  16 Tengo Hambre OK OK
int contadorHambre = 0;

if ( manoizq.Position.X > columna.Position.X && manoizq.Position.Y < hombrocen.Position.Y &&
    body.HandLeftState == HandState.Open && body.HandRightState==HandState.Closed)
{
    contadorHambre = 1;
}
else
{
    contadorHambre = 0;
}
if (contadorHambre == 1)
{

    tblRightHandState.Text = ("Tengo hambre");
    BotonHablar_Click(null, null);
    //.Items.Add(tblRightHandState.Text);

    contadorHambre = 0;

}
#endregion
```

Para el correcto funcionamiento de los algoritmos de apagado y encendido fue necesario crear una pequeña clase llamada Activador, que simplemente nos provee de ciertas funciones para poder modificar el valor de una bandera, la que nos servirá para utilizarla como condiciones de activación o desactivación de los gestos:

```
public partial class Activador {
      public int flag;
    public
        Activador() {
            flag = 0;
        }
        public void valorP() {
            flag = 1;
        }
        public void valorM() {
            flag = 0;
        }
        public int ValorM1() {
            return flag;
        }


}
```

Finalmente, la siguiente y última región de la clase principal MainWindow es la región texto hablado, la que contiene la función que permite seleccionar la voz que se va a escuchar a

través de las interfaces de salida y de llamar al método Speak que finalmente ejecuta la traducción final.

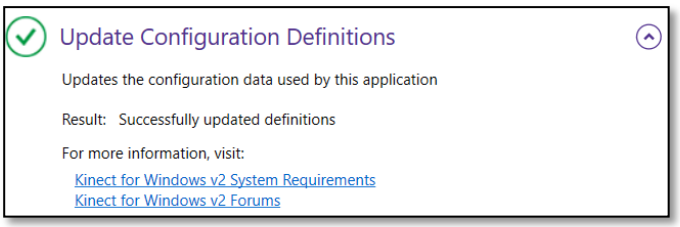
```
#region texto hablado
private void BotonHablar_Click(object sender, RoutedEventArgs e)
{
    int indice;
    indice = Voz.SelectedIndex;
    String nombre = vocesinfo.ElementAt(indice).Name;
    synthesizer.SelectVoice(nombre);
    synthesizer.Speak(tblRightHandState.Text);

}
#endregion
```

3.1.4 Verificación de Requerimientos mínimos.

3.1.4.1 Configuración actualizada.

Verifica la versión de SDK que se tiene instalada y si existen versiones que falten actualizar.

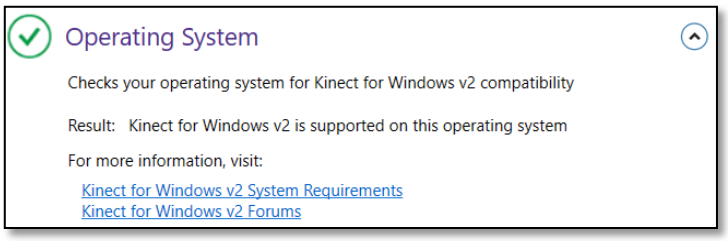
Figura 35. Verificación versión.
Elaborado por: Santiago Constante; Andrés Yépez



3.1.4.2 Sistema Operativo.

Verifica que el sistema operativo instalado sea: Windows 8.1 o versiones mas recientes.

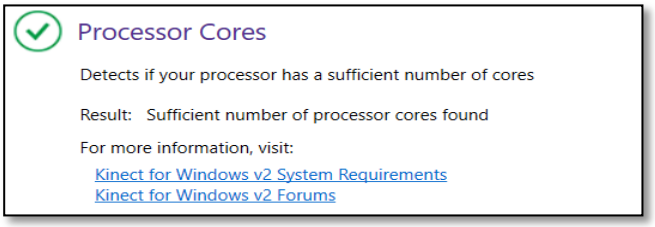
Figura 36. Verificación de Sistema Operativo.
Elaborado por: Santiago Constante; Andrés Yépez



3.1.4.3 Numero de núcleos del procesador.

Verifica si el procesador es apto para utilizar el Kinect V2. Como requerimiento mínimo se debe tener un procesador Dual-core 3.1 GHz. Este proceso es de suma importancia porque si no se cumple con este requerimiento el programa no funcionara correctamente. Esto se debe a que las aplicaciones realizadas con el equipo Kinect utilizan una gran cantidad de memoria para procesar la información.

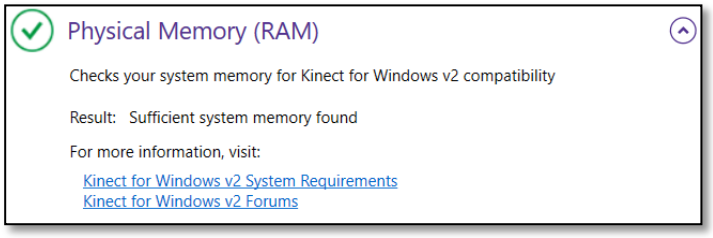
Figura 37. Verificación de procesador.
Elaborado por: Santiago Constante; Andrés Yépez



3.1.4.4 Memoria RAM

Verifica que la memoria RAM contenga mínimo 4 GB de capacidad.

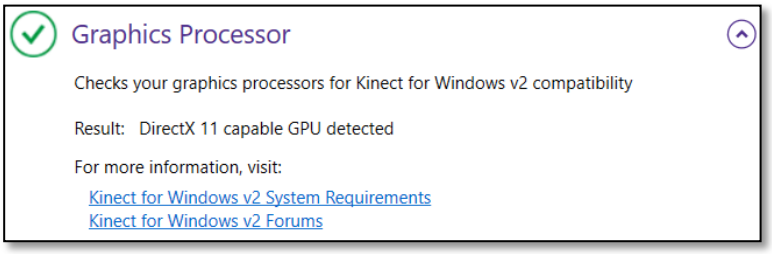
Figura 38. Verificación de Memoria RAM
Elaborado por: Santiago Constante; Andrés Yépez



3.1.4.5 Tarjeta Gráfica

Verifica que la tarjeta gráfica soporte DirectX 11.

Figura 39. Verificación de tarjeta gráfica.
Elaborado por: Santiago Constante; Andrés Yépez



3.1.4.6 Puerto USB

Verifica que la computadora tenga un puerto 3.0. Si no se cuenta con este tipo de puerto, la computadora no podrá reconocer al Kinect v2.

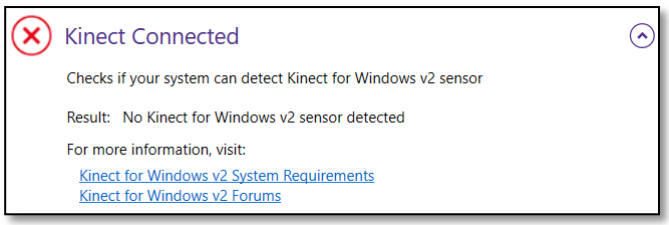
Figura 40. Verificación de puerto USB.
Elaborado por: Santiago Constante; Andrés Yépez

3.1.4.7 Kinect V2

Verifica que esté conectado a la computadora un Kinect V2. Cabe resaltar que se debe contar con un adaptador a puertos UBS de tal manera que se pueda conectar el Kinect.

Para el proceso de la programación no es necesario que esté conectado el Kinect, sin embargo si se desea realizar pruebas de funcionamiento se debe conectarlo para poder ver los resultados.

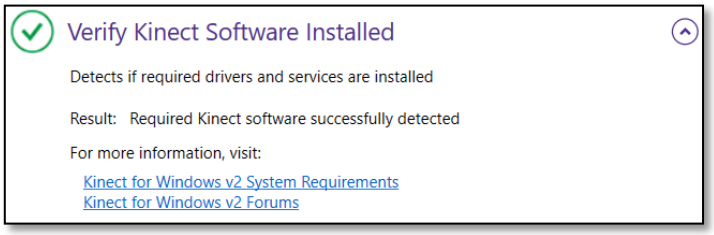
Figura 41. Verificación de requerimientos para el uso de Kinect V2.
Elaborado por: Santiago Constante; Andrés Yépez



3.1.4.8 Software de Kinect v2.

Verifica que estén instalados los Drivers necesarios para la utilización del Kinect v2. Si no cuenta con los Drivers necesarios el equipo Kinect no podrá enviar o recibir información de la computadora.

Figura 42. Verificación de drivers.
Elaborado por: Santiago Constante; Andrés Yépez

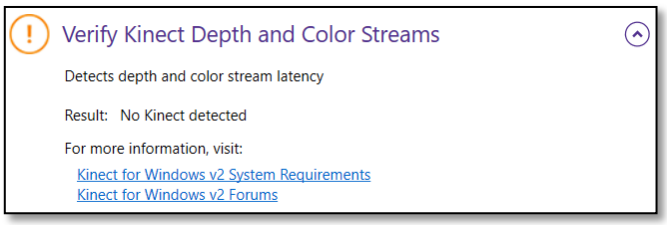


3.1.4.9 Color y profundidad.

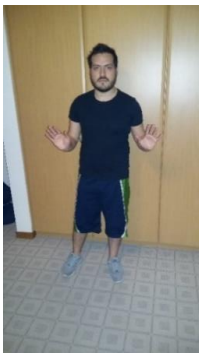
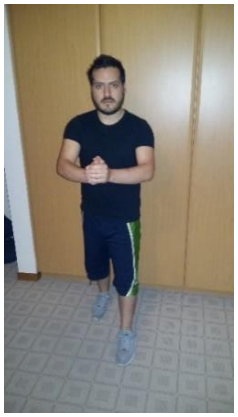
Verifica el color y la profundidad captada por el sensor del Kinect v2.

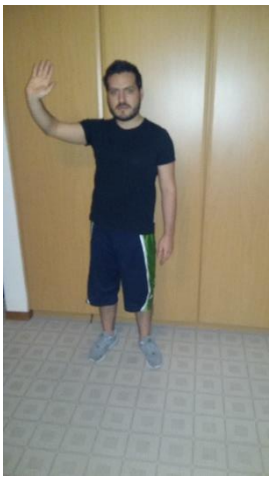
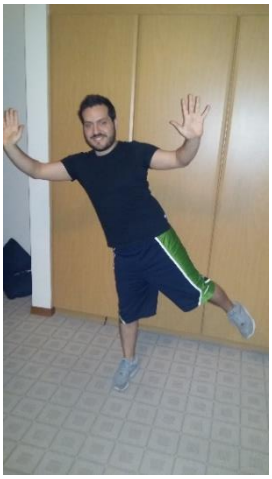
Para poder realizar la verificación debe estar conectado un Kinect v2.

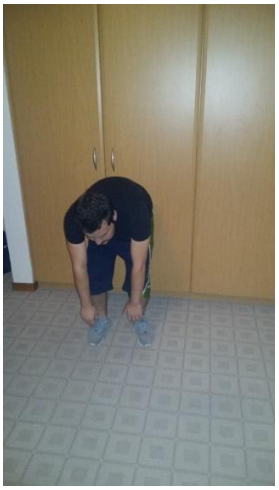
Figura 43. Verificación de profundidad y color.
Elaborado por: Santiago Constante; Andrés Yépez

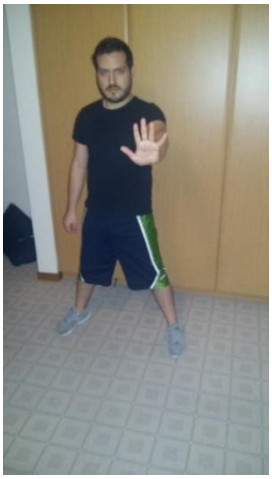


3.1.5 Diccionario de Gestos y Cómo se Realizan.

Nº	Gesto	Activación
1	Encendido del sistema	
2	Por Favor	

Nº	Gesto	Activación
3	Hola	
4	Estoy Emocionado	
5	Necesito ayuda, Sígueme	

Nº	Gesto	Activación
6	En este lugar	
7	Necesito Ejercitarme	
8	Estoy Agotado	

Nº	Gesto	Activación
9	Regresare Luego	
10	Espera un momento	
11	Detente, Habla más despacio	

Nº	Gesto	Activación
12	Estoy Asombrado	
13	No lo Recuerdo Bien	
14	Estoy Enojado	

Nº	Gesto	Activación
15	Tengo Frio	
16	Tengo Hambre	
17	Si, Correcto	

Nº	Gesto	Activación
18	No, Incorrecto	
19	Tengo Fiebre	
20	Donde	

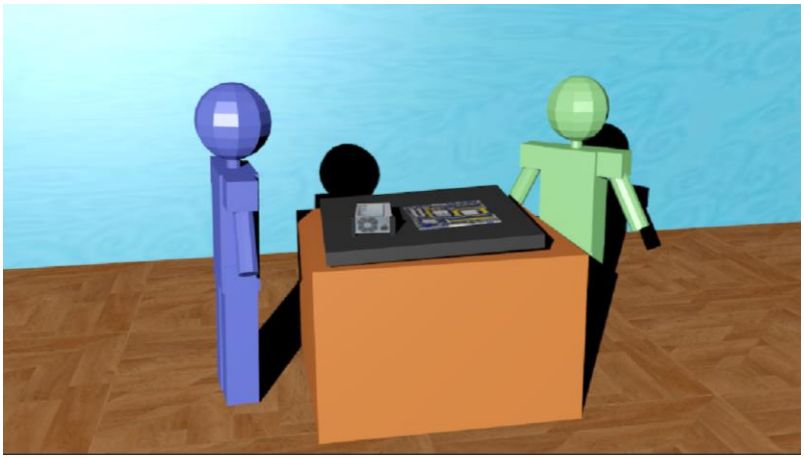
Nº	Gesto	Activación
21	Apagado del Sistema	

3.1.6 Construcción de la Estructura Física

Para poder construir la estructura física, primeramente se realizó una un gráfico base para proseguir con la elaboración, los pasos a seguir se muestran a continuación:

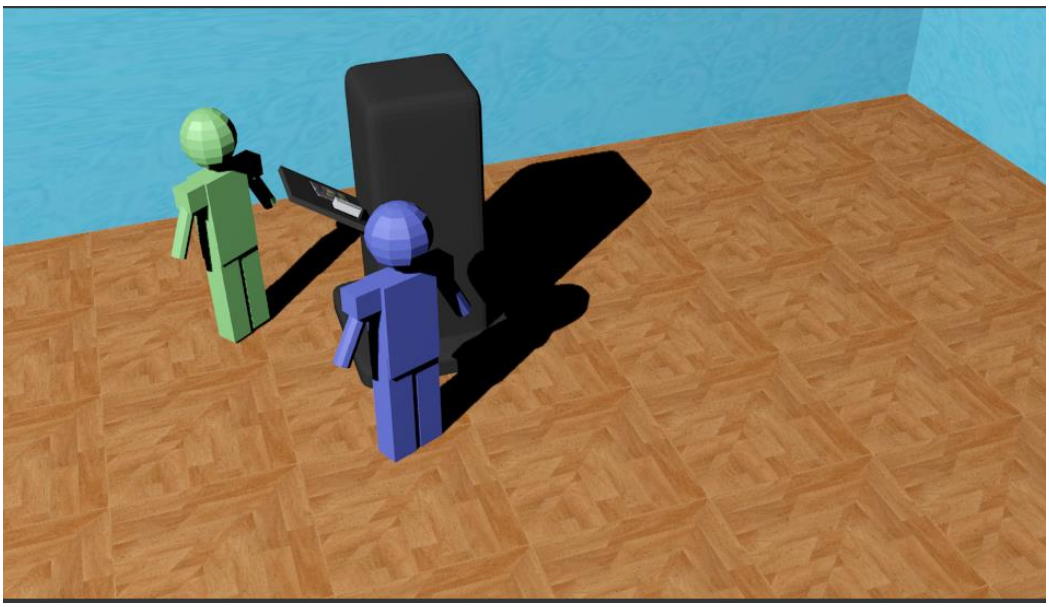
Primeramente se coloca los componentes internos de la computadora encima de una plataforma.

Figura 44. Ensamblaje placa madre y fuente de poder.
Elaborado por: Santiago Constante, Andrés Yépez.



Posteriormente se añade la plataforma a una estructura de madera en donde estará incluido el monitor, los parlantes y el Kinect v2.

Figura 45. Acoplamiento de placa base con el Case del sistema.
Elaborado por: Santiago Constante, Andrés Yépez.



Finalmente se obtiene la estructura con todos los elementos de hardware incluidos para el funcionamiento del sistema.

Figura 46. Resultado final.
Elaborado por: Santiago Constante, Andrés Yépez.

En base al modelo realizado se prosiguió con la construcción de la estructura, realizando algunas variaciones que fueron modificadas en proceso. Las imágenes del resultado final se pueden apreciar en los Anexos de este proyecto. (Ver Anexo 2).

3.2 Implementación

3.2.1 Pruebas de funcionamiento.

Las pruebas que se hicieron para comprobar el correcto funcionamiento del sistema se realizaron pruebas de caja blanca, para comprobar el funcionamiento interno del sistema, y pruebas de caja negra para el externo.

3.2.1.1 Pruebas de Caja Blanca:

La primera prueba que se realizó fue la del camino básico. Partiendo de esta prueba se verifico que el código permita traducir los gestos, es decir, se confirmó la posibilidad de realizar una entrada de datos y que esta pueda ser utilizada para el proceso de traducción.

Luego se realizó una prueba de bucles para verificar que no existan ciclos infinitos al momento de la traducción. Durante este proceso, se realizó correcciones de errores para que los ciclos comiencen y terminen correctamente.

3.2.1.2 Pruebas de Caja Negra

Para realizar la prueba de caja negra se fue comprobando gesto por gesto verificando que la traducción de cada uno se realice correctamente. Para poder registrar el funcionamiento de los gestos se utilizó la siguiente tabla.

Tabla 7. Pruebas de funcionalidad.
Elaborado por: Santiago Constante, Andrés Yépez.

Nº	Descripción	Requisitos	Resultado Esperado	Resultado
1	Encendido del sistema	El programa no debe estar inicializado	Mensaje de voz indicando que el sistema se ha encendido	OK
2	Gesto Gracias	El programa debe estar inicializado	Traducción a voz: Gracias	Ok
3	Gesto Por Favor	El programa debe estar inicializado	Traducción a voz: Por favor	Ok
4	Gesto Hola	El programa debe estar inicializado	Traducción a voz: Hola	OK
5	Gesto Estoy Feliz	El programa debe estar inicializado	Traducción a voz: Estoy Feliz	OK
6	Gesto Por Ahí	El programa debe estar inicializado	Traducción a voz:: Por Ahí	OK
7	Gesto En este Lugar	El programa debe estar inicializado	Traducción a voz: En este lugar	OK
8	Gesto Estoy Fuerte	El programa debe estar inicializado	Traducción a voz: Estoy fuerte	OK
9	Gesto Estoy Agotado	El programa debe estar inicializado	Traducción a voz:: Estoy Agotado	OK
10	Gesto estoy Confundido	El programa debe estar inicializado	Traducción a voz: Estoy Confundido	OK
11	Gesto Detente	El programa debe estar inicializado	Traducción a voz:: Detente no sigas	OK

Nº	Descripción	Requisitos	Resultado Esperado	Resultado
12	Gesto Estoy Sorprendido	El programa debe estar inicializado	Traducción a voz: Estoy Sorprendido	OK
13	Gesto Espera Un Minuto	El programa debe estar inicializado	Traducción a voz: Espera un minuto	OK
14	Gesto No Estoy Seguro	El programa debe estar inicializado	Traducción a voz: No Estoy Seguro	OK
15	Gesto Estoy Enojado	El programa debe estar inicializado	Traducción a voz: Estoy Enojado	OK
16	Gesto Tengo Hambre	El programa debe estar inicializado	Traducción a voz: Tengo Hambre	OK
17	Gesto de afirmación	El programa debe estar inicializado	Traducción a voz: Si, Correcto	OK
18	Gesto de negación	El programa debe estar inicializado	Traducción a voz: No, Incorrecto	OK
19	Gesto Tengo Frio	El programa debe estar inicializado	Traducción a voz: Tengo Frio	OK
20	Gesto Donde	El programa debe estar inicializado	Traducción a voz: Donde	OK
21	Gesto de finalización del programa	El programa debe estar inicializado	Mensaje de voz indicando que se ha finalizado el programa	OK

CAPÍTULO IV

DISCUSIÓN

4.1 Conclusiones

- Gracias al análisis del prototipo se logró conocer cuáles eran las debilidades más marcadas del prototipo y cómo podrían ser mejoradas.
- La interfaz natural de usuario diseñada provee una imagen amplia y amigable para la interacción con el usuario.
- El desarrollo por módulos del código fuente del sistema resultó en un sistema mucho más legible y modificable, así como también más eficiente al utilizar menos líneas de código.
- Las pruebas realizadas gesto por gesto permitieron identificar los errores que no eran fácilmente visibles, además dieron paso a modificar ciertos valores para adaptarlos al resultado final deseado.

4.2 Recomendaciones

- El sistema utiliza los idiomas instalados en el sistema operativo para la salida de voz. Se recomienda instalar el paquete de idiomas para el Sistema Operativo Windows de manera que se puedan utilizar diferentes voces e idiomas en el sistema para el momento de traducción a voz.
- Se recomienda ubicar el sistema en un lugar con un fondo adaptable de manera que el contraste de la vestimenta del usuario con el fondo no ocasione problemas de detección por parte del sensor debido a la sensibilidad del mismo.

- Se recomienda que si se quiere modificar el sistema, que al programar los textos de los gestos que serán traducidos no sean demasiado largos debido a que el sistema necesita realizar traducciones de forma rápida y un texto largo va a ser traducido de forma lenta a voz.
- El sistema requiere de un buen sistema de ventilación y enfriamiento acorde al tiempo que va a ser utilizado. Es recomendable instalar un sistema de enfriamiento que coincida con el uso que se le va a dar al sistema.
- Se recomienda constar de periféricos inalámbricos para la configuración del sistema (mouse y teclado) para evitar tener que abrir el equipo y conectar cables que podrían crear interferencias dentro del equipo.

BIBLIOGRAFIA

Amon, C., & Fuhrmann, F. (2014). *Evaluation of the spatial resolution accuracy of the face tracking system for Kiect for windows v1 and v2.* Recuperado el 04 de Abril de 2016, de: content/uploads/2014/12/Amon_Fuhrmann_Graf_EvaluationOfTheSpatialResolutionAccuracyOfTheFaceTrackingSystemForKinectForWindowsV1AndV21.pdf

Cerezo, Y., Peñalba, O., & Caballero, R. (2007). *Iniciación a la Programación en C# en un Enfoque Práctico.* Madrid: Delta Publicaciones.

León, F. (2007). *Lógica y Programación Orientada a Objetos: Un Inicio al Desarrollo de Software. ITM.*

George, B. (2005*). Introducción a la informática. Madrid: PEARSON EDUCACIÓN.*

Joyanes, L. (2002). *C# Manual de programación. Madrid: McGrawHill.*

Martinez, P., Cabello, M., & Días, J. (1997). *Sistemas Operativos Teoría y Práctica.* Madrid: Días de Santos.

Montserrat, M. (26 de Mayo de 216). *Kinect for Developers*. Obtenido de http://www.kinectfordevelopers.com/es/2014/01/28/caracteristicas-kinect-2/

Niño, j. (2011). *Sistemas Operativos Monopuestos.* Madrid: EDITEX.

Westerman, W. (2010). *Hand Tracking, Finger Identification on Multisource Frameworks.* Chicago.

Wigdor, D. (2011). *Brave NUI World. Amsterdam.*

ANEXOS

ANEXO 1. Prototipo Traductor de Gestos.

Figura 47. Prototipo del Sistema Traductor de Gestos

ANEXO 2. Ensamblaje segundo prototipo

Figura 48. Construcción de la parte superior de la estructura.
Elaborado por: Santiago Constante, Andrés Yépez

Figura 49. Construcción de la parte media de la estructura
Elaborado por: Santiago Constante, Andrés Yépez

Figura 50. Colocación de los elementos en la estructura
Elaborado por: Santiago Constante, Andrés Yépez

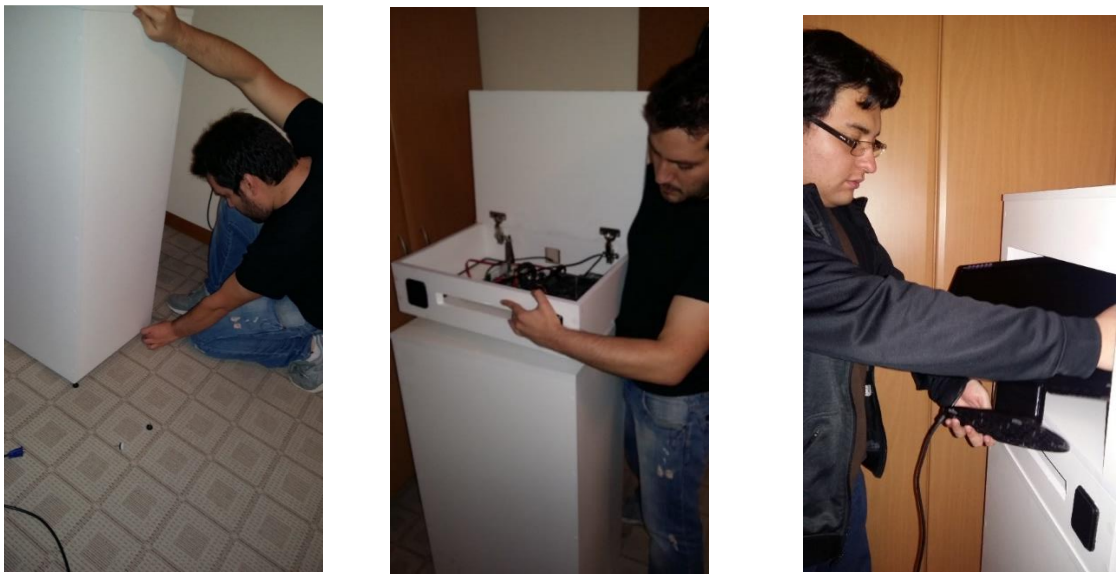


Figura 51. Estructura finalizada
Elaborado por: Santiago Constante, Andrés Yépez