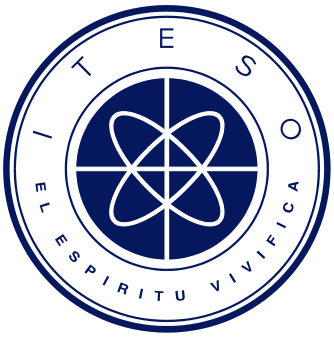
Instituto Tecnológico y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

ESPECIALIDAD EN SISTEMAS EMBEBIDOS



Control por gestos usando Leap Motion.

Tesina para obtener el grado de:

Especialista en sistemas embebidos

Presenta(n)
Juan Eduardo López Flores.

Director de tesina: Dr. Luis Enrique Gonzalez Jimenez

San Pedro Tlaquepaque, Jalisco. 18 de Noviembre de 2016.

Índice

Tabla de figuras

Agradecimientos

Mis más sinceros agradecimientos a todas las personas como mis padres y colegas ingenieros, que con sus esfuerzos y enseñanzas me sirvieron de motivación para dedicar mi tiempo en ser cada día un poco menos ignorante en la rama de sistemas embebidos. Dios guarde de ellos cada día de sus vidas. De igual manera, espero que nuestras autoridades sigan fomentando programas para facilitar la profesionalización de nuestros compañeros y colegas ingenieros.

Resumen

El siguiente trabajo de tesis está enfocado a demostrar que en un sistema embebido donde la interfaz gráfica juega un papel preponderante con la interfaz del usuario de sistemas como los automotrices dígase tablero o sistemas de navegación, en donde es necesario buscar la manera en que el usuario pueda interactuar con el mismo sistema sin que descuide aspectos tan importantes como la conducción. Si bien, hay dispositivos en los sistemas de propósito como el “mouse” que le permiten a cualquier usuario la navegación en un sistema con interfaz gráfica, este no sería ergonómico o adecuado para ciertas situaciones tales como un tablero de automóvil. Es por ello que dicho trabajo propondrá una alternativa que haga un balance entre lo ergonómico, seguro y no tan caro.

Para el caso en el que los clásicos sistemas de navegación en una interfaz gráfica como botones, “mouse”, “joysticks” y pantallas táctiles no satisfagan las nuevas necesidades de navegación en las cuales el tiempo en el que uno dispone para usar algún menú, así como el espacio disponible para tener dicha interfaz, se requiere ya de otros tipos de dispositivos que así lo permitan. Afortunadamente, ya se empiezan a desarrollar algunas soluciones que nos permitan tener sistemas de interfaz graficas con un sistema de interfaz más amigable con el usuario. Son los dispositivos de control por gestos. Estos y en especial uno que por sus características no intrusivas con el usuario nos permiten llevar a cabo el siguiente trabajo de investigación.

Abstract

This postgraduate dissertation work is focused on showing an Embedded system's HMI has a key play in the recent automotive user interfaces such as Dashboards or infotainment devices where new technologies like screens have replaced the classic display way. The advantages of the new kind of screens are fulfilled by the advanced operating systems. These OS of the general purpose aimed to be used only by desktops are available now for embedded systems due to the large-scale integration of the new embedded processor that allows having more benefits using operating system based on Linux. Without the Linux's resources, it is not possible to display and control the graphics printed in infotainment systems screen nor the dashboard's indicators.

1 CAPITULO I: INTRODUCCIÓN.

Actualmente en los avances en ciertos campos de la electrónica nos sorprende al tener opciones como el control de gestos. El control a través de gestos es posible con la solución ofrecida por Leap Motion. Mediante el uso de librerías del fabricante esta solución promete adaptarse a cualquier entorno que permita usarlas. Sin embargo, lo que hace aún más interesante a esta pequeña plataforma (además de ser una ventaja comercial) es el control de una interfaz gráfica.

La cámara barre el plano del usuario y modela virtualmente la acción las manos igual que ya lo hace cuando lo usamos con un laptop. Todo ello colocando frente al monitor y conectándolo por medio de cable USB (Véase Figura 1) a una terminal con algún SO soportado (Windows 7,8 y 10), Mac OS y Linux el cual será nuestro caso de estudio.

Figura 1. Conexión de un ordenador con el Leap Motion mediante USB.

Con lo anterior se es capaz de capturar los movimientos de manos, dedos y brazos desde luego, con alta precisión.

El cerebro de tal dispositivo son 2 cámaras monocromáticas con 3 LEDs infrarrojos como se muestra en la Figura 2, lo que permite interactuar mediante el uso de las manos.

Figura 2. Se muestran los 2 lentes de las cámaras monocromáticas en conjunto con los 3 LEDs infrarrojos.

Posteriormente, el fabricante procesa en la terminal la imagen removiendo el ruido y construyendo mediante modelos las manos, los dedos y gestos. Otra de las comodidades ofrecidas es la interacción de una interfaz de usuario que nos indica que movimiento es el ejecutado.

1.1 Estado del arte

El control por gestos es un método de interfaz con el usuaria basado en la interpretación de gestos humanos a través de dispositivos informáticos (hardware y software) y algoritmos matemáticos. Sin embargo, hay una gran gama de dispositivos que ofrecen desde prestaciones similares de control a través de gestos usando principios similares hasta aquellos que recurren a tecnologías (y procesamientos novedosos) como cámras. Solo mencionarán algunos de ellos dependiendo como procesan las señales del exterior (tipo de sensor).

1.1.1 Myo.

El MYO es un producto permite el control de diferentes dispositivos a través de gestos, es decir, movimientos con los brazos, sin embargo, utilizando un método de reconocimiento de movimientos distinto al de las cámaras infrarrojas como puede ser el caso del Leap Motion o Kinect. El concepto se

basa en los sensores tipo “Electromiograficos” basados en electrodos pegados en brazo. A continuación, en la figura 3, en el sensor MYO se pueden apreciar los electrodos, así como la forma en que este se adapta al brazo (tipo brazalete) el cual es mostrado a continuación.

Figura 3. Brazalete Myo.

1.1.2 Nod.

Este dispositivo que utiliza sensores basados en acelerómetros y giroscopios que conectados con un pequeño trasmisor Bluetooth sustituye en gran medida el control por contacto físico ya sea de los botones de una pantalla de TV o un celular. Sin embargo, para nuestro propósito, sigue siendo intrusivo para un ambiente como el de un usuario de un automóvil, donde quizá el usuario no prefiera conectarse algún dispositivo al cuerpo por motivos de comodidad. En la figura 4 se muestra un ejemplar de dicho dispositivo.

Figura 4. Anillo “Nod”.

1.1.3 Kinect.

El sistema Kinect (similar al Leap Motion), cuenta con una cámara tipo RGB, un sensor de profundidad, un micrófono y un procesador que ejecuta el un algoritmos, el cual, proporciona captura de movimiento de todo el cuerpo en tres dimensiones, reconocimiento facial y capacidades de reconocimiento de la voz. El Kinect, al contrario de las opciones mostradas anteriormente, es sobrado para los posibles objetivos donde sólo se requiere reconocimientos de gestos, sin embargo, puede ser útil por las características de reconocimiento de patrones de voz dentro de un automóvil, algo que ya tiene algunos usos dentro del ambiente automotriz.

En la figura 5, se aprecian todos los componentes del Kinect.

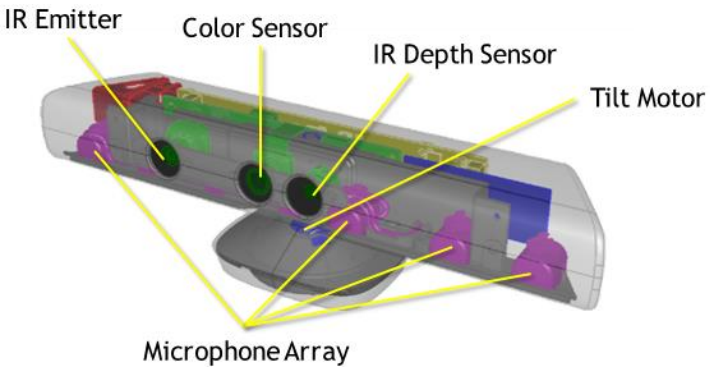


Figura 5. Sistema Kinect mostrando todos sus subcomponentes.

1.1.4 Leap Motion.

Este dispositivo dadas las cualidades en precio (descartando de momento opciones como el Kinect) será el objeto de estudio del presente trabajo. El dispositivo combina tanto recursos de Hardware como de software para lograr decodificar los gestos del usuario. Consta de un par de cámaras estéreo infrarrojas con iluminación LED. Los datos del sensor son recibidos especialmente los provenientes de las manos, dedos y brazos.

En la figura 6 se puede ver como es la interacción con el sistema Leap Motion y el usuario.

Figura 6. Pantalla contralada por gestos del Leap Motion.

La ventaja de usar este sistema, es que tiene una amplia comunidad de desarrollo, tal vez no es tan robusto como Kinect, pero por el precio y las prestaciones lo hacen idóneo para ser considerado la opción a desarrollar en el presente trabajo. Por tal motivo, se puede empezar a trabajar en ya una base conocida o el llamado “Hello world” Project que puede ser la base del proyecto.

1.2 Objetivo

Controlar la interfaz gráfica de usuario del sistema operativo basado en Linux integrando las librerías del sensor Leap Motion.

1.3 Justificación

Debido a que otros sistemas no permiten al usuario la libertad de interactuar en un entorno grafico sin necesidad de ser intrusivo con mismo usuario, la opción del Leap Motion es la más idónea, es decir, el control de una interfaz con el uso de gestos es hasta seguro en un ambiente automotriz donde el mismo usuario debe poner especial cuidado en otros aspectos como la conducción del automóvil.

1.4 Problemática

El desarrollo del problema gira alrededor de una plataforma de desarrollo compatible con el hardware, y más específicamente, con el soporte por parte del fabricante para poder usar el sensor Leap Motion.

1.4.1 Integración de Leap Motion a Minnowboard MAX 2

Es necesario definir el tipo de código que se necesita para manipular el dispositivo. Para ello, vale la pena recordar **dos** aspectos fundamentales. El primero y el que definirá el seguimiento del presente trabajo, y esto es la plataforma en el que las ***APIs*** del Leap Motion tienen soporte[1]. Debido a la falta de soporte para plataformas basadas en el microprocesador ARM XX, se determina por simple eliminación a la plataforma de Intel x64 (incluso hay soporte para la plataforma x86).Si bien, y no menos importante es la selección del sistema operativo donde ejecutará el código. Ya definido la plataforma donde dicho código se ejecutará, se seleccionará Linux sobre un procesador Intel x64. Sobre esta plataforma se correrá una distribución de Ubuntu 14.04 debido al buen nivel de integración con el hardware[2]. Dicha plataforma es de *hardware abierto* conocida como “Minnowboard” basada en un procesador Intel Atom E38xx.

Relacionado a lo anterior, sobre el uso de Linux sobre una plataforma embebida, el hardware usado se manejará por un driver tipo “wrapper” que es una colección de subrutinas o clases para manejar al Leap Motion. En esto es donde se centrará este proyecto. En el caso de un automóvil donde se puedan centralizar algunas opciones de *infotainment.* Algunas opciones correspondientes a la navegación, entretenimiento o control de medios son por el momento ajenas a las funciones del tablero principal donde por el momento en la mayor parte de los OEMs solo permiten indicadores sobre la conducción y otros aspectos importantes.

Por ejemplo, en la siguiente figura 7 podemos ver un ejemplo donde está el velocímetro en el tablero como imagen y no analógico o analógico-digital y al lado se podría seleccionar entre el GPS, menú de canciones o de otras opciones de navegación.

Figura 7. Tablero automotriz con opciones de infotainment y conducción (velocímetro).

1.5 Alcances

Como se mencionó anteriormente, las actividades con el Leap Motion son las de control de ventanas en un sistema operativo basado en Linux y en específico sobre la versión 14.04 de la distribución de Ubuntu. Lo anterior, sobre una plataforma Minnowboard MAX 2 de Intel.

2 CAPITULO II: MARCO TEORICO Y ANALISIS.

2.1 Definición de ciclo de desarrollo del proyecto

De los ciclos de desarrollo conocidos, se enfocará al ciclo en cascada. Si bien, hay distintos tipo de ciclos de desarrollo, el más usado en la industria automotriz es el ciclo en “V” el cual, es una mejora al ciclo en cascada. Y como se sabe, la elección del ciclo de desarrollo depende del mercado donde se desarrolló el producto. A continuación, se detalla de manera somera el modelo de desarrollo en la figura 8[3].

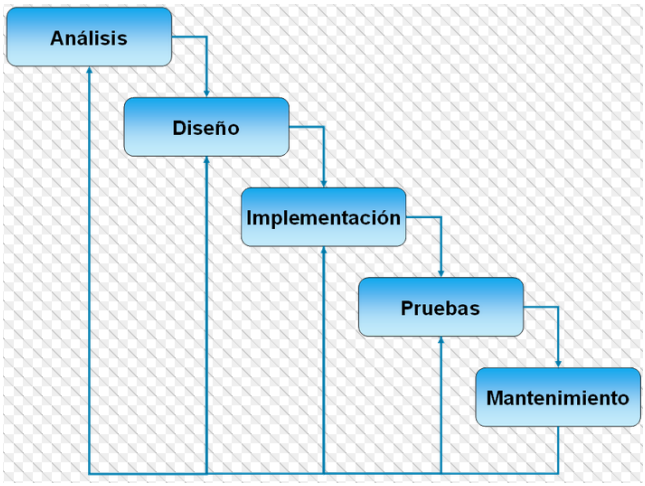


Figura 8. Modelo de desarrollo en cascada aplicado al proyecto.

El desarrollo del proyecto se llevó a cabo en el lapso de Agosto de 2016 a Noviembre de 2017, con las siguientes actividades principalmente:

2.1.1 Análisis.

Esta fase se trata primordialmente de analizar la documentación del desarrollador del dispositivo, ya que nos permitiría determinar el uso que se le podría dar. A partir de ello, se pueden establecer en los requerimientos de sistema las características adecuadas para un óptimo funcionamiento.

2.1.2 Diseño.

Es la siguiente fase de desarrollo después del análisis, en esta se pueden trazar los requerimientos a las posibles modelos de implementación. Además, se analizarán cuestiones del ambiente y lenguaje de desarrollo

2.1.3 Implementación.

Es la etapa más larga que consiste principalmente en implementar los requerimientos usando las herramientas previamente definidas. Ya habiendo hecho esto, se integrará en una solución ya completa que nos permita hacer pruebas. Es aquí donde se *verifica* el diseño.

2.1.4 Pruebas.

Es la forma de *validar* los requisitos de usuario y donde se tratará de demostrar que funciona correctamente.

2.1.5 Mantenimiento.

Debido a la premura del proyecto, esta fase no se tomará en cuenta. Sólo si se requiere que una mejora se implemente y posteriormente se verifique y revalide dicho cambio en la aplicación.

2.2 “Device driver”

Una de las partes fundamentales de este trabajo, es la integración de las librerías provistas por el desarrollador del producto. Sin poder integrar dichas librerías deberíamos hacer por nuestra cuenta el driver del dispositivo. Por ello se debe tomar en cuenta los siguientes términos.

Desde el punto de vista del usuario, el “device driver” es como una caja negra y la complejidad de trabajar directamente con el hardware no es accesible directamente al usuario. Es decir, el driver provee una manera abstracta de usar el mismo. Es como una aplicación que es interfaz a muy bajo nivel. La figura 9 nos dice [4]:

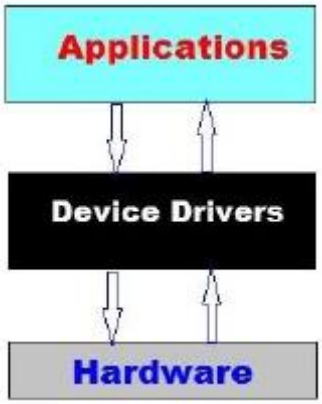


Figura 9. Interacción del driver con la aplicación y el hardware.

Si bien, la funcionalidad del kernel de Linux está dividida en varias tareas, el driver interactúa con el kernel para llevar a cabo alguna operación. En la capa baja del sistema operativo están los drivers de cada uno de los dispositivos conectados al sistema. Un driver por lo menos tiene implementado diferentes tipos de llamados: Para abrir, cerrar, leer y escribir. La integración de los drivers viene explicada en la figura 10 mostrada a continuación.

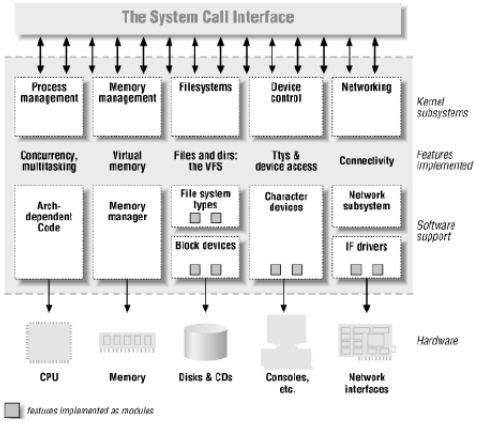


Figura 10. Interacción del kernel con los drivers.

2.3 Control por gestos

Es necesario definir los gestos detectados por el dispositivo a utilizar [5]:

Tipo “***CircleGesture”***. Este es un movimiento circular del dedo.

Tipo “***SwipeGesture”***. Este es un movimiento en línea de la mano con los dedos extendidos.

Tipo “***ScreenTapGesture”.*** Este es un movimiento de pulso (“Click”) sobre la pantalla. Este va de arriba hacia abajo.

Tipo “***KeyTapGesture”.*** Este es un movimiento de pulso (“Click”) sobre la pantalla. Sin embargo, este al contrario del anterior, va de abajo hacia arriba.

Los movimientos explicados anteriormente se ilustran a continuación en la figura 11:

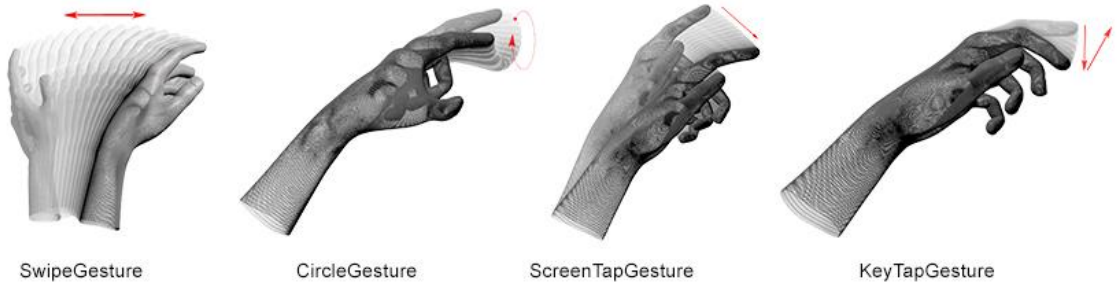


Figura 11. Tipos de movimientos soportados por el Leap Motion.

Estos gestos están contenidos en una estructura de datos tipo ***Enum*** en la librería “Leap.h” dentro de la clase “Gestures” como viene a continuación:

```
class Gesture : public Interface {
  public:
    // For internal use only.
    Gesture(GestureImplementation*);

    /**
     * The supported types of gestures.
     */
```

```
  enum Type {
    TYPE_INVALID    = -1, /**< An invalid type. */
    TYPE_SWIPE      = 1,  /**< A straight line movement by the hand with fingers extended. */
    TYPE_CIRCLE     = 4,  /**< A circular movement by a finger. */
    TYPE_SCREEN_TAP = 5,  /**< A forward tapping movement by a finger. */
    TYPE_KEY_TAP    = 6   /**< A downward tapping movement by a finger. */
  };
…
};
```

Ya en la parte de la aplicación, y tal como viene en el ejemplo del Leap Motion, se usa la librería de la siguiente manera:

```
void SampleListener::onConnect(const Controller& controller)
{
        controller.enableGesture(Gesture::TYPE_CIRCLE, bool enable = true);
        controller.enableGesture(Gesture::TYPE_KEY_TAP, bool enable = true);
        controller.enableGesture(Gesture::TYPE_SWIPE, bool enable = true);
        controller.enableGesture(Gesture::TYPE_SCREEN_TAP, bool enable = true);
        std::cout << "Connected" << std::endl;
}
```

donde la función "SampleListener" es usa llama a esa función "enableGesture". Tenemos el caso contrario, y es donde al cerrar la sesión de Leap Motion se recomienda deshabilitar los gestos, esto como se muestra a continuación:

```
void SampleListener::onDisconnect(const Controller& controller)
{
        controller.enableGesture(Gesture::TYPE_CIRCLE, bool enable = false);
        controller.enableGesture(Gesture::TYPE_KEY_TAP, bool enable = false);
        controller.enableGesture(Gesture::TYPE_SWIPE, bool enable = false);
        controller.enableGesture(Gesture::TYPE_SCREEN_TAP, bool enable = false);
        std::cout << "Disconnected" << std::endl;
}
```

En esta función, habilita los tipos de gestos enunciados en el ***ENUM***. Tenemos en la tabla 1 mostrada a continuación.

Name	enableGesture
Type:	type: The type of gesture to enable or disable. Must be a member of the Gesture::Type enumeration. enable : True, to enable the specified gesture type; False, to disable.
Range:	Gesture::Type ***<-1,1,4,5,6>*** enable ***<False, True>***
Description:	Enables or disables reporting of a specified gesture type.
Constants of this type:	

Tabla 1. Descripción de enum donde se contienen los gestos soportados.

Ya teniendo las funciones de conexión y desconexión, se tiene la función para procesar las tramas de imágenes o "Frames". Al detectar una mano, se puede acceder a las propiedades del objeto "Frame". Este se crea llamando a la función "***Controller::frame()***", para este caso, el objeto se llamará "frame".

```
void SampleListener::onFrame(const Controller& controller)
{
        const Frame frame = controller.frame(); //Creating "Frame" object named "frame"
        …
        …
};
```

Cabe señalar que podemos acceder hasta 60 frames y de igual manera se puede seleccionar un histórico de ellos. Esto sería como se muestra a continuación en el código:

```
void onFrame(Leap::Controller &controller)
  {
    Leap::Frame frame = controller.frame(); //The latest frame
    Leap::Frame previous = controller.frame(1); //The previous frame
    //...
  }
```

En el mismo frame tenemos las siguientes propiedades del objeto "**frame**", se mencionarán las más importantes para el proyecto:

Identificador de frames. El orden en que va llegando información de cada gesto: frame.timestamp() .

Timestamp: Tiempo absoluto de la llegada del frame: frame.timestamp() .
Numero de manos: Numero de mano en la ***lista:*** frame.hands().count() .
Numero de dedos: Numero de dedos en la ***lista:*** frame.fingers().count()

2.3.1 Descripción de los gestos.

El software de Leap Motion reporta gestos observados en una trama o "frames" de la misma manera que se reportan datos de manos y dedos. Para cada gesto descrito a continuación, de un objeto tipo "Gesto" se tiene de un objeto "Frame" estas descritas a continuación.

2.3.1.1 Discriminación de mano o "Hands".

Son un conjunto de funciones que nos permiten diferenciar que mano, dedos y gestos corresponden a cada una de las manos detectadas (derecha o izquierda). La forma en que se puede diferenciar que mano es la que se está moviendo es a través de la clase "hands". Esta representa la mano detectada por el Leap Motion. El objeto a esta clase nos proporciona una lista de atributos como la posición, orientación y movimiento. De igual manera se puede identificar que mano se está moviendo [6].

2.3.1.2 Atributos de la clase “Hands”.

En la tabla 2 se muestra algunos atributos de la clase “hands” que pudieran ser útiles en el desarrollo del código, principalmente aquellos que nos pudieran permitir discriminar que mano es la que está produciendo el movimiento.

Attribute	Description
isRight	Si la mano es la derecha
isLeft	Si es la mano izquierda.
Palm Position	La posición de la palma de la mano desde el centro del Leap Motion.
Palm Velocity	La velocidad y movimiento de la palma de la mano en mm/s.
Palm Normal	Vector de posición debajo de la palma.
Direction	Vector de la palma de la mano hacia los dedos.

Tabla 2. Atributos de la clase mano.

Un caso de uso de las dos primeras expresiones sería como se sugiere a continuación.

Para la mano derecha:

```
if(hand.isRight()){
   // .. Do right handed stuff
}
```

Para la mano izquierda:

```
if(hand.isLeft()){
   // .. Do right handed stuff
}
```

En ambos casos dichas expresiones regresan un “True” sí es que la condición se cumple.

2.3.1.3 Deslizamiento o “Swipe”.

Este tipo de movimiento se detecta por *dedo*, por tal motivo, esto al parecer no puede ayudarnos a detectar si la *mano completa* se desliza de un lado para otro, de un lado para otro se refiere de derecha a izquierda o viceversa, de igual manera de arriba hacia abajo o viceversa. Los casos anteriores solo hacen mención a dos dimensiones, esto tomando en cuenta que el plano “X” (derecha - izquierda) y el plano “Y” (arriba - abajo) y son precisamente por que un tablero de auto solo cuenta con estas 2 dimensiones.

Por tal motivo, el plano “Z” quedaría descartado para ese tipo de desarrollos. La forma que se puede determinar la dirección del movimiento en los planos “X” o “Y” es por una función llamada “direction()”. En código se puede apreciar de la siguiente forma:

Leap::Vector swipeDirection = swipe.direction();

En este caso, para diferenciar el tipo de deslizamiento se pueden comparar los componentes del Vector “swipeDirection” (nombre usado en el ejemplo anterior). Por ejemplo, se puede comparar los valores de “X” o “Y” en sus respectivos valores absolutos, además entre estos mismos para saber si el desplazamiento fue horizontal o vertical, y esto se puede ver sobre la pantalla en cuestión. Relacionado a lo anterior, se debería discriminar o darle preferencia a cual de las dos dimensiones tiene un movimiento más relevante, y teniendo en cuenta eso darle preferencia a ello.

2.4 Descripción del sistema

Al momento de tener un “Listener”, se tiene un objeto del cual podemos usar sus propiedades para decodificar lo que las cámaras infrarrojas captaron. Dichas conversiones se llaman tramas o “frames” (nombre dado al objeto). La

cantidad de información por cada “frame” depende del procesamiento y visibilidad de las cámaras. El rango de acción se muestra en la figura 12.

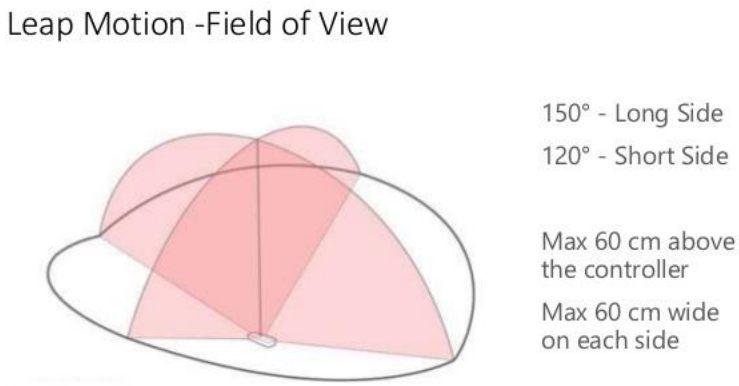


Figura 12. Rango operativo de las cámaras infrarrojas Leap Motion.

Por cada interacción de las manos con el sensor, se tendrá un “frame” totalmente distinto. Como ya se mencionó arriba, se tiene un buffer para poder almacenar las tramas de hasta 60 miembros. Estás pueden usarse posteriormente para hacer cálculos con filtros para ajustar la sensibilidad del apuntador.

2.4.1 Sistemas de coordenadas

El Leap Motion trabaja sobre los 3 ejes básicos, y sobre estos en el campo de visión soportado por el dispositivo. Los gestos grabados por las dos cámaras se transducen en “frames”, que en realidad son objetos, dichos objetos algunas de sus propiedades son la mano o manos, dedos, giros (ángulos).

A continuación, en la figura 13, se muestra la distribución de los ejes en el dispositivo [7].

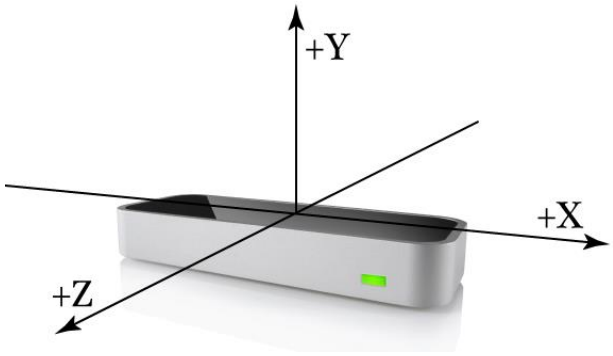


Figura 13. Distribución espacial del Leap Motion.

La información de cada gesto, viene dado por el tipo de información que entran a las cámaras. Tenemos que la velocidad de las manos, el movimiento de izquierda a derecha o viceversa da lecturas diferentes. En la siguiente captura de pantalla de la figura 14 tenemos un “Listener” o escuchador que polea los datos de entrada de la cámara.

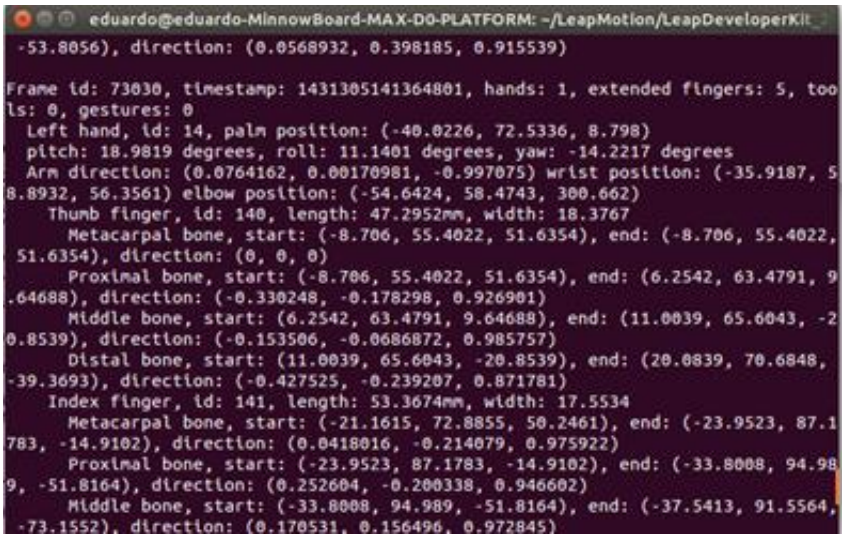


Figura 14. Captura de los datos con del escucha.

El escucha o “Listener” corresponde a las propiedades del objeto “frame” en este caso se tiene como se representa la siguiente parte de código tomado del ejemplo de proyecto.

```
void SampleListener::onFrame(const Controller& controller) {
```

```
    const Frame frame = controller.frame();
    std::cout << "Frame id: " << frame.id()
        << ", timestamp: " << frame.timestamp()
        << ", hands: " << frame.hands().count()
        << ", fingers: " << frame.fingers().count();
}
```

2.4.2 Arquitectura del sistema.

El Leap Motion tiene soporte para tres sistemas operativos principalmente, siempre y cuando corran sobre una arquitectura Intel x_64 o x_86. En plataformas de Windows corren como servicios y en sistemas operativos Linux o Mac OS son conocidos como “daemons”. El dispositivo se conecta con la plataforma a través de USB. El ambiente de desarrollo maneja dos variedades de APIs para capturar datos desde nuestro sensor: Interfaz nativa y una intarfaz WebSocket, esta última permitiría correr en el ambiente de explorador [8].

2.4.2.1 Interfaz de programación de aplicaciones o Leap Motion APIs.

Como ya se ha mencionado arriba, hay dos diferentes tipos de librerías Interfaz nativa, la cuales es más cercana al HW o más bajo nivel de abstracción con el HW, el desarrollador tiene soporte para varios lenguajes como Java, Unity, C++, C# y Python. Las interfaces WebSocket y el cliente Java Script permiten crear aplicaciones del Leap Motion enfocadas a un ambiente Web, como por ejemplo un ambiente de navegador.

Interfaz de aplicación Nativa del Leap Motion.

Este tipo de aplicación es dada por una librería cargada dinámicamente, o mejor conocida como DLL. Esta DLL conecta el servicio o “*demonio*” en Linux, lo cual provee datos a la aplicación. A continuación, se mostrará como funciona el Leap Motion sobre esta opción.

Aplicación del Leap Motion.

La figura 15 demuestra como el Leap Motion es empleado. Tomando en cuenta dichos campos hay 4 apartados.

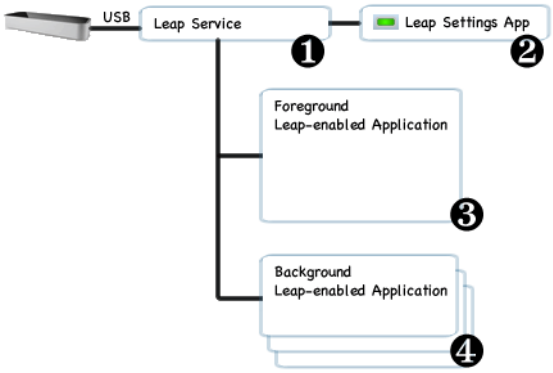


Figura 15. Arquitectura Leap Motion.

En el apartado “1” hace mención a la recepción de datos a través del puerto USB. Manda información al demonio o servicio, aquí, la misma información podría ser usado por otras aplicación. El usuario podría denegar el uso de la misma información a otras aplicaciones. Otra modalidad es el paso “2”, aquí el Leap Motion puede correr separado del servicio o demonio y permitiría al usuario correr su propia instalación.

En el caso “3”, aplicación en primer plano recibe datos desde el demonio o servicio. Aquí, llamando una DLL, donde la librería se liga con la aplicación directamente con un “header file” en el caso de C++. Y, por último, se tiene el estado “4” que indica que cuando el sistema deja en segundo plano a la tarea, el servicio del Leap Motion deja de enviar datos. Hay aplicaciones en segundo plano que a pesar que están en segundo plano, estas pueden recibir datos aún en segundo plano.

2.4.2.2 Ambiente de desarrollo o Leap Motion SDK

El ambiente de desarrollo contiene básicamente dos tipos de librerías que definen a las APIs. Una librería está programada en C++ la otra está en C. Hay una característica en dichas librerías cuentan con un repositorio de librerías de tipo Wrapper. A continuación, se listarán todas opciones soportadas para compilar en la tabla 3.

Lenguajes soportados.
C++ on Windows: Visual Studio 2008, 2010, 2012, and 2013
C++ on Mac: Xcode 3.0+, clang 3.0+, and gcc
Objective-C: Mac OS 10.7+, Xcode 4.2+ and clang 3.0+
C# for .NET framework versions 3.5+
Mono version 2.10
Unity Pro and Personal versions 5.3+
Java versions 6 and 7
Python version 2.7.3
UnrealEngine 4.9+

Tabla 3. Lenguajes soportados.

En la tabla 3 anterior, exceptuando la opción de Windows, se usará el compilador g++ (gcc), este debería estar instalado en el sistema operativo Linux. Durante el desarrollo del mismo, se pedirá instalar este y otros componentes.

2.4.2.3 Distribución Ubuntu sobre Minnowboard MAX II

De acuerdo a las especificaciones del fabricante[3], hay ciertos sistemas operativos soportados por la minnowboard que se listan a continuación:

- Debian GNU, Ubuntu, Fedora, Linux Mint.
- Yocto Project Compatible.
- Android 4.4 (Kitkat) and 5.0 (Lollipop) System.
- Microsoft Windows 8.1 and 10.

Para este trabajo se elegirá la plataforma Ubuntu. De acuerdo a las especificaciones del fabricante para instalar Ubuntu se necesita:

- Una SD card o unidad de memoria de más de 2 GB. En este caso se usará una SD card.
- Grabar el Rtfs de Ubuntu.

Tomando en cuenta lo anterior, se descargará la versión de 14.04 para AMD64, ya que el firmware de la tarjeta solo soporta dicha versión [9].

2.4.2.4 Manejo de gráficos en Linux.

Si bien, este podría ser un tema bastante vasto, el presente trabajo sólo se enfocará en un tipo de manejo de gráficos en el escritorio de la aplicación. En este caso hay varias opciones para el manejo de gráficos para la aplicación, a continuación se mencionarán los "drivers" gráficos de Ubuntu.

- X11.
- Qt, GTK, Motif.
- lxdm, lightdm, gdm, kdm, xdm.
- compiz, kwin, metacity, xfwm, openbox.
- KDE, Gnome, XFCE, LXDE.

En el caso de las opciones de manejo de ventanas, son las más conocidas Qt y X11. Sin embargo, debido al manejo de eventos gráficos de ambas opciones, la opción más idónea es X11.

2.4.2.5 Compiladores gcc y g++

El GCC es un compilador que viene ya en la distribución Ubuntu 14.04 y viene en varias versiones GNU. Por lo regular y dependiendo de las configuraciones del "Makefile" del del código así como un archivo .gdb que nos serviría para poder depurar la aplicación si es que así se necesita. La diferencia entre ambas opciones es el lenguaje del cual compilan. El gcc es el compilador

para lenguaje C y el g++ en el compilador para C++. Por el tipo de librerías que tenemos el compilador a usar es el ***g++***.

3 CAPITULO III: DESARROLLO.

3.1 Requisitos.

Hay distintas clases de requerimientos de las cuales las más relevantes de mencionar son los requerimientos funcionales y no funcionales. De los cuales se pueden resaltar las características de cada uno de los dos principales grupos mencionados.

Requisitos funcionales. Son aquellos que detallan el funcionamiento de una tarea específica del software, en los cuales pueden ser:

- Tener uno o varios parámetros de entrada.
- Por lo menos una salida.
- Ser probable por un caso de prueba.

Requisitos no funcionales. No afectan el funcionamiento de sistema, pero si el desempeño.

3.1.1 Requisitos funcionales.

1. Implementar el sistema “Leap Motion” en una arquitectura compatible con las librerías provistas por el desarrollador del dispositivo.

Debido a lo anterior, el desarrollador sabe que no hay soporte para arquitecturas ajenas al x_64 ó x_86, esto en base a las especificaciones del fabricante (en inglés):

“We don't support the Raspberry PI. The architecture problem occurs because you are trying to use a library compiled for Intel architectures on an ARM architecture system. This simply won't work. In any case, the Raspberry PI, even the newer one, doesn't have sufficient processing power to run the Leap Motion software acceptably.”[10]

Como sugiere el link anterior, las plataformas en base a ARM serían descartadas para un posible desarrollo, esto hasta que el desarrollador pueda compilar sus librerías para una arquitectura ARM. Por tal motivo, la plataforma Minnowboard es la idónea, ya que esta plataforma está basada en Atom x_64 en la cual hay soporte.

2. Integrar las librerías sobre un sistema operativo Linux.

Las librerías se van a compilar con el “makefile” del desarrollador sobre un sistema Linux. La distribución seleccionada para ser instalada es Ubuntu. Se instalará en una unidad de memoria tipo SD, sobre la cual se instalará el boot y RTFS emulando el “Hard drive” o disco duro. La memoria seleccionada en capacidad es de 16 GB. Según el desarrollador de la plataforma ha recomendado el uso de una memoria arriba de clase 4.

Requerimientos de reconocimientos de gestos de control:

3. Seleccionador de iconos. Seleccionar un elemento de la pantalla usando un dedo.

Aquí, de acuerdo a las características de los gestos ya descritos, el movimiento descrito con este movimiento es el conocido como el “Screen Tap Gesture”. Aquí, para crear el efecto de “Click” es ir de presionar de arriba hacia abajo, después, ir de abajo hacia arriba. Se bebe tener en cuenta que para poder darle click a un punto es necesario un apuntador en la pantalla, dicho apuntador podrá indicar al usuario que es lo que va a seleccionar. Esto haciendo analogía con un monitor de un CPU.

4. Desplazador Vertical. Desplazar verticalmente los elementos de la pantalla.

De acuerdo a las características de los gestos ya descritos, el movimiento descrito con este movimiento es el conocido como el “Swipe Gesture”.

5. Desplazador Horizontal. Desplazar verticalmente los elementos de la pantalla.

Aquí, de acuerdo a las características de los gestos ya descritos, el movimiento descrito con este movimiento es el conocido como el “Swipe Gesture”.

3.1.2 Requerimientos no funcionales

6. Velocidad de respuesta del apuntador.
7. Precisión del gesto de selección con un solo dedo.
8. Precisión del gesto de deslizamiento de pantalla de izquierda a derecha.
9. Numero de dedos en gesto de deslizamiento de pantalla de izquierda a derecha.
10. Precisión del gesto de deslizamiento de pantalla de derecha a izquierda.
11. Numero de dedos en gesto de deslizamiento de pantalla de derecha a izquierda.

3.2 Diagramas de flujo.

/**

Function : SampleListener::onInit

Called once, when this Listener object is newly added to a Controller.

**/

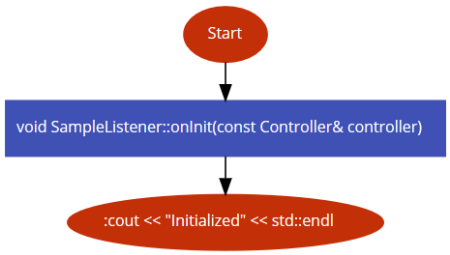


Figura 16. Function : SampleListener::onInit

/**

Function : SampleListener::onDisconnect

Called when the Controller object disconnects from the Leap Motion software.

The controller can disconnect when the Leap Motion device is unplugged, the user shuts -

the Leap Motion software down, or the Leap Motion software encounters an unrecoverable error.

**/

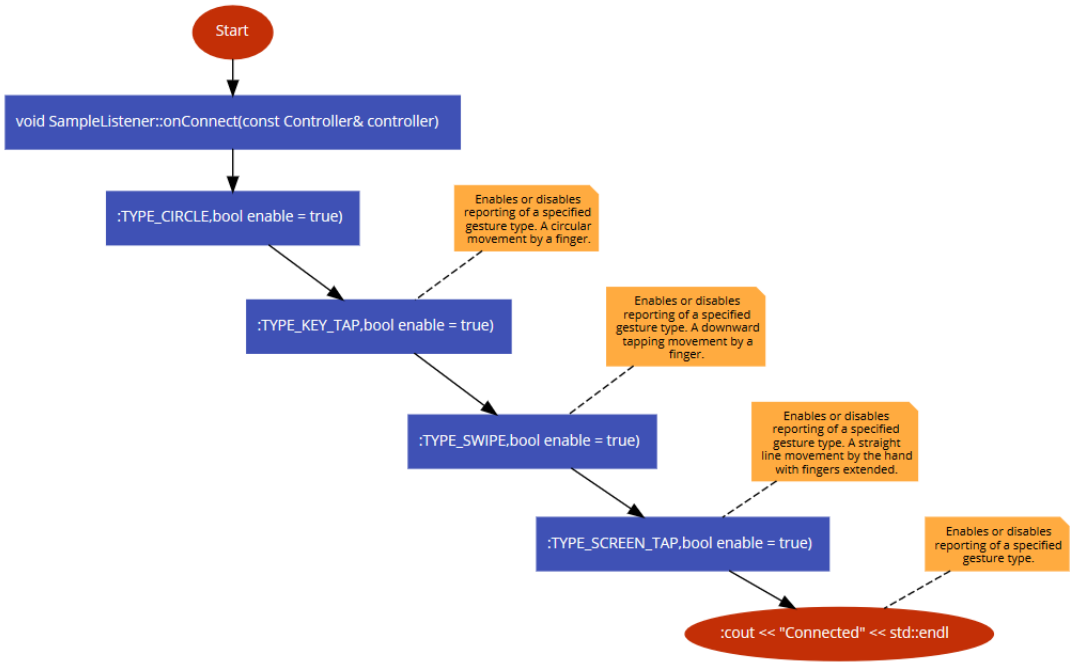


Figura 17. Function : SampleListener::onDisconnect

/**

Function : SampleListener::onDisconnect
Called when the Controller object disconnects from the Leap Motion software.
The controller can disconnect when the Leap Motion device is unplugged, the user shuts -
the Leap Motion software down, or the Leap Motion software encounters an unrecoverable error.

**/

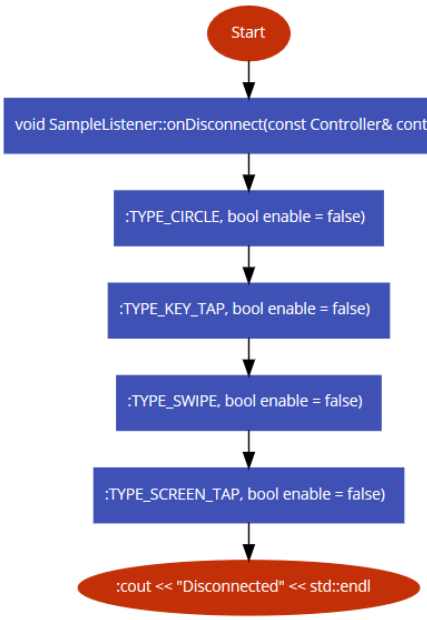


Figura 18. Function : SampleListener::onDisconnect

/**

Function : SampleListener::onExit
Called when this Listener object is removed from the Controller or the Controller instance is destroyed.

**/

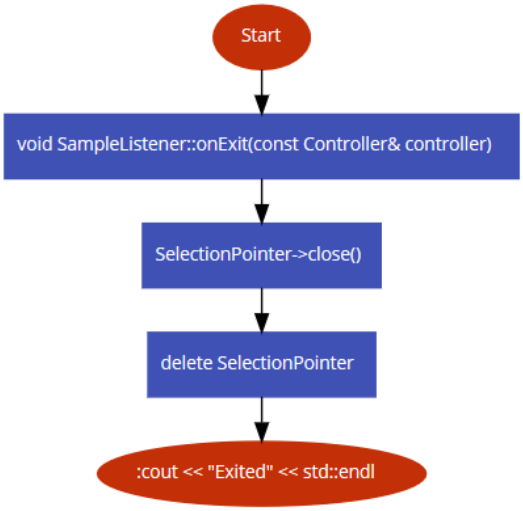


Figura 19. Function : SampleListener::onExit

/**

Function : SampleListener::onFocusGained
Only the foreground application receives tracking data from the Leap
Motion Controller. This function is only called when the controller object is in a connected state.

**/

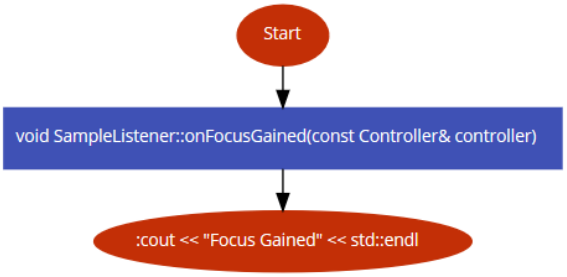


Figura 20. Function : SampleListener::onFocusGained

/**

Function : SampleListener::onFocusLost
Only the foreground application receives tracking data from the Leap Motion Controller.
This function is only called when the controller object is in a connected state.

**/

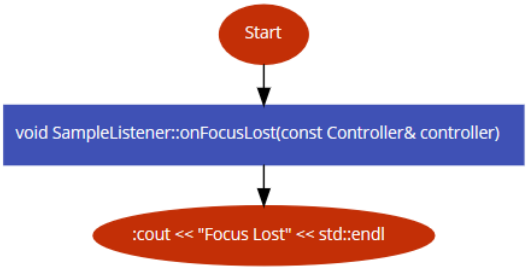


Figura 21. Function : SampleListener::onFocusLost.

/**

Function : SampleListener::onFrame
Called when a new frame of hand and finger tracking data is available.
Access the new frame data using the Controller::frame() function.

**/

Parte 1. Ésta parte de la función “onFrame”.

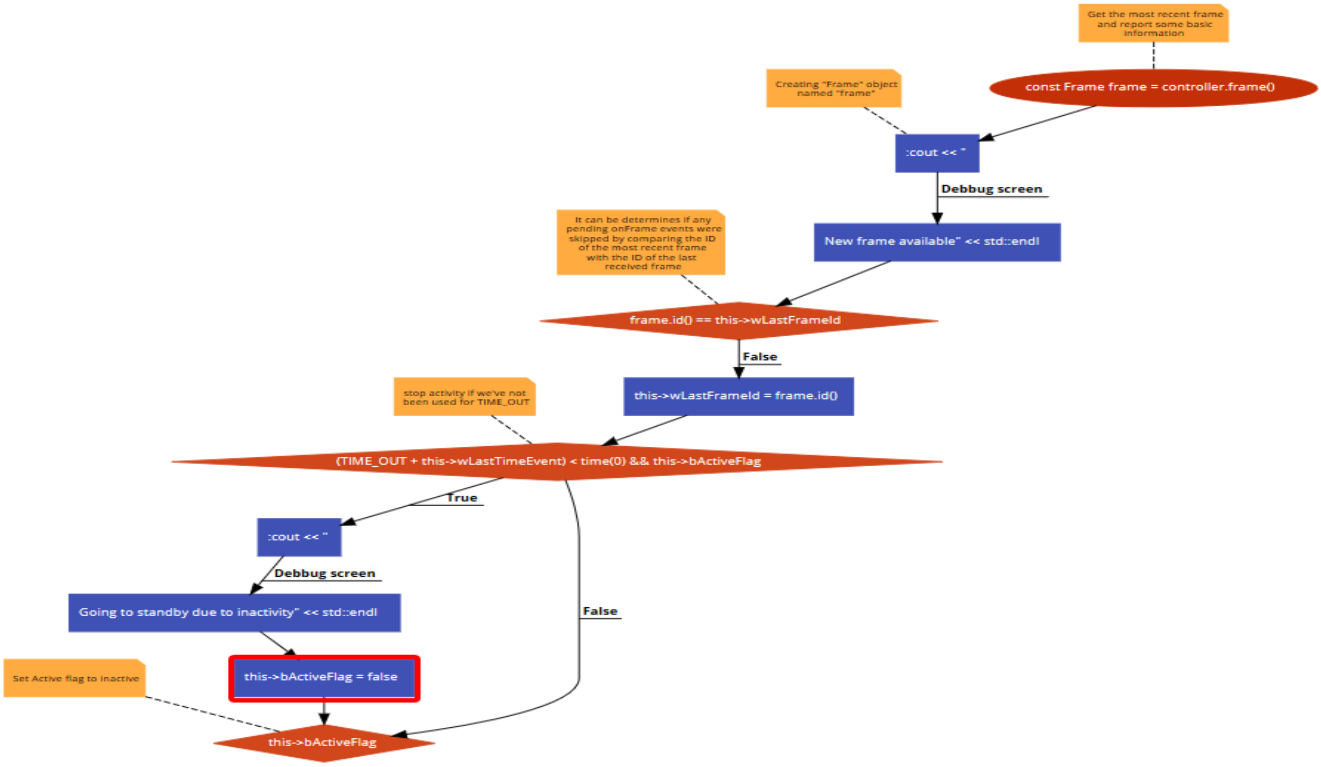


Figura 22. Function : SampleListener::onFrame, entrada de datos y TIME_OUT

/**

Function : Main Function

This function starts the Leap Motion application.

**/

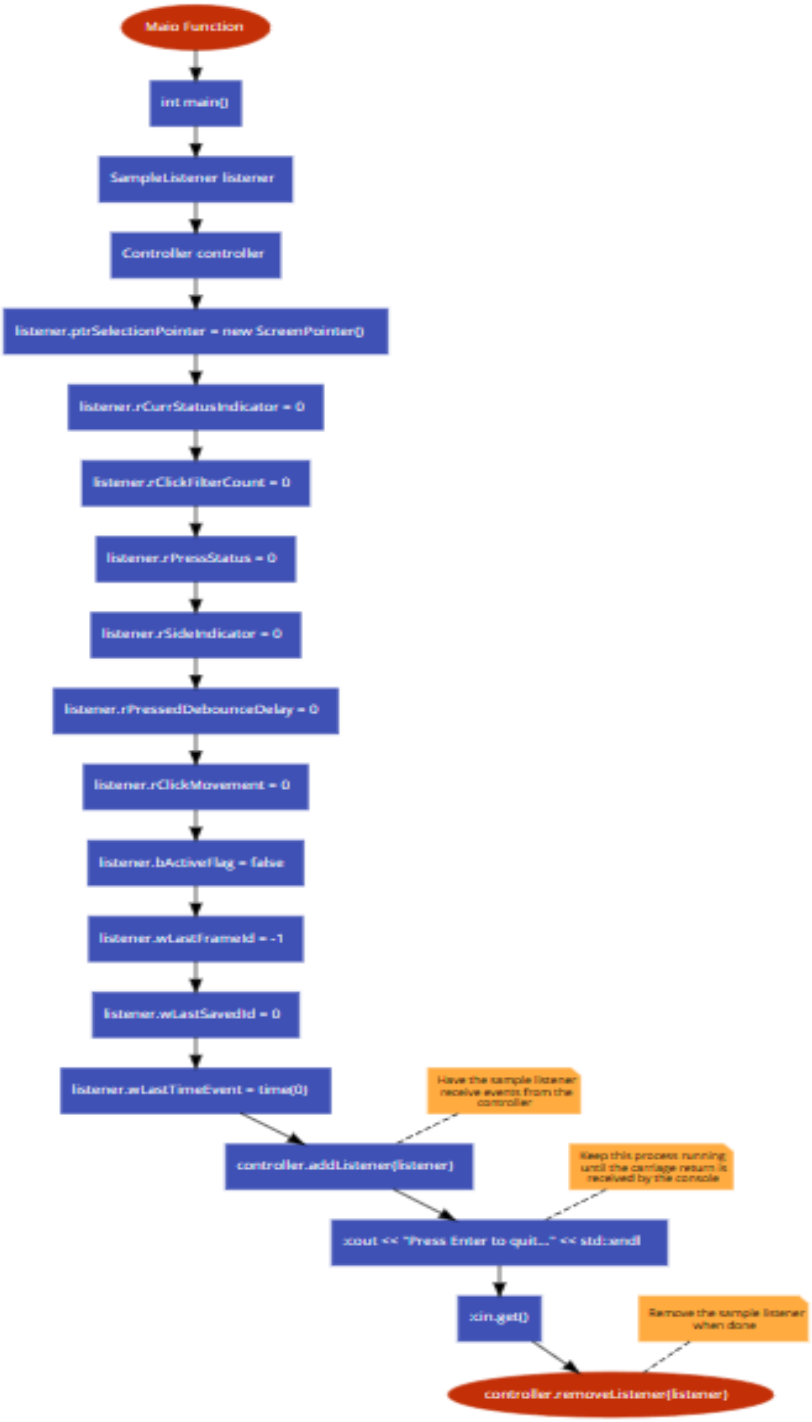


Figura 23. Function : Main Function.

3.3 Presentación de resultado.

El sistema operativo sobre el cual se va a trabajar es el Ubuntu14.04 sobre Minnowboard, habiéndolo previamente instalado se procederá a abrir la terminal. En la cual mediante el uso de comando en la misma (como se ve en la figura 24) se instalarán, configurarán y correrán los servicios requeridos para la aplicación.

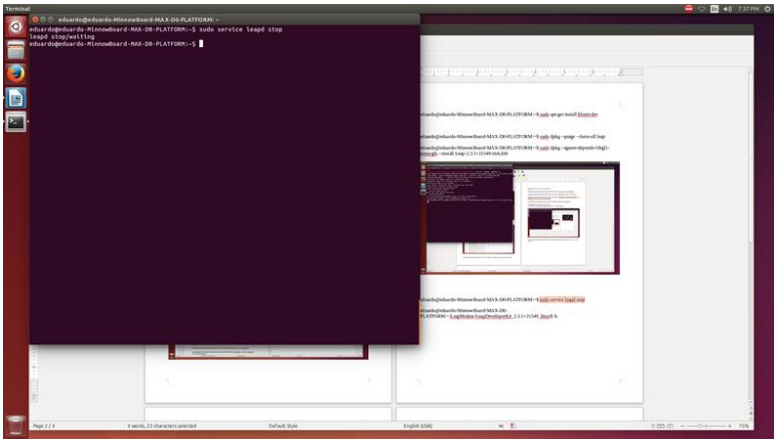


Figura 24. Terminal de Ubuntu.

Antes de poder correr la aplicación es necesario verificar que las librerías invocadas por la aplicación estén instaladas en el sistema operativo. Las más necesarias es el mismo compilador de GNU g++, que de manera análoga al conocido gcc, servirá para poder compilar el código de la aplicación. En seguida, en la terminal del sistema operativo:

```
sudo apt-get install g++
```

Pero, en el caso de nuestro sistema operativo ya fue instalado previamente para la realización de este trabajo, por tal como viene en la figura 25 se pondrá en la terminal de Linux el comando “Whereis” para ver el directorio donde está instalado la aplicación.

```
eduardo@eduardo-MinnowBoard-MAX-D0-PLATFORM:~$ whereis g++
g++: /usr/bin/g++ /usr/bin/X11/g++ /usr/share/man/man1/g++.1.gz
```

Figura 25. Comando whereis g++ para saber la ubicación del compilador g++.

La opción de arriba nos permitiría instalar la última versión del compilador g++, otras alternativas son primero verificar si este ya se encuentra instalado o instalar toda la paquetería completa de compiladores invocando el comando build essentials. De manera similar al caso del compilador g++, en el caso de nuestro sistema operativo ya fue instalado previamente la paquetería build essential para la realización de este trabajo, por tal como viene en la figura 25 se pondrá en la terminal de Linux el comando “Whereis” para ver el directorio donde está instalado la aplicación.

```
eduardo@eduardo-MinnowBoard-MAX-D0-PLATFORM:~$ whereis build-essential
build-essential: /usr/share/build-essential
```

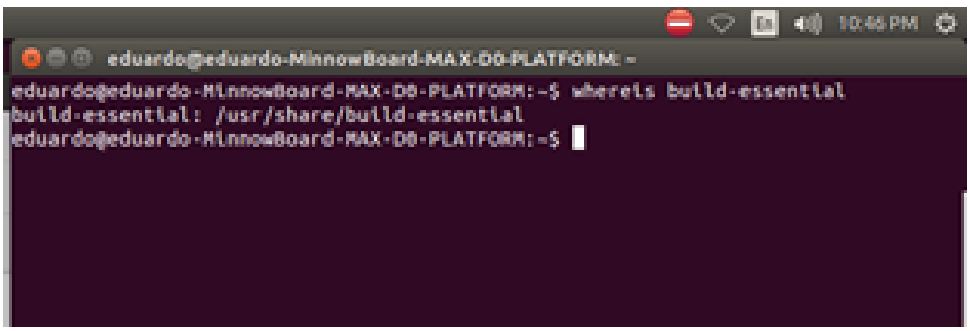


Figura 26. Ubicación de las librerías build essentials.

Teniendo instaladas dichas librerías, es necesario tener las librerías del dispositivo, estas se bajan de la página del desarrollador de Leap Motion, la cual es necesario descomprimir y correr los servicios de Leap Motion para que se pueda reconocer el driver de Leap Motion. De acuerdo al fabricante, sólo se tiene soporte en el caso de Ubuntu de la versión 12.04 en adelante, por tanto no habrá problemas si este es instalado en la versión 14.04 que se instalará en

la plataforma Minnowboard. El siguiente comando en la terminal instalará los drivers del dispositivo [11].

```
eduardo@eduardo-MinnowBoard-MAX-D0-PLATFORM:~$ sudo dpkg –install
Leap-2.3.1+31549-x64.deb
```

Si la instalación no se completó debido a la falta de la librería como se muestra en la figura 27, se debería tener una excepción para poder instalar los driver sin necesidad de tener cierta librería. Incluso, si se conecta el dispositivo sin los drivers instalados, las cámaras no funcionarían como se aprecia en la figura

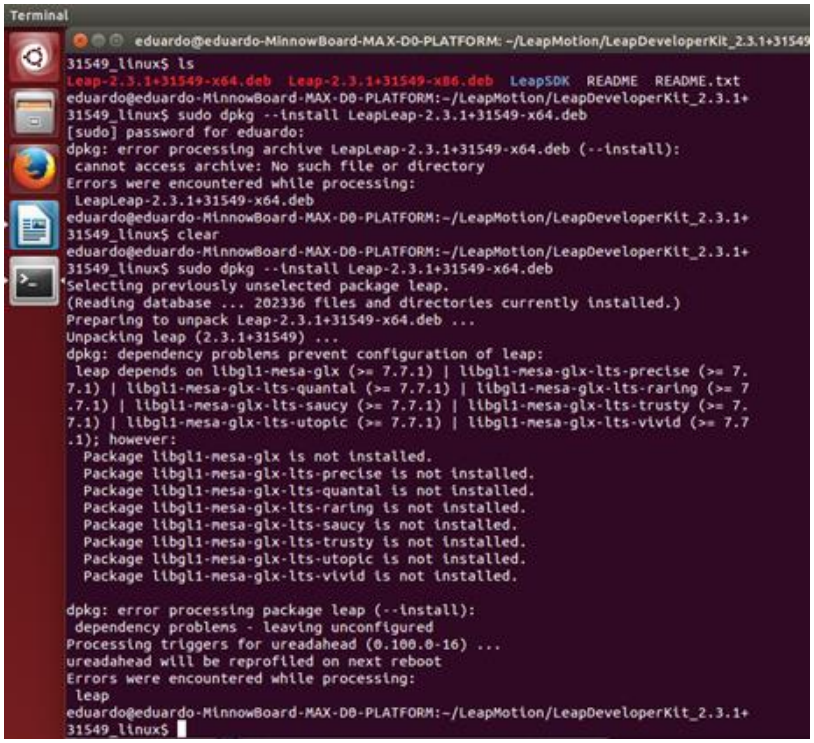


Figura 27. Instalación no terminada por falta de la librería libgl1.

Se puede correr el comando siguiente para crear excepciones que no impidan la instalación del driver, y ya creadas dichas excepciones como se muestra en la figura 28, ya se ha instalado el driver correctamente. El comando en la terminal busca instalar la librería de 64 bits que es la arquitectura del procesador de la plataforma donde corre el sistema operativo Ubuntu de este trabajo.

eduardo@eduardo-MinnowBoard-MAX-D0-PLATFORM:~$ sudo dpkg --ignore-depends=libgl1-mesa-glx --install Leap-2.3.1+31549-x64.deb

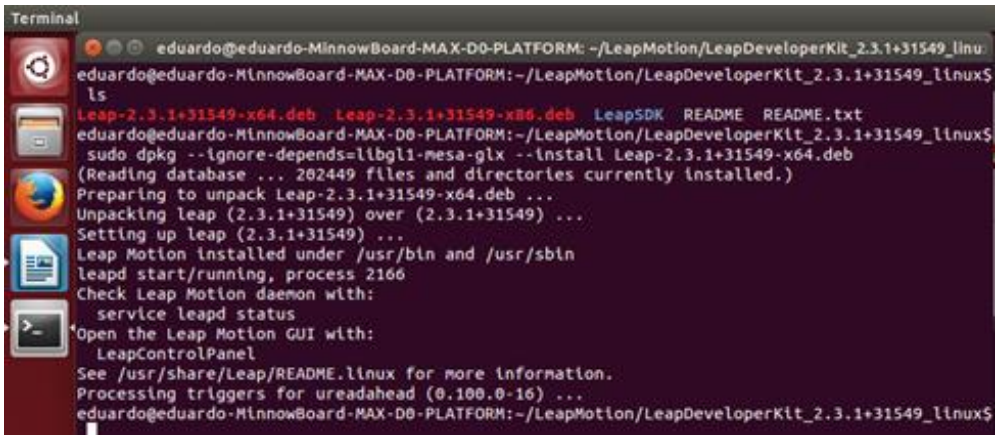


Figura 28. Drivers instalados correctamente.

En donde tenemos, que la API de C++ es dada en una librería dentro del directorio “lib” en el directorio raíz, en donde se tiene librerías para x64 y para 32 bits. En el caso para x64 que es la que se usará “`libLeap.so`”.

```
eduardo@eduardo-MinnowBoard-MAX-D0-
PLATFORM:~/LeapMotion/LeapDeveloperKit_2.3.1+31549_linux$ ls
```

Dentro de la carpeta de “Sample”, se encuentra el “Makefile” que podríamos usar como referencia para poder compilar la aplicación. A continuación, se tiene dicho Makefie:

```
OS := $(shell uname)
ARCH := $(shell uname -m)


EXECUTABLE = LeapGraphicv2
INCLUDES:=                                                              -I
/home/eduardo/LeapMotion/LeapDeveloperKit_2.3.1+31549_linux/LeapSDK/sa
mples/leapmouse-master/include

SHAREDLIBS:= -lXtst

ifeq ($(OS), Linux)
  ifeq ($(ARCH), x86_64)
```

```
    LEAP_LIBRARY = -Wl,-rpath,../../lib/x64  -L../../lib/x64 -lLeap -L
/usr/X11R6/lib -lX11
  else
    LEAP_LIBRARY = -Wl,-rpath,../../lib/x86  -L../../lib/x86 -lLeap -L
/usr/X11R6/lib -lX11
  endif

endif

SOURCES:=LeapGraphicPointer.cpp \
            LeapGraphicv2.cpp \

SHAREDLIBS:= -lXtst


LeapGraphicv2:
ifeq ($(OS), Linux)
      $(CXX)  -o  $(EXECUTABLE)  $(INCLUDES)  $(SOURCES)  $(SHAREDLIBS)
$(LEAP_LIBRARY)
endif

clean:
      rm -rf $(EXECUTABLE) $(EXECUTABLE).dSYM
```

En el Makefile mostrado anteriormente, tenemos como entradas las librerías, headers y código fuentes en C++. Los archivos de entrada son:

- LeapGraphicv2.cpp
- LeapGraphicPointer.cpp
- Leap.h
- LeapMath.h
- LeapGraphicPointer.h
- X11 libraries

Se tiene que el ejecutable resultante de compilación se llamaría "`LeapGraphicv2`".

Antes de poder compilar la aplicación es necesario tener las librerías de la aplicación X11. Dichas librerías de manera similar al compilador y las del dispositivo necesitarían estar instaladas previamente, para instalar las librerías X11/Test se puede usar el siguiente commando en la línea de commandos de Linux:

```
sudo apt-get update
sudo apt-get install libxtst-dev
```

De manera muy similar, dicha librería se instaló previamente. En este caso haremos una petición o “Query” en la terminal para ver donde quedó instalada dicha librería. Usando el mismo comando “Whereis” en la figura 29 se aprecia la dirección de la instalación hecha ya previamente.

Figura 29. Ubicación de las librerías X11.

Viendo el dispositivo conectado, este se quedaría encendido hasta que el servicio no sea deshabilitado, si no se apagase, este dispositivo suele calentarse apreciablemente no al grado de quemarse, pero si disipa mucha energía lo cual puede ser contraproducente en un ambiente automotriz donde se usa baterías cuando este permanece apagado. Para ello, hay servicios del driver de Leap Motion que permiten activar y reactivar dicho servicio cada vez que este se requiera, por ejemplo, en la figura 30, se aprecia como el servicio de Leap Motion es desactivado, y lo que por ende las cámaras de apagan como se ve en la figura . El servicio usado para desactivar el Leap Motion es:

```
eduardo@eduardo-MinnowBoard-MAX-D0-PLATFORM:~$ sudo service leapd stop
```

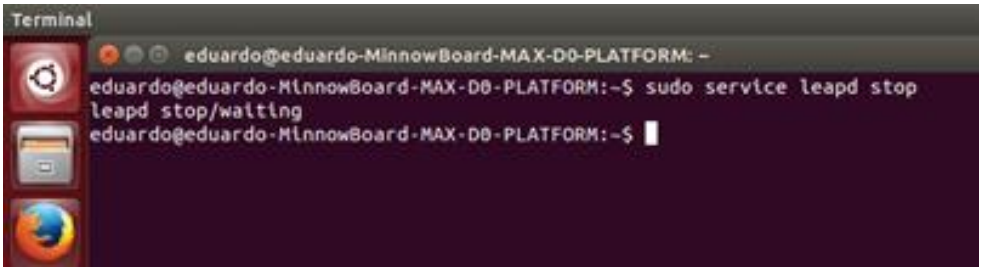


Figura 30. Desactivación del servicio "leapd".

Después de comandar la desactivación del servicio, el dispositivo apaga las cámaras.

Ahora bien, si ya está desactivado el servicio de Leap Motion puede volver a activarse mediante el query siguiente, donde la figura 31 muestra la ejecución de dicho comando.

```
eduardo@eduardo-MinnowBoard-MAX-D0-PLATFORM:~$ sudo leapd
```

Figura 31. Activación del servicio "Leapd".

Después de comandar la activación del servicio, el dispositivo prende las cámaras infrarrojas nuevamente como se aprecia en la figura 32.

Enseguida, antes de ejecutar la aplicación del apuntador, se correrán unas rutinas de diagnósticos sencillas para asegurar que cualquier otra función se ejecute adecuadamente. La primera es la rutina de diagnóstico más básica que se ejecuta con solo el visualizador nativo de gestos, de igual manera se usa para calibrar el dispositivo. Esta se llama con el siguiente comando, y al correr se ve como en la figura las manos sobre el Leap Motion se plasman en la pantalla.

```
eduardo@eduardo-MinnowBoard-MAX-D0-PLATFORM:~$ Visualizer &
```

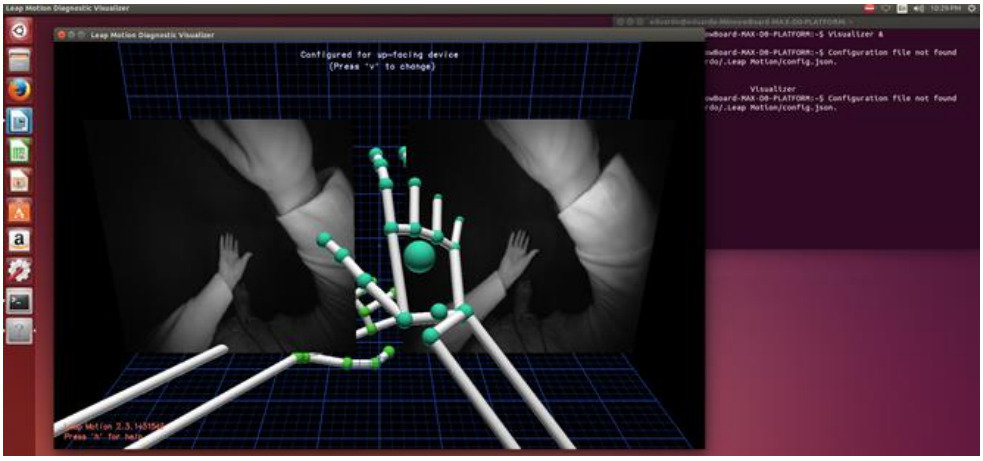


Figura 32. Manos sobre en la pantalla a través del Leap Motion.

Dados los resultados anteriores, se ve que el dispositivo esta funcionando correctamente, es decir, hay conectividad entre la plataforma y el Leap Motion. Ahora, se puede correr una aplicación no tan compleja, útil para ver de forma preliminar como puede correr una aplicación mucho más compleja como un apuntador.

La siguiente aplicación de prueba consiste en imprimir en la terminal lo siguientes parámetros enlistados a continuación [12]:

- Frame id,
- Timestamp,
- Hands,
- Fingers.

Teniendo ya el código de prueba desarrollado por el fabricante del Leap Motion, sólo habrá de compilarlo para ver los parámetros de salida arriba mencionados. Usando el “makefile” ya contenido en el SDK, se corre y se ejecuta usando los siguientes comandos como se ve en la figura 33 donde el archivo “Sample” resaltado en verde es el resultado de la compilacion:

```
eduardo@eduardo-MinnowBoard-MAX-D0-
PLATFORM:~/LeapMotion/LeapDeveloperKit_2.3.1+31549_linux/LeapSDK/sampl
es$ make

eduardo@eduardo-MinnowBoard-MAX-D0-
PLATFORM:~/LeapMotion/LeapDeveloperKit_2.3.1+31549_linux/LeapSDK/sampl
es$ ./Sample
```

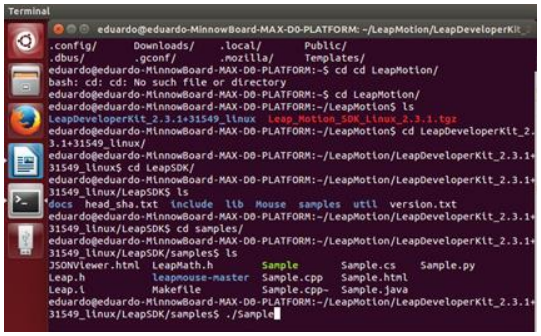


Figura 33. Archivo "Sample" creado después de compilar la aplicación de prueba.

La última instrucción ejecutó el ejemplo, en el cual en la figura se corrió con una la mano izquierda, en la figura se corrió con la derecha y en la tercera con ambas manos. Teniendo como resultado en la terminal la siguiente información que de igual manera se puede constatar en la figura 34.

```
Frame id: 11387, timestamp: 1431302814551231, hands: 1, extended fingers: 5, tools: 0,
gestures: 0
  Left hand, id: 2, palm position: (-18.2001, 156.44, -1.2156)
  pitch: 37.2156 degrees, roll: 34.1974 degrees, yaw: -15.9856 degrees
  Arm direction: (0.21672, 0.504499, -0.835771) wrist position: (-14.9802, 126.163,
39.2662) elbow position: (-69.1603, 0.0382027, 248.209)
    Thumb finger, id: 20, length: 48.2559mm, width: 18.75
      Metacarpal bone, start: (13.0858, 130.481, 38.5867), end: (13.0858, 130.481,
38.5867), direction: (0, 0, 0)
      Proximal bone, start: (13.0858, 130.481, 38.5867), end: (28.4495, 153.974,
1.86682), direction: (-0.332405, -0.508275, 0.794458)
      Middle bone, start: (28.4495, 153.974, 1.86682), end: (34.3946, 166.182, -
26.6336), direction: (-0.188313, -0.386711, 0.902769)
      Distal bone, start: (34.3946, 166.182, -26.6336), end: (44.6763, 179.518, -
40.2727), direction: (-0.474468, -0.615415, 0.6294)
    Index finger, id: 21, length: 54.4514mm, width: 17.91
      Metacarpal bone, start: (-3.59858, 144.492, 41.2457), end: (-4.07753, 182.524, -
15.2669), direction: (0.00703103, -0.558309, 0.829603)
      Proximal bone, start: (-4.07753, 182.524, -15.2669), end: (-1.10736, 183.499, -
54.9239), direction: (-0.0746649, -0.0245077, 0.996907)
      Middle bone, start: (-1.10736, 183.499, -54.9239), end: (2.78348, 179.875, -
76.6632), direction: (-0.173854, 0.161891, 0.971373)
      Distal bone, start: (2.78348, 179.875, -76.6632), end: (6.50525, 175.424, -
91.3808), direction: (-0.235257, 0.28138, 0.930311)
    Middle finger, id: 22, length: 62.043mm, width: 17.59
      Metacarpal bone, start: (-14.3949, 141.739, 38.8272), end: (-22.7122, 174.087, -
16.4681), direction: (0.12875, -0.500749, 0.855964)
      Proximal bone, start: (-22.7122, 174.087, -16.4681), end: (-21.463, 174.877, -
61.0736), direction: (-0.0279903, -0.0176839, 0.999452)
      Middle bone, start: (-21.463, 174.877, -61.0736), end: (-17.3098, 171.404, -
86.8411), direction: (-0.157735, 0.131872, 0.978637)
      Distal bone, start: (-17.3098, 171.404, -86.8411), end: (-13.1046, 167.407, -
103.245), direction: (-0.241678, 0.229746, 0.942766)
    Ring finger, id: 23, length: 59.656mm, width: 16.738
      Metacarpal bone, start: (-23.9939, 136.244, 36.1691), end: (-38.9769, 160.845, -
14.1729), direction: (0.258327, -0.424149, 0.867966)
      Proximal bone, start: (-38.9769, 160.845, -14.1729), end: (-34.5938, 165.621, -
55.0318), direction: (-0.105948, -0.115452, 0.987647)
      Middle bone, start: (-34.5938, 165.621, -55.0318), end: (-27.0936, 163.087, -
79.4296), direction: (-0.292405, 0.0987794, 0.951179)
      Distal bone, start: (-27.0936, 163.087, -79.4296), end: (-20.0864, 158.95, -
94.6962), direction: (-0.40504, 0.239173, 0.882462)
    Pinky finger, id: 24, length: 46.7692mm, width: 14.868
      Metacarpal bone, start: (-30.0251, 125.188, 32.2901), end: (-51.0866, 145.202, -
12.8594), direction: (0.39228, -0.372767, 0.840929)
      Proximal bone, start: (-51.0866, 145.202, -12.8594), end: (-53.8065, 145.213, -
45.4862), direction: (0.083077, -0.00032764, 0.996543)
      Middle bone, start: (-53.8065, 145.213, -45.4862), end: (-51.2807, 142.904, -
63.27), direction: (-0.139473, 0.127487, 0.981985)
      Distal bone, start: (-51.2807, 142.904, -63.27), end: (-46.7327, 139.564, -
78.1992), direction: (-0.284961, 0.209272, 0.935416)
```

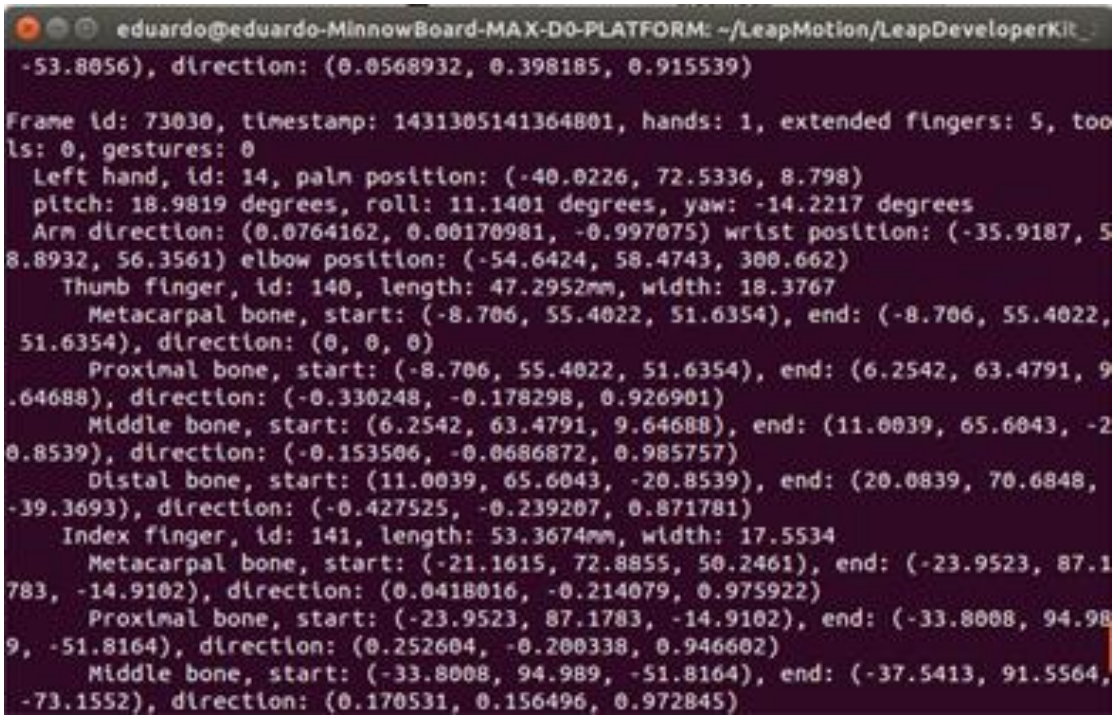


Figura 34. Ventana de terminal con los parámetros de la mano izquierda impresos por Leap Motion.

Bien, de manera empírica se demostró que el dispositivo se comunica correctamente con la plataforma y además arroja información que se puede reusar para nuestros propósitos como: el “frame_id”, números de gestos, dedos involucrados y número de manos en el mismo “frame”. Estos elementos se usarán en las estructuras de control de la aplicación propuesta.

Hasta el momento se ha instalado ya todo para poder compilar la aplicación y el código de la aplicación esta disponible.

Configuradas ya las rutas para usar las librerías como se mostró en el Makefile, se ejecuta el makefile con las siguientes opciones dentro de la terminal:

```
make clean
nice make
```

No sin antes ir a la ubicación del proyecto, y constatando hasta el momento la inexistencia del ejecutable. Usando el comando “cd” para movernos en los directorios de Linux llegamos al directorio del proyecto y listamos con el comando “ls” el contenido actual, es decir, antes de correr el makefile como se muestra en la figura donde no está el ejecutable que se llamaría como dice el makefile: Leapgraphicv2. Recordando que los archivos ejecutables según el tipo de Shell que estamos manejando se distinguen en color verde.

```
eduardo@eduardo-MinnowBoard-MAX-D0-
PLATFORM:~/LeapMotion/LeapDeveloperKit_2.3.1+31549_linux/LeapSDK/samples/leapmouse-
master$ ls

include  LeapGraphicPointer.cpp  LeapGraphicv2.cpp  LeapGraphicv2.cpp~  Makefile
Makefile~  rebuild.sh

eduardo@eduardo-MinnowBoard-MAX-D0-
PLATFORM:~/LeapMotion/LeapDeveloperKit_2.3.1+31549_linux/LeapSDK/samples/leapmouse-
master$
```

Figura 35. Listado del contenido antes de compilar la aplicación.

Si la compilación fue sin algún error, se tiene el ejecutable llamado como se propuso en el “Makefile”: “`Leapgraphicv2`”. En la figura 36, se puede constatar que la compilación salió sin ningún “warning” ni error, por lo que después, el ejecutable del código fue creado y en la figura XX usando el mismo comando “ls” en el directorio de trabajo puede apreciarse la creación del ejecutable resaltando un archivo color verde.

```
eduardo@eduardo-MinnowBoard-MAX-D0-
PLATFORM:~/LeapMotion/LeapDeveloperKit_2.3.1+31549_linux/LeapSDK/samples/leapmouse-
master$ make clean

rm -rf Leapgraphicv2 Leapgraphicv2.dSYM
```

Figura 36. Compilación exitosa de la aplicación.

Al ejecutar el archivo resultante de la compilación se tiene:

```
eduardo@eduardo-MinnowBoard-MAX-D0-
PLATFORM:~/LeapMotion/LeapDeveloperKit_2.3.1+31549_linux/LeapSDK/samples/leapmouse-
master$ ./Leapgraphicv2
```

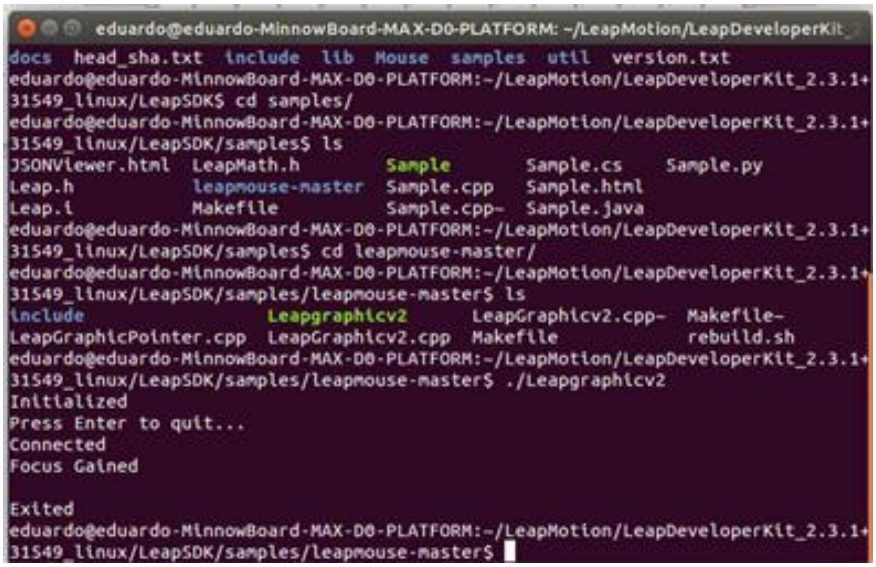


Figura 37. Creación y ejecución del ejecutable de la aplicación.

4 CAPÍTULO IV: CONCLUSIONES Y TRABAJO FUTURO

El presente trabajo de investigación tuvo como objetivo usar las librerías del dispositivo Leap Motion en un ambiente Linux sobre la plataforma embebida Minnowboard Max II dado los resultados obtenidos fue posible utilizarlas con dicho dispositivo. Así mismo se observó que dichas librerías o DLLs se enlazan correctamente con el dispositivo es necesario un esfuerzo mucho mayor para mejorar la experiencia del usuario lo cual podría desembocar en uno o más trabajos de investigación para proponer mejoras en su uso mediante algoritmos mucho más eficientes, y aunado a ello también mejorar el hardware para reducir algunas cuestiones de procesamiento, así como la disipación de temperatura que representó una situación no considerada en un inicio y podría generar otros conflictos en relación a regulaciones en un mercado como el automotriz que debe cumplir estándares de seguridad más estrictos que una aplicación de propósito general. Pero, pese a lo anterior, este tipo de tecnologías aumentaría la comodidad del pasajero y mejoraría la manera en el usuario realiza la conducción, lo cual puede ser otro tema de investigación.

Ahora bien, en cuanto a darle continuidad al trabajo mostrado en este trabajo de investigación una buena iniciativa sería integrarlo a un sistema operativo basado en Linux, es decir, fuera de un a distribución comercial como la versión Ubuntu 14.04 usada para este trabajo. De igual manera, otra propuesta a futuro sería controlar otras funciones de mismo automóvil como lo son por ejemplo el elevador de los vidrios, encendido o apagado del automóvil o viéndonos un poco más innovadores el control de las velocidades de una transmisión automática. Para lo anterior, se puede seguir usando el mismo sensor Leap Motion y la misma plataforma, sin embargo, puede que el fabricante puede extender el soporte al sensor a otras plataformas ajenas a los procesadores Intel (o AMD) como el ya famoso x_64_86 de AMD/Intel, dichas plataformas son las son la pujante en el mundo de los embebidos ARM, y por supuesto en una plataforma lo suficientemente robusta para ofrecer prestaciones similares a la ofrecida por Intel/AMD. Además de lo anterior, para

satisfacer otra propuestas como controlar otros actuadores tales el elevador de vidrios, sería necesario comunicarse con la red automotriz y para ello habría que tener un transceiver ya sea de CAN (High o Low speed) o LIN, y así aplicaciones similares. . También otro punto a poder refinar para un siguiente tema de investigación es el uso de filtros para mejorar la experiencia con el apuntador y robustecer la funcionalidad. Por último, como trabajo futuro también podemos considerar un área de mejora, esto es la automatización a través del “scripting” del Shell de Linux; tareas como el verificar sí un paquete de librería como las de “X-Windows” (X11/Test) y su posterior instalación podría ser con un script del Shell de Linux.

5 BIBLIOGRAFÍA.

[1]https://community.leapmotion.com/t/driver-for-raspbian/2788

[2]http://elinux.org/Minnowboard:MinnowMaxDistros#Ubuntu_14.04_LTS_Desktop_AMD64

[3] Ingeniería de Software. Sommerville, Ian. 2011. Addison Wesly, México.774 páginas.

[4] Cursos Sistemas operativos avanzados.Verano 2016. Alfredo Soto Villegas. Embedded Linux - Parte 2: Linux Kernel and Embedded Linux System

[5] https://developer.leapmotion.com/documentation/cpp/api/Leap.Gesture.html

[6]https://developer.leapmotion.com/documentation/cpp/devguide/Leap_Hand.html

[7]https://developer.leapmotion.com/documentation/cpp/devguide/Leap_Coordinate_Mapping.html

[8]https://developer.leapmotion.com/documentation/cpp/devguide/Leap_Architecture.html

[9] http://www.elinux.org/Minnowboard:MaxBios#32-bit_vs._64-bit_UEFI

[10]https://community.leapmotion.com/t/driver-for-raspbian/2788

[11] https://support.leapmotion.com/hc/en-us/articles/223782608-Linux-Installation

[12]https://developer.leapmotion.com/documentation/cpp/devguide/Sample_Tutorial.html

6 EQUIPO DE TRABAJO.

El equipo de trabajo está formado por los siguientes integrantes:

- Juan Eduardo López Flores (Continental) Estudiante.
- Juan Ley (ITESO) Profesor a cargo de la investigación.
- Luis Enrique González Jiménez (ITESO) Profesor a cargo de la investigación.

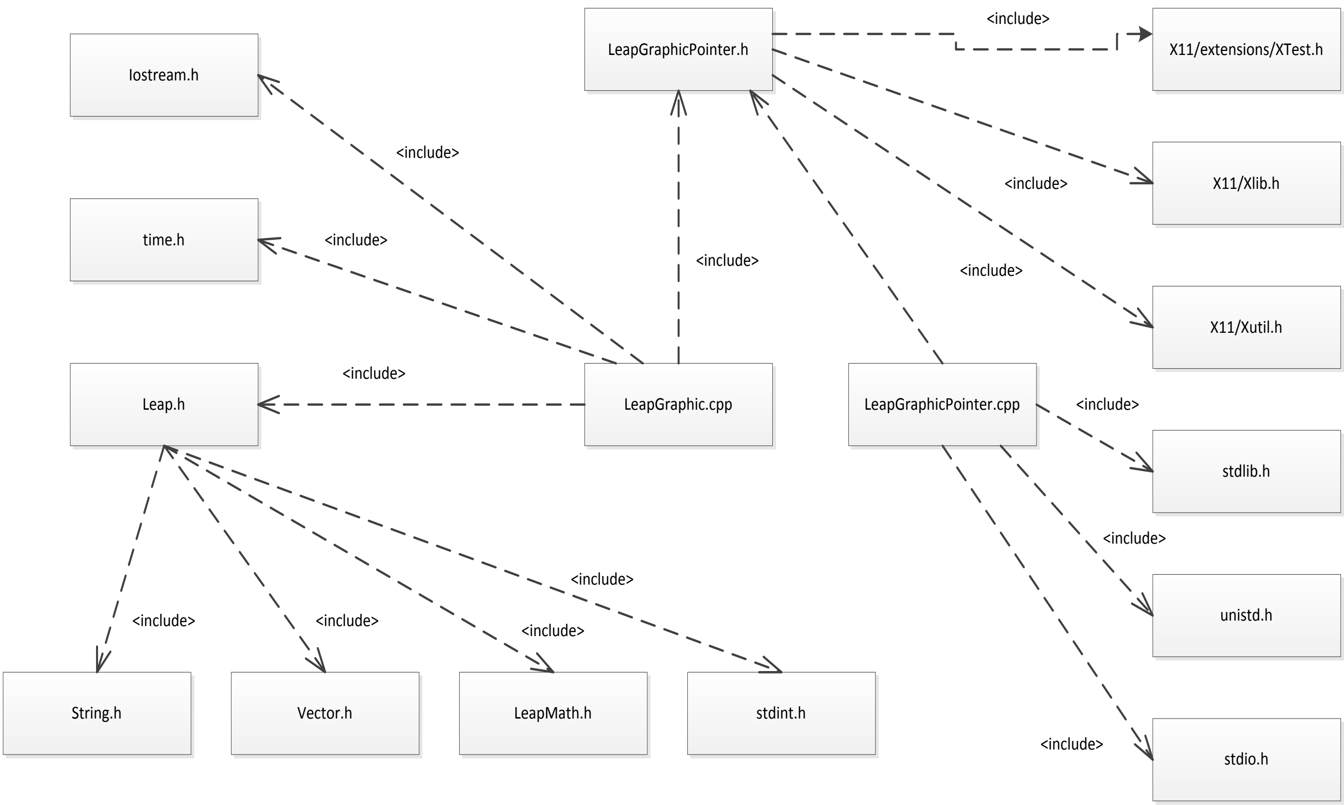
7 APÉNDICE

7.1 Archivos:

El código está comprendido de los siguientes archivos:

7.2 Estructura de archivos.

La relación y llamados entre cada archivo se muestran a continuación, recordando que la inclusión de cada clase es a donde apunta la flecha.



7.3 Código

Se pone de manera íntegra el contenido de todos los archivos mencionados en el apartado a 7.1

7.3.1 Leap.h

Se puede descargar de la página del desarrollador del Leap Motion. La versión usada es la 2.3.1+31549

7.3.2 LeapMath.h

Se puede descargar de la página del desarrollador del Leap Motion. La versión usada es la 2.3.1+31549

7.3.3 LeapGraphicPointer.h

```
#ifndef __LEAPGRAPHICPOINTER_H__
#define __LEAPGRAPHICPOINTER_H__

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/extensions/XTest.h>
#include <unistd.h>


/*****************************************************************
*
*************************Constants
definition************************
*****************************************************************
/
#ifndef TIME_OUT
#define TIME_OUT 300 // seconds wait before going to sleep
#endif

#ifndef DEBOUNCE_CLICK_SENSIBILITY
#define DEBOUNCE_CLICK_SENSIBILITY 10 // seconds wait before going to
sleep
#endif

/*****************************************************************
*
*************************Classes
definition*************************
*****************************************************************
/

class LeapGraphicPointer
{
      public:
            LeapGraphicPointer();
            virtual ~LeapGraphicPointer(){};
            void vfnMovmt(int x,int y);
```

```
            void vfnLeftKeyPress();
            void vfnLeftKeyRelease();
            void vfnScrollUp(); //Button_4
            void vfnScrollDown();  //Button_5
            void vfnClose();
            Screen * ptrScreen;
            int rScreenWidth;
            int rScreenHeight;

      private:
            Display * ptrDisplay;
};

#endif
```

7.3.4 LeapGraphicPointer.cpp

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include "LeapGraphicPointer.h"

/*
#include <X11/extensions/XTest.h>
int XTestFakeButtonEvent(display, button, is_press, delay);

Display *display;
unsigned int button;
Bool is_press;
unsigned long delay;


int XTestFakeMotionEvent(display, screen_number, x, y, delay);

Display *display;
int screen_number;
int x, y;
unsigned long delay;
*/

LeapGraphicPointer::LeapGraphicPointer()
{
      this->ptrDisplay= XOpenDisplay(NULL);
    ptrScreen = DefaultScreenOfDisplay( ptrDisplay );
    rScreenWidth = ptrScreen->width;
      rScreenHeight = ptrScreen->height;
}

void LeapGraphicPointer::vfnMovmt(int x ,int y )
{
      XTestFakeMotionEvent(this->ptrDisplay, -1, x, y, 0);
      XFlush(this->ptrDisplay);
}

void LeapGraphicPointer::vfnLeftKeyPress()
{
      XTestFakeButtonEvent(this->ptrDisplay, 1, true, 0); // int
XTestFakeButtonEvent(display, button, is_press, delay);
      XFlush(this->ptrDisplay);
```

```
}

void LeapGraphicPointer::vfnLeftKeyRelease()
{
      XTestFakeButtonEvent(this->ptrDisplay, 1, false, 0); // int
XTestFakeButtonEvent(display, button, is_press, delay);
      XFlush(this->ptrDisplay);
}

void LeapGraphicPointer::vfnScrollUp()
{
      XTestFakeButtonEvent(this->ptrDisplay, 4, true, 0); // int
XTestFakeButtonEvent(display, button, is_press, delay);
      XTestFakeButtonEvent(this->ptrDisplay, 4, false, 0); // int
XTestFakeButtonEvent(display, button, is_press, delay);
      XFlush(this->ptrDisplay);
}
void LeapGraphicPointer::vfnScrollDown()
{
      XTestFakeButtonEvent(this->ptrDisplay, 5, true, 0); // int
XTestFakeButtonEvent(display, button, is_press, delay);
      XTestFakeButtonEvent(this->ptrDisplay, 5, false, 0); // int
XTestFakeButtonEvent(display, button, is_press, delay);
      XFlush(this->ptrDisplay);
}

void LeapGraphicPointer::vfnClose()
{
      XCloseDisplay(this->ptrDisplay);
}
```

7.3.5 LeapGraphicv2.cpp

```
#include <iostream>
#include <time.h>

#include
"/home/eduardo/LeapMotion/LeapDeveloperKit_2.3.1+31549_linux/LeapSDK/i
nclude/Leap.h"
#include
"/home/eduardo/LeapMotion/LeapDeveloperKit_2.3.1+31549_linux/LeapSDK/i
nclude/PointerDefinition.h"

using namespace Leap;

class SampleListener : public Listener
{
  public:
    virtual void onInit(const Controller&);          //Called once,
when this Listener object is newly added to a Controller.
    virtual void onConnect(const Controller&);       //Called when the
Controller object connects to the Leap Motion software and the Leap
Motion hardware device is plugged in, or when this Listener object is
added to a Controller that is already connected.
    virtual void onDisconnect(const Controller&);    //Called when the
Controller object disconnects from the Leap Motion software or the
Leap Motion hardware is unplugged.
    virtual void onExit(const Controller&);          //Called when
this Listener object is removed from the Controller or the Controller
instance is destroyed.
```

```
    virtual void onFrame(const Controller&);         //Called when a
new frame of hand and finger tracking data is available.
    virtual void onFocusGained(const Controller&);   //Only the
foreground application receives tracking data from the Leap Motion
Controller. This function is only called when the controller object is
in a connected state
    virtual void onFocusLost(const Controller&);     //Called when
this application loses the foreground focus.

      LeapGraphicPointer* ptrSelectionPointer;

      int rCurrStatusIndicator;
    int rClickFilterCount;
    int rPressedDebounceDelay;
    int rPressStatus;
    int rSideIndicator;
    int rClickMovement;

    bool bActiveFlag;         // Is the device currently active

    int64_t wLastFrameId;
    int64_t wLastSavedId;     // frame id of the last frame we toggled
activity
    time_t wLastTimeEvent;    // epoch time in seconds of last event

      float fAbsValueInX;     // Compare directions and give preference
to the greatest linear movement in X axis.
      float fAbsValueInY;     // Compare directions and give preference
to the greatest linear movement in Y axis.

};

/************************************************************************
*******************************

Function : SampleListener::onInit
Called once, when this Listener object is newly added to a Controller.

************************************************************************
*******************************/
void SampleListener::onInit(const Controller& controller)
{
      std::cout << "Initialized" << std::endl;
}

/************************************************************************
*******************************

Function : SampleListener::onConnect
Called once, when this Listener object is newly added to a Controller.

************************************************************************
*******************************/
void SampleListener::onConnect(const Controller& controller)
{
      controller.enableGesture(Gesture::TYPE_CIRCLE,bool enable =
true);            //Enables or disables reporting of a specified
gesture type. A circular movement by a finger.
```

```
      controller.enableGesture(Gesture::TYPE_KEY_TAP,bool enable =
true);      //Enables or disables reporting of a specified gesture
type. A downward tapping movement by a finger.
      controller.enableGesture(Gesture::TYPE_SWIPE,bool enable =
true);            //Enables or disables reporting of a specified
gesture type. A straight line movement by the hand with fingers
extended.
      controller.enableGesture(Gesture::TYPE_SCREEN_TAP,bool enable =
true);      //Enables or disables reporting of a specified gesture
type.
      std::cout << "Connected" << std::endl;
                        //See APIs definition guide
}

/*****************************************************************
*******************************

Function : SampleListener::onDisconnect
Called when the Controller object disconnects from the Leap Motion
software.
The controller can disconnect when the Leap Motion device is
unplugged, the user shuts -
the Leap Motion software down, or the Leap Motion software encounters
an unrecoverable error.

*****************************************************************
*******************************/
void SampleListener::onDisconnect(const Controller& controller)
{
      controller.enableGesture(Gesture::TYPE_CIRCLE, bool enable =
false);
      controller.enableGesture(Gesture::TYPE_KEY_TAP, bool enable =
false);
      controller.enableGesture(Gesture::TYPE_SWIPE, bool enable =
false);
      controller.enableGesture(Gesture::TYPE_SCREEN_TAP, bool enable =
false);
      std::cout << "Disconnected" << std::endl;
}

/*****************************************************************
*******************************

Function : SampleListener::onExit
Called when this Listener object is removed from the Controller or the
Controller instance is destroyed.

*****************************************************************
*******************************/
void SampleListener::onExit(const Controller& controller)
{
      ptrSelectionPointer->vfnClose();
      delete ptrSelectionPointer;
      std::cout << "Exited" << std::endl;
}

/*****************************************************************
*******************************
```

```
Function : SampleListener::onFrame
Called when a new frame of hand and finger tracking data is available.
Access the new frame data using the Controller::frame() function.

*************************************************************************
*****************************/
void SampleListener::onFrame(const Controller& controller)
{


      // Get the most recent frame and report some basic information
      const Frame frame = controller.frame(); //Creating "Frame"
object named "frame"

      std::cout << "[Debbug screen] New frame available" << std::endl;

      //It can be determines if any pending onFrame events were
skipped by comparing the ID of the most recent frame with the ID of
the last received frame
      if (frame.id() == this->wLastFrameId)
      return;
      this->wLastFrameId = frame.id();

      // TIME_OUT after that go to sleep.
      if ((TIME_OUT + this->wLastTimeEvent) < time(0) && this-
>bActiveFlag)
      {
            std::cout << "[Debbug screen] Going to standby due to
inactivity" << std::endl;
            this->bActiveFlag = false; //Set Active flag to inactive
      }

      if (!this->bActiveFlag)
      return;

      //Left key press action
      if (frame.fingers().count() == 1 & rSideIndicator != 2)
      {
            PointableList pointables = frame.pointables();
            InteractionBox oIntBox = frame.interactionBox();

            for (int rIndexPointer = 0; rIndexPointer <
pointables.count(); ++rIndexPointer)
            {
              Pointable pointable = pointables[rIndexPointer];
              Vector vNormalizedPosPoint =
oIntBox.normalizePoint(pointable.stabilizedTipPosition());
              float fDistance = pointable.touchDistance();
              float x = vNormalizedPosPoint.x * (ptrSelectionPointer->
rScreenWidth + 250);
              float y = (ptrSelectionPointer-> rScreenHeight + 250) -
vNormalizedPosPoint.y * (ptrSelectionPointer-> rScreenHeight + 250);

              if (rSideIndicator == 0)
              {
                    ptrSelectionPointer->vfnMovmt((int)x, (int)y);
                    rPressedDebounceDelay = 0;
              }
```

```
                  if (fDistance < 0)
                  {
                        rClickFilterCount++;
                        rSideIndicator = 1;
                        if (rClickFilterCount >
DEBOUNCE_CLICK_SENSIBILITY)
                        {
                          if (rPressStatus == 0)
                          {
                              ptrSelectionPointer->vfnLeftKeyPress();
                              rPressStatus = 1;
                          }
                          else
                          {
                              if (rClickFilterCount >= 30)
                              {
                                if (rClickMovement == 0)
                                    ptrSelectionPointer ->
vfnMovmt((int)x,(int)y);

                                std::cout << "[Debbug screen] move
side: "<< rSideIndicator << std::endl;
                                rClickFilterCount = 20;
                              }
                          }
                        }
                  }

                  else if (rSideIndicator == 1)
                  {
                        ptrSelectionPointer->vfnLeftKeyRelease();
                        std::cout << "[Debug screen] released side: "
<< rSideIndicator << std::endl;
                        rSideIndicator = 0;
                        rPressStatus = 0;
                        rPressedDebounceDelay = 0;
                        rClickFilterCount = 0;
                        rClickMovement = 0;
                  }
            }
            this->wLastTimeEvent = time(0);
      }

      //Swipe Movement
      else if (frame.fingers().count() > 1 & rSideIndicator != 1)
      {
            rClickFilterCount = 0;

            const GestureList gestures = frame.gestures(); // Get
gestures

                  for (int rGestureIndex2 = 0; rGestureIndex2 <
gestures.count(); ++rGestureIndex2)
                  {
                    Gesture gesture = gestures[rGestureIndex2];
                        switch (gesture.type() && gesture.isValid())
                        {
                          case Gesture::TYPE_CIRCLE :
                          //No action
```

```
                        break;

                        case Gesture::TYPE_SWIPE :
                        {
                            rSideIndicator = 2;
                            SwipeGesture sgSwipeMovement = gesture;
                            Vector vSwipeScrollScreen =
sgSwipeMovement.direction();

                            fAbsValueInX =
fabs(sgSwipeMovement.direction().x);
                            fAbsValueInY =
fabs(sgSwipeMovement.direction().y);


                            if (fAbsValueInX > fAbsValueInY) // Was X
the most significative movement?
                            {
                                  if (sgSwipeMovement.direction().x >
0)
                                  {
                                        std::cout << "Right" <<
std::endl;
                                  }
                                  else
                                  {
                                        std::cout << "Left" <<
std::endl;
                                  }
                            }

                            else
      // Was Y the most significative movement?
                            {
                                  if (sgSwipeMovement.direction().y >
0)
                                  {
                                        this->rCurrStatusIndicator =
1; //scroll up movement
                                        std::cout << "Up" <<
std::endl;
                                  }
                                  else
                                  {
                                        this->rCurrStatusIndicator =
2; //scroll down movement
                                        std::cout << "Down" <<
std::endl;
                                  }
                            }

                            if (sgSwipeMovement.state() ==
STATE_STOP) //STATE_STOP    = 3,  /**< The gesture has completed or
stopped. */
                            {
                              if (this->rCurrStatusIndicator == 1)
                              {
                                  rSideIndicator = 0;
                                  this->rCurrStatusIndicator = 0;
```

```
                                    ptrSelectionPointer->vfnScrollUp();
                                }
                                else if (this->rCurrStatusIndicator ==
2)
                                {
                                    rSideIndicator = 0;
                                    this->rCurrStatusIndicator = 0;
                                    ptrSelectionPointer-
>vfnScrollDown();
                                }
                            }
                            break;
                        }

                        default:
                        //Do nothing
                        break;
                    }
                }
            this->wLastTimeEvent = time(0);
      }

      std::cout << "[Debug screen] New frame available" << std::endl;

      if (!frame.hands().isEmpty() || !gestures.isEmpty())
      {
            std::cout << std::endl;
      }
}

/*****************************************************************
*******************************

Function : SampleListener::onFocusGained
Only the foreground application receives tracking data from the Leap
Motion Controller. This function is only called when the controller
object is in a connected state.

*****************************************************************
*******************************/
void SampleListener::onFocusGained(const Controller& controller)
{
  std::cout << "Focus Gained" << std::endl;
}

/*****************************************************************
*******************************

Function : SampleListener::onFocusLost
Only the foreground application receives tracking data from the Leap
Motion Controller.
This function is only called when the controller object is in a
connected state.

*****************************************************************
*******************************/
void SampleListener::onFocusLost(const Controller& controller)
{
  std::cout << "Focus Lost" << std::endl;
```

```
}

/******************************************************************************
*******************************

Function : Main Function
This function starts the Leap Motion application.

******************************************************************************
*******************************/

int main()
{

  SampleListener listener;
  Controller controller;

  listener.ptrSelectionPointer = new LeapGraphicPointer();
  listener.rCurrStatusIndicator = 0;
  listener.rClickFilterCount = 0;
  listener.rPressStatus = 0;
  listener.rSideIndicator = 0;
  listener.rPressedDebounceDelay = 0;
  listener.rClickMovement = 0;
  listener.bActiveFlag = false;

  listener.wLastFrameId = -1;
  listener.wLastSavedId = 0;
  listener.wLastTimeEvent = time(0);

  // Have the sample listener receive events from the controller
  controller.addListener(listener);

  // Keep this process running until the carriage return is received
by the console
  std::cout << "Press Enter to quit..." << std::endl;
  std::cin.get();

  // Remove the sample listener when done
  controller.removeListener(listener);

  return 0;
}
```

7.3.6 Makefile

```
OS := $(shell uname)
ARCH := $(shell uname -m)


EXECUTABLE = LeapGraphicv2
INCLUDES:= -I
/home/eduardo/LeapMotion/LeapDeveloperKit_2.3.1+31549_linux/LeapSDK/sa
mples/leapmouse-master/include

SHAREDLIBS:= -lXtst

ifeq ($(OS), Linux)
  ifeq ($(ARCH), x86_64)
```

```
    LEAP_LIBRARY = -Wl,-rpath,../../lib/x64  -L../../lib/x64 -lLeap -L
/usr/X11R6/lib -lX11
  else
    LEAP_LIBRARY = -Wl,-rpath,../../lib/x86  -L../../lib/x86 -lLeap -L
/usr/X11R6/lib -lX11
  endif

endif

SOURCES:=LeapGraphicPointer.cpp \
            LeapGraphicv2.cpp \

SHAREDLIBS:= -lXtst


LeapGraphicv2:
ifeq ($(OS), Linux)
      $(CXX) -o $(EXECUTABLE) $(INCLUDES) $(SOURCES) $(SHAREDLIBS)
$(LEAP_LIBRARY)
endif

clean:
      rm -rf $(EXECUTABLE) $(EXECUTABLE).dSYM
```