UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Registro de datos de una plataforma robotizada mediante una arquitectura específica

Autor: Víctor Chamizo Álvarez

20/06/2013

Director: Enric Xavier Martin Rull

Departamento: Enginyeria de Sistemes, Automàtica i Informàtica Industrial (ESAII)

Titulación: Grado en Ingeniería Informática

Especialidad: Ingeniería de Computadores

Centro: FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)

Universidad: UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) – BarcelonaTech

1 Índice

2 Índice de ilustraciones

3 Índice de tablas

4 Resumen

Este proyecto se enmarca como una funcionalidad para el proyecto iWalker. El proyecto iWalker consiste en un prototipo de andador inteligente que va más allá de los convencionales. Este andador puede enfocarse para ser usado tanto en el área de rehabilitación de los hospitales, como para uso doméstico ayudando a la movilidad y la rehabilitación. La funcionalidad que concretamente se trata en este trabajo de fin de grado, es la de utilizar el sensor Microsoft Kinect para extraer el movimiento de las piernas de un usuario, y enviar estos datos a un servidor remoto sincronizándolos con los datos del iWalker.

En el proyecto se pueden diferenciar tres etapas. La primera corresponde con la asignatura de gestión de proyectos, en la cual se llevó a cabo toda la tarea de planificación y proyección. En la segunda etapa se procedió con el análisis de las posibles plataformas y su elección. Puesto que la plataforma inicial marcada por el por el andador acabó no siendo viable, hubo que investigar varias alternativas. La tercera etapa corresponde a la implementación, en ella se desarrolló todo el conjunto de procedimientos de visión por computador para extraer el movimiento de las piernas y enviarlo al servidor remoto.

El resultado final del proyecto es satisfactorio, puesto que se cumplen con los objetivos inicialmente establecidos; sin embargo, quedan como tareas futuras algunas optimizaciones de la solución actual, y la mejora en la representación de los datos obtenidos. Otro aspecto a tener en cuenta es que a comienzos de Junio, con el proyecto prácticamente terminado, se contó con un nuevo sensor 3D que podría abrir las puertas al enfoque inicial; utilizando la plataforma Raspberry Pi. Por lo tanto este proyecto se considera prácticamente cerrado para la plataforma tratada, pero en cierto modo con continuidad gracias al nuevo dispositivo.

5 Descripción del proyecto

5.1 Objetivos

El objetivo principal del proyecto es realizar un software que permita:

1. Recibir datos de un robot móvil y publicarlos en internet, vía un dispositivo WiFi haciendo inserciones en una base de datos.
2. Hacer un log de esos datos en una unidad de almacenamiento.
3. Conectar un sensor Kinect al sistema a bordo y sincronizar (registrar) los datos con los del robot.

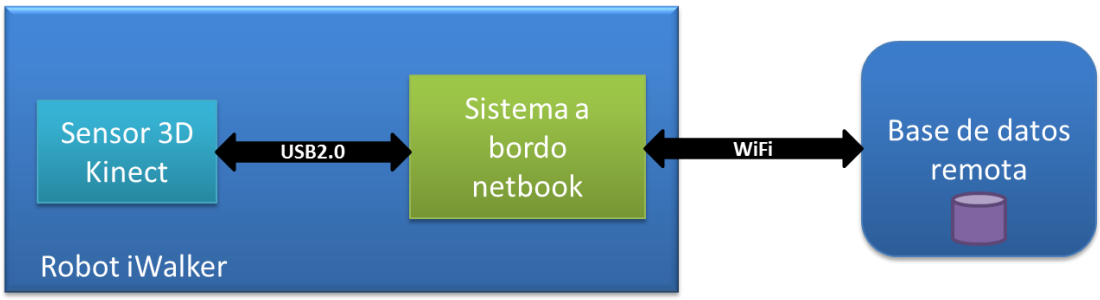


Ilustración 1 Diagrama de bloques general

5.2 Contexto

Este proyecto es una característica del proyecto iWalker. Éste es un andador inteligente que permite la comunicación con el usuario, la toma de decisiones y contiene un conjunto de programas inteligentes. La versatilidad de este andador permite adaptar el producto a las necesidades del usuario y al mercado al que se puede dirigir. Este mercado puede ser tanto el área de rehabilitación de hospitales, o el uso doméstico para la ayuda a la movilidad o de rehabilitación, ya que permite compensar la pérdida sensorial, motora y las funciones cognitivas provocadas por el paso del tiempo y las enfermedades de los ancianos.

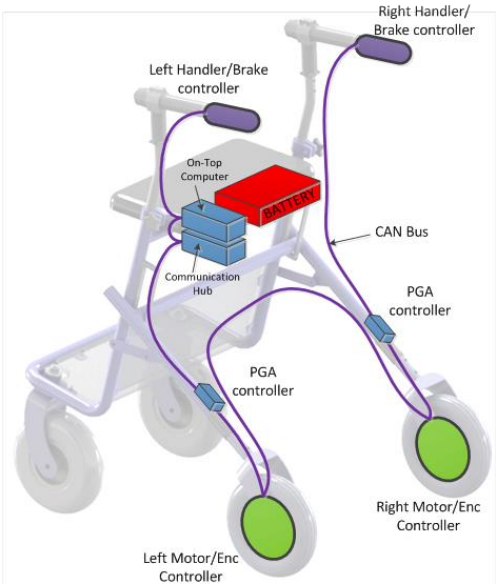


Ilustración 2 Esquema del iWalker

El iWalker incluye un sistema de vigilancia especial que permite detectar la posición del paciente en el hogar y en otros sitios conocidos, como los hospitales y centros de atención primaria. Otro de los sistemas que incluye es el tratado en este proyecto, la detección del movimiento de las piernas del usuario. Este sistema está enfocado sobre todo para que el personal médico especializado tenga un monitoreo de la forma de andar del paciente sin necesidad de estar presente.

No se encuentran proyectos parecidos que incluyan el sensor Kinect para detectar la movilidad de un usuario de un andador. Sin embargo se encuentran muchos proyectos diferentes que utilizan este sensor. El potencial y el relativo bajo coste de este dispositivo ha sido la causa que ha motivado a la gente a investigar y crear proyectos con él. Proyectos de tecnología punta en el sector de la automoción como el de *Steer-by-wire implementation using Kinect* [1]. En este trabajo se describe la implementación de la dirección por cable basándose en Kinect. La idea clave es reemplazar el volante y el sensor de ángulo por Kinect y utilizar su capacidad de reconocimiento de gestos para realizar acciones de dirección.

Otros trabajos tratan la teleoperación de un robot humanoide a través del reconocimiento de los movimientos humanos mediante el uso de un mecanismo de interfaz natural [2]. Proponiendo en un nivel satisfactorio, que el usuario en tiempo real controle un robot para imitar a una persona, para caminar, manipular el entorno con sus manos y llevar a cabo algunos comportamientos predeterminados. También hay proyectos que tratan de evaluar la carga de la columna vertebral humana, mediante el reconocimiento del esqueleto humano con el sensor de profundidad [3].

5.3 Alcance del proyecto

El alcance del proyecto se basa en la creación del software necesario para adquirir y tratar los datos necesarios realizando las funciones de los objetivos. Para ello se han tenido en cuenta varios enfoques, tanto a nivel hardware como de software; todo ello dependiendo de las herramientas disponibles para el desarrollo de software según la plataforma escogida. Desde un primer momento no se contempló la creación de controladores de dispositivos, simplemente utilizar las opciones viables de desarrollo que existen actualmente.

5.4 Actores implicados

Este proyecto desarrollado fue una propuesta del departamento de ESAII *(Enginyeria de Sistemes, Automàtica i Informàtica)* de la Universidad Politécnica de Cataluña.

Ilustración 3 Logo UPC

5.5 Metodología

A causa de que el proyecto del andador motorizado es un prototipo, la parte correspondiente a este proyecto también ha seguido los rasgos de un desarrollo de prototipos. Como las aplicaciones software requeridas no han sido de grandes magnitudes; sino simples aplicativos que reciben y tratan datos, no ha sido necesario un sistema de desarrollo más complejo; como podría ser el modelo clásico en cascada.

Las características útiles del sistema de desarrollo de prototipos para este proyecto son por ejemplo:

- Cuando los requerimientos son cambiantes.
- Si se quiere probar una arquitectura o tecnología.
- En el caso de que la aplicación no esté totalmente definida.
- En el caso de que se requiera rapidez en el desarrollo.

Todas estas características encajan con el proyecto en sí. Por el contrario, con esta metodología es difícil saber cuándo el producto será aceptable, ya que no se sabe de un principio cuantas iteraciones serán necesarias, y da una falsa ilusión sobre la velocidad de desarrollo.

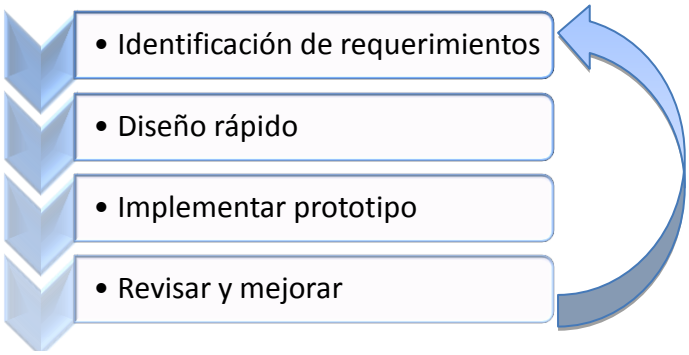


Ilustración 4 Esquema desarrollo de prototipos

6 Definición de alternativas

6.1 Plataformas Hardware

Gran parte de este proyecto ha sido encontrar una plataforma viable sobre la cual implementar la solución requerida. La idea inicial era encontrar una plataforma que se adaptase al andador y que funcionase con el sensor Kinect. La primera plataforma que se tuvo en cuenta fue la de usar la placa SBC (Single Board Computer) Raspberry Pi. Esta es una de placa de bajo coste desarrollada en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas. Debido a su pequeño tamaño bajo consumo y amplias características técnicas, fue la primera opción a tener en cuenta. Las características técnicas más relevantes son las de a continuación:

	Raspberry Pi modelo B
System-on-a-chip (SoC)	Broadcom BCM2835 (CPU + GPU. SDRAM chip a parte apilado encima.
CPU	700 MHz ARM11 ARM1176JZF-S core
GPU	Broadcom VideoCoreIV,OpenGL ES 2.0,OpenVG 1080p30 H.264 high-profile encode/decode
Memoria (SDRAM)iB	512 MiB
USB 2.0	2 (via USB hub)
Salida Video	Video compuesto RCA, HDMI
Salida Audio	TRS conector 3.5 mm jack, HDMI
Almacenamiento interno	SD / MMC / SDIO
Conexión de redes	10/100 Ethernet RJ45
Periféricos de bajo nivel	Pins Input/Output de propósito general(GPIO), Serial Peripheral Interface Bus (SPI), I²C, I²S[2], Universal asynchronous receiver/transmitter (UART)
Potencia nominal	700 mA, (3.5 W)
Fuente de alimentación	5 V (DC) via Micro USB type B o GPIO
Tamaño	85.0 x 56.0 mm

Tabla 1 Características Raspberry Pi

El hecho de que posee conexión UART facilita la integración con el robot ya que el protocolo de comunicación entre los diversos dispositivos es CAN. Con un módulo puente CAN/UART se puede conseguir la integración. El motivo por el cual es importante que la placa pueda conectarse al bus; es para poder tener constancia de lo que hacen el resto de sensores e interactuar si es necesario. Otro punto a favor para cumplir los requisitos de los objetivos, era

que mediante un dispositivo WiFi USB conectado a la placa se pudiese enviar los datos a la base de datos remota.

La segunda opción disponible era utilizar un ordenador portátil. Exactamente se disponía de un netbook; un netbook es una categoría de portátiles con características comunes: reducido tamaño, bajo peso, bajo coste, y con la particularidad de que no disponen de unidad óptica. El modelo concreto disponible, se trata de un Asus EeePC 901. Dentro del proyecto general, el de tener una flota de andadores robotizados, solo unos cuantos integrarían el sensor 3D. Debido a esto, el incluir un netbook en el andador sencillamente haría que fuese una unidad especial de la flota; unidad con funcionalidades especiales donde obviamente el aspecto variaría.

El pequeño tamaño del Asus EeePC 901, hace que sea viable la integración con el robot. Además ofrece potencial suficiente para el procesado de imágenes, condición indispensable para el proyecto. Obviamente esta plataforma cuenta con conexión a redes WiFi y conectores USB para poder conectar Kinect; así que cumple los requisitos esenciales para el proyecto. Algunas de las características de esta plataforma se ven en la siguiente tabla:

	Asus EeePC 901
CPU & Chipset	Intel Atom N270 1.6Ghz , Intel 945GC
GPU	Intel GMA 950
Memoria	1GB DDR2
Almacenamiento	32GB SSD
Pantalla	8.9” 16:9 1024x600 pixel
Batería	6 celdas Li-ion 7.6V 6.6Ah 50Wh
Peso	1.1 Kg.
Tamaño	42x225x174 mm (HxWxD)
Conexiones	3 USB 2.0, salida VGA, MMC/SD
Conexión a redes	Atheros AR8121/AR8113 PCI-E Ethernet Controller (10/100MBit), Ralink RT2860 (bgn), Broadcom Bluetooth

Tabla 2 Características Asus EeePC 901

Por ultimo cabe decir que se descartó la opción de usar un PC debido a las dificultades para integrarlo con el robot. Dificultades principalmente de espacio y consumo, principalmente porque para alimentar los componentes de un PC convencional se requiere más energía y una fuente de alimentación específica.

6.2 Sensores 3D

Desde el inicio del proyecto se ha tenido como primera opción el sensor 3D Microsoft Kinect. No obstante, a medida que avanzaron las pruebas sobre las diferentes plataformas comentadas anteriormente, surgió la necesidad de buscar otras alternativas. En esencia, los otros sensores considerados, ofrecen características similares a Kinect, aunque con algunas peculiaridades que mostraré más adelante. El sensor Kinect contiene un conjunto de elementos que funcionan concurrentemente; uno es la cámara VGA de video a color RGB, otro el proyector laser y sensor de profundidad, también incluye un motor y acelerómetro que permite modificar el enfoque, y una matriz de múltiples micrófonos.

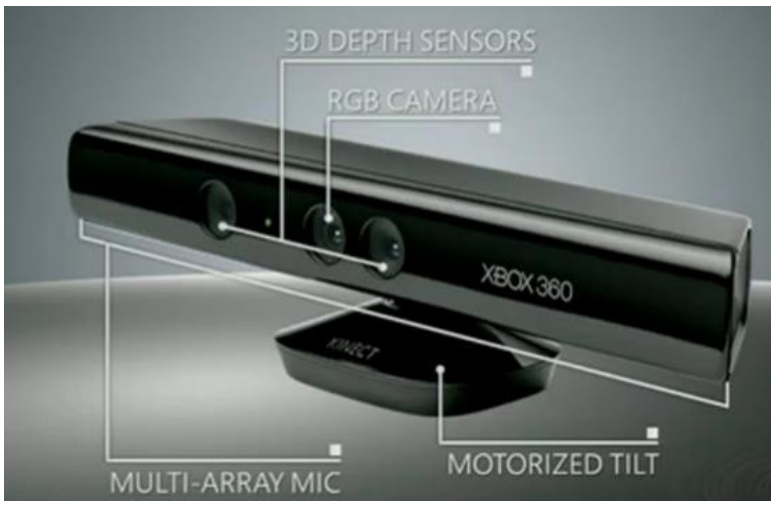


Ilustración 5 Kinect descripción general

La cámara detecta los componentes rojo, verde y azul. Ésta tiene una resolución de 640x480 pixeles. Para detectar la profundidad se proyecta patrones de puntos IR por la escena. Estos puntos son detectados por un sensor de imagen CMOS monocromo con un filtro de infrarrojos. El patrón cambia según los objetos que reflejan la luz; es decir, el conjunto de puntos cambia en tamaño y posición dependiendo de la distancia de los objetos con respecto de la fuente de emisión. Estas variaciones sirven para crear el mapa de distancias; todo ello a un framerate de 30 imágenes por segundo. El rango de operación de este sensor está a partir de 0.8 metros hasta los 4 metros.

Microsoft Kinect ha sido el sensor utilizado en el desarrollo de todo el proyecto, pero como he mencionado anteriormente se llegaron a considerar otros. Uno de ellos fue el sensor Asus Xtion Pro Live, que tiene un rango de operación similar a Kinect, de 0.8m a 3.5m. En cambio no cuenta con un motor para enfoque por lo que no consume tanta energía. Por ello puede alimentarse simplemente de la conexión USB, a diferencia de Kinect que necesita una fuente de alimentación extra.

Ilustración 6 Asus Xtion Live Pro

6.3 Plataformas Software

La primera plataforma hardware probada fue el binomio Kinect Raspberry Pi, por lo que el software se tenía que adaptar a éste. Raspberry Pi monta un procesador ARM, por lo que no puede usar un sistema operativo convencional para CPUs x86/64. Debido a esto, utiliza un sistema operativo adaptado para la placa; en concreto Linux Debian distribución Raspbian Wheezy. Las librerías más importantes que hay en la actualidad para Kinect son: OpenNI, OpenKinect, Kinect SDK de Windows. Las dos primeras ofrecen soporte para Linux Debian, por lo que la tercera se descartó para la placa Raspberry Pi.

La siguiente plataforma probada fue Kinect+netbook. Asus EeePC 901 viene con el sistema Operativo Windows XP preinstalado por defecto, a la vista está que las librerías de desarrollo a escoger debían de funcionar en este entorno. Una vez más OpenKinect y OpenNI cumplían el requisito, a diferencia de Kinect SDK de Windows que solamente funciona sobre Windows 7/8.

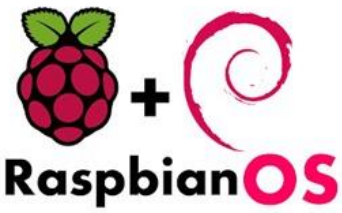


Ilustración 8 RaspbianOs logo

Ilustración 7 Windows XP logo

Ilustración 9 OpenKinect logo

Ilustración 10 OpenNI logo

En el siguiente árbol de decisiones se puede ver los posibles caminos y configuraciones seguidas. El sensor Asus Xtion se consideró aunque no se llegó a probar ya que no se dispuso de él durante el proyecto.

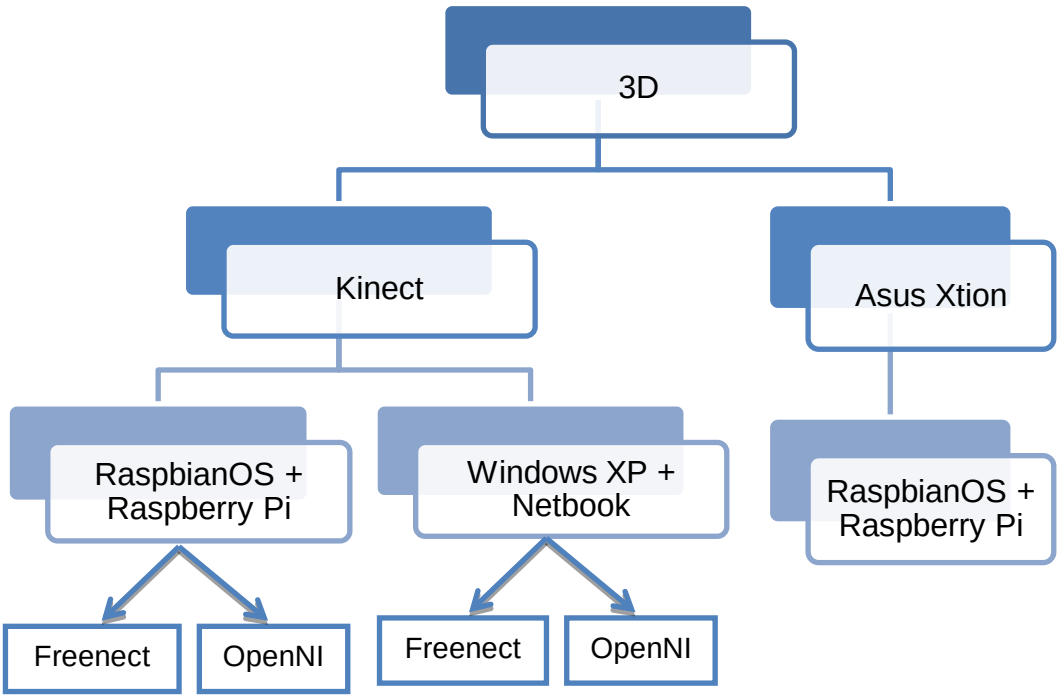


Ilustración 11 Árbol de decisiones

7 Búsqueda de la plataforma óptima

7.1 Kinect + Raspberry Pi

Ésta fue la combinación ideal desde el principio; para poderse llevar a cabo la conexión, se requirió que el sensor Kinect fuese conectado a un HUB USB con alimentación eléctrica externa. Esto se debe a que la propia Raspberry Pi no es capaz de proporcionar la alimentación requerida por el sensor. A nivel software, tal y como se ha dicho anteriormente hay dos alternativas:

7.1.1 Freenect en Raspbian y Ubuntu

Freenect es una librería perteneciente al proyecto de la comunidad OpenKinect. Se trata de una biblioteca de librerías de código abierto en las que se trabaja para poder utilizar el sensor Kinect sobre los sistemas operativos Windows, Linux y Mac. Esta desarrollada con el lenguaje de programación C, aunque al ser una cross platform es posible desarrollar aplicaciones en otros lenguajes como: Python, Java, C#, C++.

Para tomar contacto con su funcionamiento primero fue testeada en un PC en entorno Linux; exactamente con la distribución Ubuntu 12.04. Para ello se siguieron los pasos, proporcionados por la web de OpenKinect, para instalar los repositorios oficiales. Básicamente se consigue abriendo un terminal e introduciendo los siguientes comandos:

```
$ sudo apt-get install freenect
$ sudo modprobe -r gspca_kinect
$ sudo modprobe -r gspca_main
$ echo "blacklist gspca_kinect" |sudo tee -a
/etc/modprobe.d/blacklist.conf
$ sudo adduser $USER plugdev
```

Con estos comandos se consigue instalar la librería y los programas de demostración que se incluyen. De la siguiente forma se ejecuta uno de los programas de demostración que se incluyen:

```
$ sudo ./glview
```

Al ejecutar se obtiene una ventana mostrando la imagen que captura la cámara RGB de Kinect y la imagen previamente tratada del sensor de profundidad. La imagen se trata para destacar con diferentes colores y tonalidades las distancias relativas al sensor. Tal y como podemos ver en la siguiente captura de pantalla:

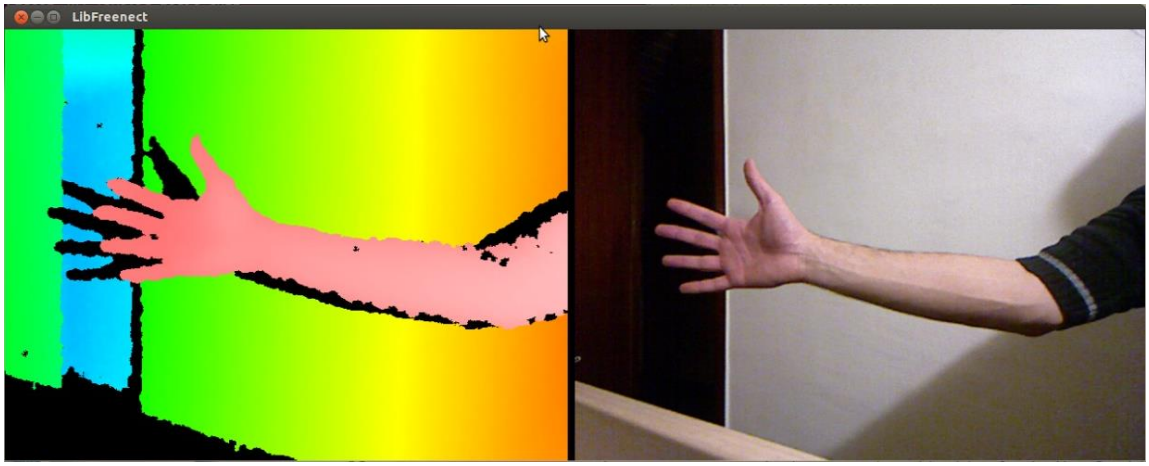


Ilustración 12 Captura de glview

Los mismos pasos se siguieron para la Raspberry Pi y se consiguió instalar el software sin mayor problema. Al intentar ejecutar el mismo programa de demostración (*glview)* apareció un problema. Debido a que dicha aplicación está programada para ejecutarse mediante la librería de reproducción grafica OpenGL, el programa terminaba con un error. El procesador gráfico de Raspberry Pi no soporta esta librería, sólo soporta ejecuciones para la librería OpenGL ES. El único programa de demostración de Freenect que acabó funcionando es *tiltdemo*; éste consiste en ejecutar movimientos con el motor de enfoque y cambiar el color y parpadeo del LED que incluye.

Para solucionar el problema, se decidió modificar el código fuente de *glview*, eliminando el apartado de representación gráfica; ya que lo que el proyecto requiere es la obtención de los datos capturados por el sensor de profundidad y no su representación visual. Con la instalación anteriormente descrita no se obtienen los códigos fuentes necesarios para desarrollar. Sencillamente se instala la librería y sus dependencias con los programas de ejemplo. Para obtener los códigos fuentes y el paquete de desarrollo hay que realizar una instalación manual; en la que hay que compilar los archivos fuentes. Siguiendo los pasos que se indican en la web de la librería [4], se consigue instalar el entorno de desarrollo fácilmente.

Una vez realizada la instalación se comprobó que el programa *glview* continuaba funcionando en PC, y lamentablemente no funcionando en Raspberry Pi; por lo que se prosiguió a modificar el código fuente. Dentro de la librería se encuentra un sub-paquete denominado Fakenect; que es un conjunto de aplicaciones para emular el funcionamiento de una Kinect sin necesidad de tenerla conectada. En ella se encuentra una aplicación interesante para el proyecto, la de record. Al ejecutarla se guarda en un directorio indicado el flujo de datos obtenidos por el sensor. El programa guarda la aceleración, la imagen RGB y la imagen de profundidad como archivos individuales nombrados según el tipo y tiempo de adquisición.

El programa *record.c* sobre PC funcionó, no en cambio para Raspberry Pi. Para descartar posibles problemas de ancho de banda del USB, se modificó el código fuente para solo adquirir los datos del sensor de profundidad, e incluso se redujo la cantidad de datos a adquirir. Esta modificación se compiló y se comprobó sobre PC y funcionó como se esperaba, sin embargo al realizar lo mismo sobre Raspberry Pi el resultado fue nulo. No se adquiría ningún dato; directamente las llamadas de la librería que sirven para adquirir los flujos de datos de la Kinect nunca se ejecutaban.

Investigando por la red no se encuentra ningún proyecto que envuelva a Kinect y Raspberry Pi que funcione. Varios mensajes en foros especializados en el tema, comentan que el problema de incompatibilidad entre estos dos dispositivos reside en los controladores USB requeridos por Freenect. La versión que necesita esta librería es libusb-1.0-0-dev; sin embargo, estos controladores muestran tener una incompatibilidad con los USB de Raspberry Pi que no les permite funcionar correctamente. Por causa de este problema, este camino de desarrollo del proyecto se descartó.

7.1.2 OpenNI en Raspbian

Esta librería ofrece características similares a la anterior con algunas diferencias, tales como que el apartado de representación gráfica se ejecuta mediante java. Instalando los paquetes necesarios de java se conseguiría visualizar los programas de demostración en Raspberry Pi. Otra diferencia es el lenguaje de programación usado en ella es C++ o C#. Otro punto a favor, es que parece ser el único software en el que el binomio Raspberry Pi y un sensor 3D parecen funcionar. Instalando la versión más reciente de esta librería para ARM (OpenNI2.2 Alpha), no se consiguió ningún resultado positivo. Los programas de ejemplo, al no detectar un sensor válido, no llegaban a ejecutarse.

La propia página web de OpenNI no muestra ninguna evidencia de que su software funcione para Kinect en arquitecturas ARM. Sin embargo antes de descartar esta vía completamente, se siguieron los pasos que aparecen en esta web [5].

Parece ser que el autor de estas instrucciones fue capaz de instalar OpenNI sobre Raspberry Pi y hacer funcionar el sensor Xtion Live Pro. Como este sensor comparte características semejantes con Kinect, se decidió seguir los pasos y comprobar si funcionaba. Algunos de los pasos más complejos son los relacionados con la compilación para el procesador ARM específico de Raspberry. Dentro del directorio de la librería OpenNI, una vez descomprimida hubo que modificar el archivo que se encuentra en:

ThirdParty/PSCommon/BuildSystem/Platform.Arm

CFLAGS += -march=armv7-a -mtune=cortex-a8 -mfpu=neon -mfloat-abi=softfp #-mcpu=cortex-a8

Por:

CFLAGS += -mtune=arm1176jzf-s -mfpu=vfp -mfloat-abi=hard

Después de unos minutos de compilación debería generarse el script instalador de la librería. En mi caso aparecieron varios errores de compilación referentes a la arquitectura ARM que no logré solventar. En ese caso encontré disponible para descargar una versión compilada en otra web [6]. Si no surgen errores y se genera correctamente, se siguen los pasos indicados en la página web.

Con la librería ya instalada, ya puede acceder a los programas de ejemplo y a sus códigos fuente. Al intentar ejecutar cualquiera de los programas de prueba el mismo error fue recurrente, ninguno detectaba el sensor Kinect.

Después de mucho investigar se concluyó que no es posible conectar Raspberry con Kinect, ya que no hay una forma clara y viable de conseguirlo. Teniendo en cuenta que estos dos dispositivos son utilizados por muchos desarrolladores impacta no encontrar en toda la comunidad de internet ni una sola evidencia de su funcionamiento entre sí.

7.2 Netbook + Kinect

Llegados a este punto, mi tutor me ofreció continuar por la vía del netbook + Kinect. En el departamento existía un netbook, exactamente el nombrado anteriormente EeePC 901; que cumplía con los requisitos. Todas las librerías a probar tenían que funcionar sobre el sistema operativo Microsoft Windows XP, el cual estaba instalado por defecto.

7.2.1 Freenect en Windows XP

Sobre Windows XP la primera prueba fue con Freenect. Obviamente los pasos para instalarla fueron diferentes y bastante más complicados que con Linux. Siguiendo los pasos que aparecen en la web oficial de OpenKinect y otros más detallados que encontré por la red, conseguí finalmente instalar todos los controladores necesarios y construir el entorno de desarrollo. Microsoft Visual Studio 2008/2010 fue necesario para comprobar que se podía programar y utilizar Freenect.

Con todo el entorno instalado, se prosiguió a ejecutar las aplicaciones de demostración como *glview*, mostrada anteriormente. La aplicación parecía detectar el sensor Kinect correctamente, aunque en las pantallas donde se supone que se muestra la imagen RGB y la de profundidad, no aparecía nada. Según la web de la librería podría ser problema de controladores de la tarjeta gráfica; se actualizaron los controladores, pero el problema persistió.

La funcionalidad final del proyecto no implicaba mostrar en tiempo real el flujo de datos que envía Kinect, así que el hecho de que no funcionase la visualización no era un gran problema. Sin embargo, había que comprobar que realmente Kinect estuviese enviando el flujo de datos. Para ello se intentó utilizar el subpaquete Fakenect para almacenar en imágenes los flujos obtenidos por Kinect. Al intentar compilar Fakenect en Visual Studio aparecieron errores de compatibilidad, debido a que es una librería escrita C para entornos Linux. Se intentó solucionar este contratiempo, pero como aún quedaba otra opción por comprobar, se decidió dejar este camino sin cerrar del todo.

7.2.2 OpenNI en Windows XP

Para instalar esta librería en Windows XP hay que seguir una serie de pasos diferentes a los más recientes. Esto es debido a que la librería actual está en su versión 2.2, la cual solo funciona sobre Microsoft Windows 7 o 8. Primeramente hay que descargar la versión OpenNi 1.5.4.0, la cual funciona con un controlador modificado para nuestro sensor; SensorKinect de Avin2. Acto seguido hay que instalar este controlador antes que la librería; el controlador se instala como cualquier otro en un entorno Windows, simplemente seguir el asistente de instalación.

Una vez realizado este paso al conectar la Kinect al puerto USB se nos reconocerá como NUI motor 1, nuestro propósito es conseguir el resto de dispositivos. Para ello, hay que ir al administrador de dispositivos dónde aparecerá reconocido el dispositivo NUI motor; el cual tiene otros dos dispositivos: la cámara y el audio. Haciendo clic con el botón derecho sobre ellos y clicando en actualizar controlador; aparece un menú dónde hay que introducir la ruta del controlador SensorKinect. Este paso es el mismo si se quiere activar el controlador de audio.

A continuación ya se puede instalar OpenNi 1.5.4.0 y comprobar que el programa de ejemplo SimpleViewer funciona. Para hacer funcionar el framework OpenNite, el cual incluye más opciones y ejemplos hay k instalar primero otro controlador que se encuentra en la carpeta de SensorKinect-unstable. Dentro del directorio /Bin de la carpeta descargada, hay que ejecutar el autoinstalable SensorKinect; la versión 32 bits.

Comprobé que los ejemplos de la plataforma funcionaban, seguidamente hubo que preparar el entorno de desarrollo. Para ello hay que descargarse la rama de OpenNi 1.5.23.4 de GitHub; la cual contiene los códigos fuentes y archivos necesarios para crear un proyecto con Microsoft Visual Studio 2008/2010. Instaladas las dependencias, se prosigue con los pasos requeridos para generar la “solución” que necesita Visual Studio; se siguieron los pasos ofrecidos en GitHub para llevarlo a cabo.

Siguiendo estos pasos, no se consiguió generar la solución en Windows XP. Se siguieron los mismos pasos en Windows 7 para descartar problemas con la versión, y efectivamente si funcionó. Se copió toda la carpeta con el ejecutable y la solución creada al equipo con Windows XP. Todo se instaló sin problemas, aunque la solución para Visual Studio no acabó de funcionar. La solución creó el proyecto con sus múltiples aplicaciones disponibles; pero a la hora de intentar generar un código ejecutable de alguna de ellas aparecía un error de depuración.

Debido a los varios problemas encontrados con las librerías de desarrollo para Windows XP, las opciones se reducían a dos. Una de ellas era usar Windows 7, donde la plataforma OpenNI2 y la SDK de Microsoft funcionan correctamente y están actualizadas; no como el caso de OpenNI 1.5 que se encuentra fuera de posibles mejoras. O la otra opción era utilizar Linux con alguna distribución Debian, como por ejemplo Ubuntu 12.04.

7.3 Elección final

Finalmente se optó por la combinación Ubuntu 12.04 y la librería Freenect de OpenKinect. Se consideró viable al no depender de licencias por ser software libre y porque ya se había comprobado Freenect en un PC de sobremesa obteniendo buenos resultados. Se prosiguió a preparar el entorno instalando lo necesario, tal y como se explicó anteriormente, y comenzó una nueva etapa en el proyecto.

Ilustración 14 OpenKinect logo

Ilustración 13 Ubuntu logo

8 Implementando la solución

Una vez concretada la plataforma de desarrollo simplemente quedaba empezar la etapa en sí. Esta etapa de desarrollo de la implementación la separo en las distintas partes más significativas que se han llevado a cabo por orden cronológico. A continuación se muestra el diagrama de flujo del software implementado.

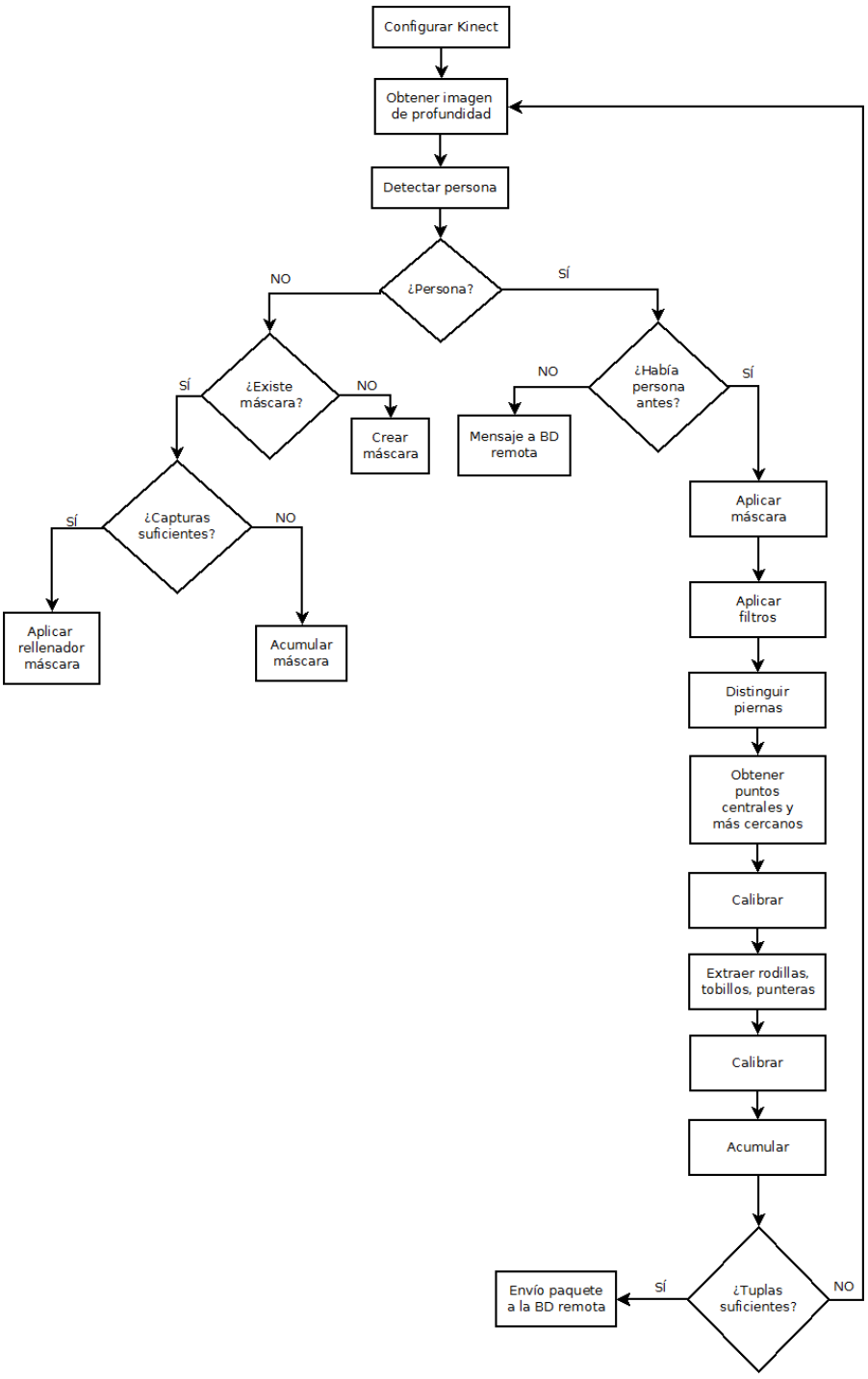


Ilustración 15 Diagrama de flujo del software

8.1 Capturas de profundidad

Gracias a los códigos fuente de la librería Fakenect, puede entender rápidamente como funcionaba el proceso de obtención del flujo de imágenes. Basándome en la aplicación de demostración record.c, puede adaptar el programa glview.c; para que a la vez que mostraba las imágenes de las dos cámaras, fuese guardando en el disco duro cada una de estas capturas. Una vez conseguí las capturas puede empezar a ver qué tipo de información aportaba Kinect. Es decir, entender esas imágenes, en escala de grises, como un mapa de relieve de donde había que extraer las piernas del usuario.

Con la ayuda del programa ImageJ, se puede conseguir plot profiles de las zonas deseadas de la imagen. Esta herramienta genera un gráfico con los valores que se encuentran en el rectángulo de selección; lo cual es útil para entrever las diferentes pates de la imagen en profundidad, tal y como se muestra en la siguiente imagen.

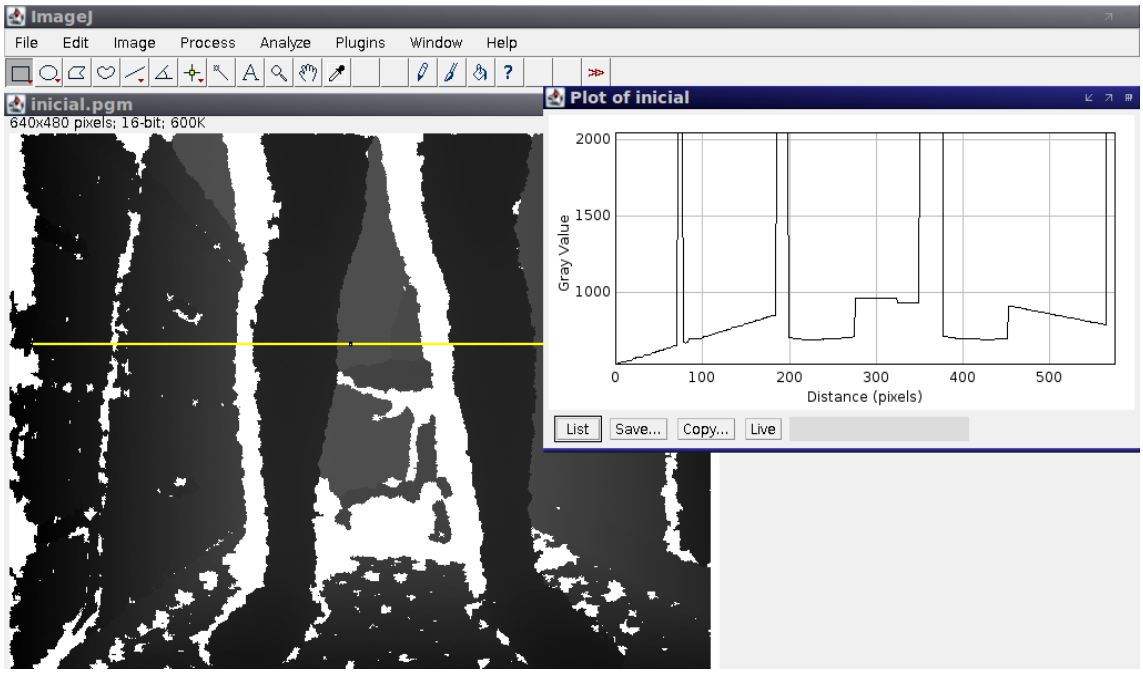


Ilustración 16 Captura ImageJ mostrando plot profile

Freenect obtiene del sensor 11bits de resolución por pixel, este valor discreto de un pixel es traducible a una distancia. En la red hay numerosos estudios sobre este aspecto, así que no me costó mucho encontrar una gráfica que tradujese esos valores discretos a distancias reales. Pese a que la resolución es de 11bits, con OpenKinect no se llega a utilizar todo el rango de valores; tal y como podemos ver en la siguiente gráfica.

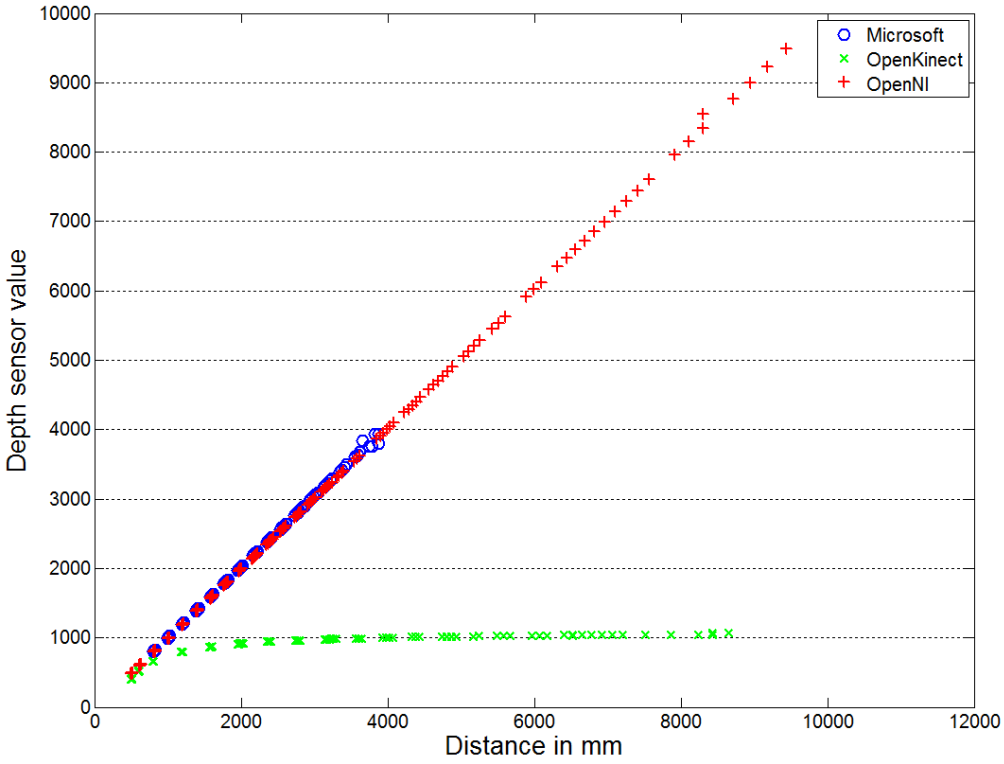


Ilustración 17 Relación distancia/valores según plataforma

8.2 Detectar usuario

Una vez comprendido los valores proporcionados por el sensor, ya se pudo comenzar a tratarlos. La primera tarea era simple, detectar si un usuario se encuentra enfrente del sensor suficientemente cerca. Básicamente lo que esto significa, es que tiene cogido el andador con las manos y puede ser que se disponga a caminar con él. Freenect utiliza una llamada callback, que activa un procedimiento el cual es ejecutado cada vez que Kinect le envía el flujo de datos. Este procedimiento recibe como parámetro, entre otras cosas un puntero a void a la matriz de profundidad. Para discernir si una persona y no el suelo, se encuentra enfrente del sensor a una distancia cercana, delimité el rango de búsqueda tal y como se muestra en la siguiente imagen.

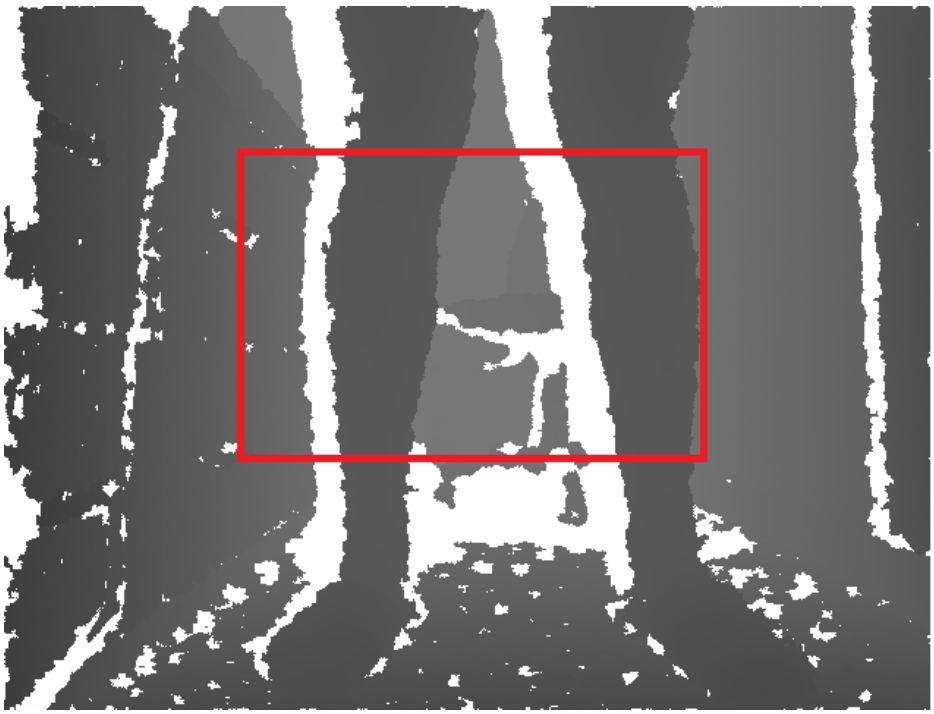


Ilustración 18 Captura mostrando la limitación en la búsqueda de usuario

Lo que el siguiente código realiza es consultar cada pixel, y si éste se encuentra dentro del rango establecido, se tiene en cuenta para el cálculo de la media. Si ningún punto se encuentra dentro del rango, la media resultante será 0 por lo tanto no se considera que haya un usuario. De lo contrario, si la media es superior a un valor de 400, comprobado empíricamente que es representativo para distancias cercanas, se considera que hay un usuario.

```
for (i = 100; i<300; ++i) {
    for (j = 160; j<480; ++j) {
        if (buffer[640*i + j] > 10 && buffer[640*i + j] < 720) {
            mean += buffer[640*i + j];
        }
    }
}
return mean /= (200*320);
```

8.3 Separar piernas de la escena

El siguiente paso era distinguir las piernas sobre el resto de la escena. El principal problema aparece con los pies y el suelo, ya que la base de los pies se encuentra a la misma distancia que el suelo y es necesario poder distinguirlos. Para solventar esto, el tutor me propuso crear una máscara donde no apareciesen las piernas del usuario, pero si el suelo con el resto de la escena. Una vez obtenida la máscara, solamente hubo que comparar pixel a pixel cuáles de ellos eran semejantes, y por tanto descartarlos. Los pixeles diferentes a la máscara son considerados apariciones nuevas en la escena; en este caso las piernas del usuario.

Este procedimiento funcionaria bien teóricamente, el problema está en que entre una captura del sensor y la siguiente, hay mucha diferencia debido al ruido del sensor. Pude identificar dos tipos de ruido; uno de ellos es el que aparece en algunos bordes de los objetos y otro que aparece de forma aleatoria por ciertas zonas de la captura. El primero es fácilmente entendible ya que es consecuencia de la propia tecnología usada en el sensor, tal y como se muestra en la primera imagen. Las otras dos imágenes muestran la variación de una captura a otra en un lapso de tiempo inferior a un segundo.

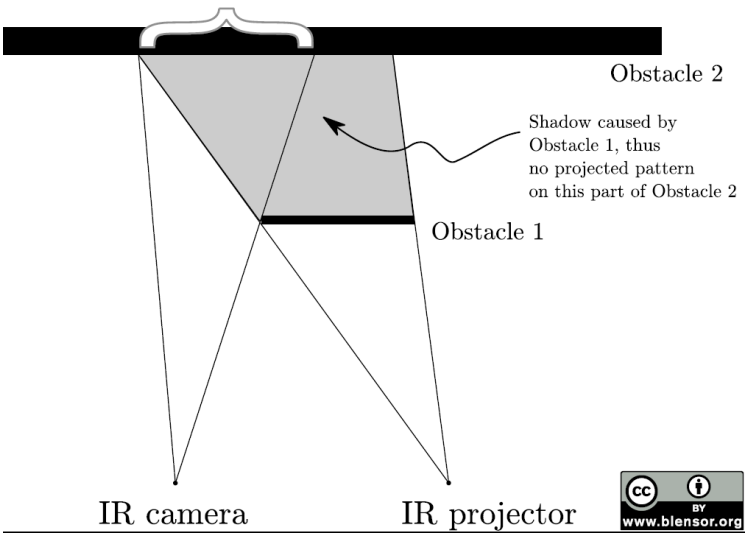


Ilustración 19 Motivo de ruido del sensor

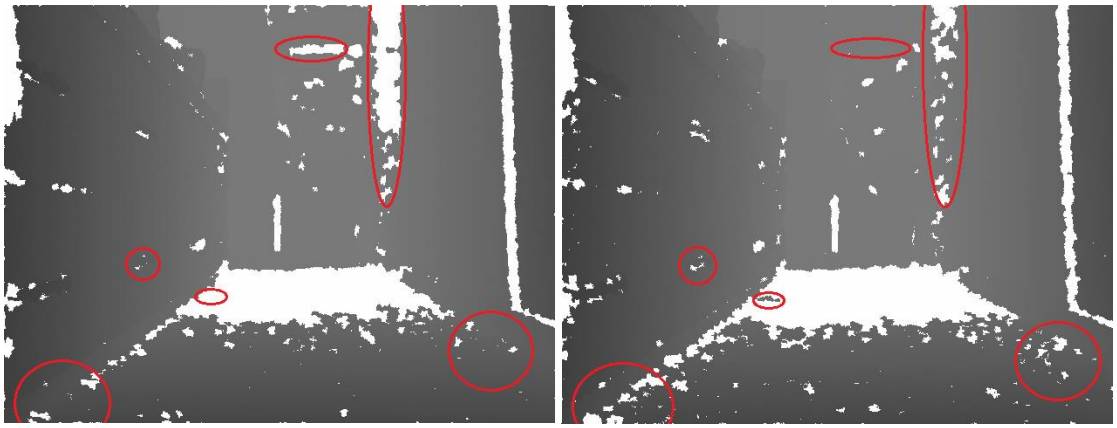


Ilustración 20 Diferencias de ruido en un intervalo inferior a 1seg

Para solucionar el segundo caso de ruido se optó por crear una máscara acumulativamente. Como el determinado ruido que aparecía en ciertas zonas de la captura variaba con las respectivas capturas, se dedujo que con varias muestras se podría llegar a completar la máscara. Para ello se parte de la primera máscara capturada, se buscan todos los pixeles que sean ruido y se almacena su posición. Buscar el ruido no es muy complicado, por deducción de la anterior gráfica, basta con recorrer la captura buscando los valores mayores a 1024.

Cuando ya se tiene la lista de posiciones con ruido, simplemente ésta se recorre cada vez que viene una nueva muestra. Si la posición que indica la lista de pixeles con ruido resulta no ser ruido en la nueva muestra, se actualiza la máscara original con el valor del pixel en concreto. Los experimentos realizados mostraron que con un número de muestras entre 100 y 200 ya era suficiente para crear una máscara prácticamente completa; más allá de estos valores no se obtenía mejora sustancial. En las siguientes imágenes se muestra la diferencia entre la captura inicial y la máscara reconstruida por acumulación.

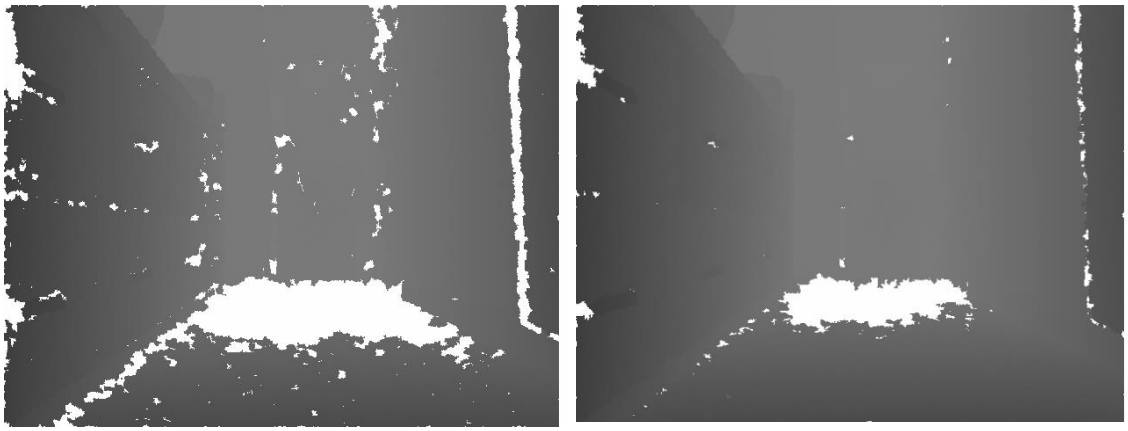


Ilustración 21 Comparación entre máscara inicial y final

Para acabar de mejorar la máscara había que solucionar el primer tipo de ruido comentado. Lo que encontré investigando sobre el tema, fueron varias formas de rellenar los espacios vacíos de información generados por el ruido. De los diferentes métodos existentes, escogí el de rellenar el pixel basándose en el valor de los vecinos más cercanos que no fuesen ruido. Por cada pixel de la matriz consultaba la submatriz de tamaño 15x15 de los pixeles de alrededor tal y como se muestra en la imagen. Se decidió este tamaño de submatriz ya que se comprobó que funcionaba y porque la mayoría de artículos relacionados con el tema así lo recomendaban.

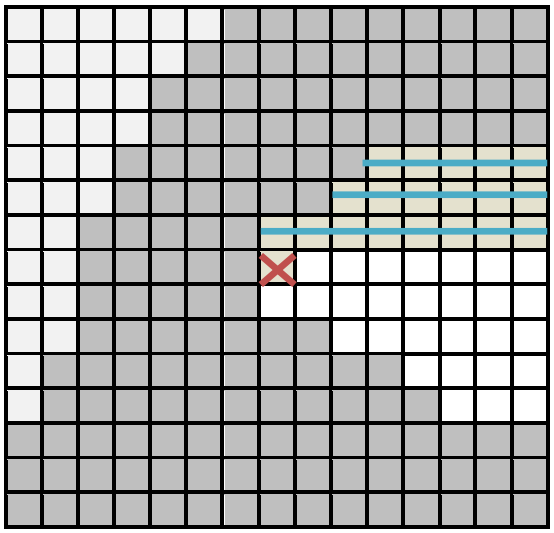


Ilustración 22 Submatriz denotando(azul) pixeles ya tratados, en rojo el pixel actual

Aplicando el algoritmo de relleno repetidas veces se consigue una máscara reconstruida bastante fiel a lo que debería ser sin ruido. El proceso de transformación es el siguiente:

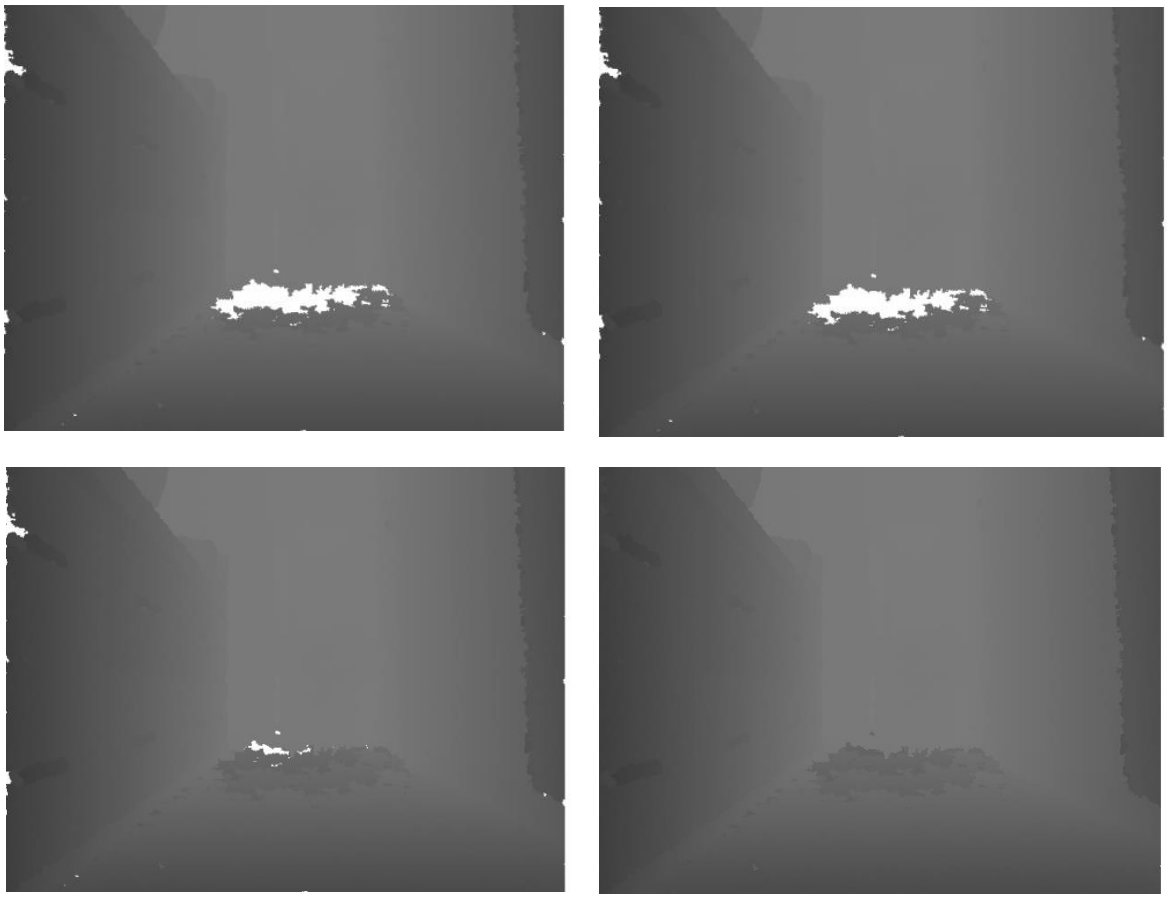


Ilustración 23 Evolución del algoritmo de relleno aplicado cuatro veces

Una vez creada la máscara, comienza el proceso de aislamiento de las piernas de la escena. La primera fase consiste en aplicar la máscara a todos los frames siguientes mientras exista un usuario en el andador. Al mismo tiempo que se aplica la máscara al buffer con la captura nueva, se aplica un filtro para delimitar la profundidad. Es decir, los pixeles cuyo valor sea mayor a cierta distancia son descartados. Concretamente los valores descartados tanto con el filtro y la máscara son tratados como valor 0. A causa de esto en la siguiente imagen donde se aplica lo expuesto, la mayor parte de la imagen que no son piernas es de color negro.

Ilustración 24 Residuos de ruido

A partir de este punto todos los filtros y procedimientos se efectúan a la matriz buffer. Tal y como se observa en la figura anterior, aún existen pequeñas motas con valores válidos pero irrelevantes para la tarea. Interesa quedarse simplemente con las piernas, para ello se vuelve a aplicar otro filtro. Esta vez con una submatriz de 9x9 pixeles se consultan todos los vecinos del pixel en cuestión. Se acumulan los valores de los pixeles de todos los vecinos y después se divide entre el número de pixeles consultados. Lo que se quiere conseguir con ello es ver si un pixel que no es “nulo”, es decir que su valor no es 0, forma parte de la pierna o simplemente es un residuo aislado de ella. Las imágenes siguientes lo ejemplifican debidamente.

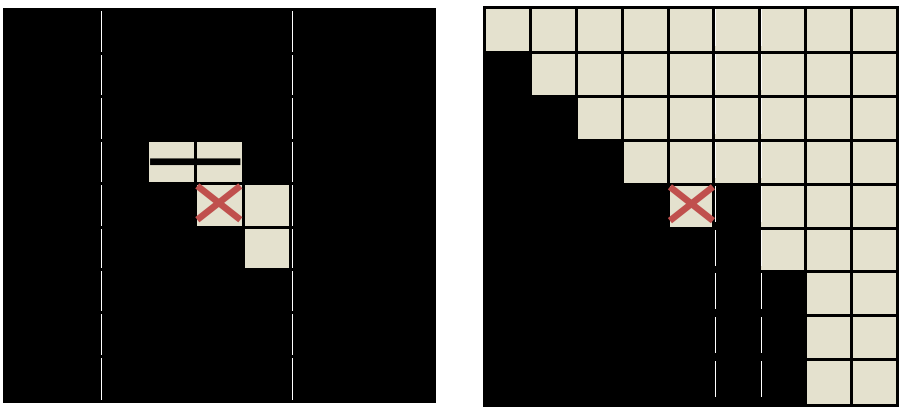


Ilustración 25 Caso izq. Mostrando un pixel considerado ruido, caso der. lo opuesto

El tamaño de la submatriz se fijó a partir de investigaciones y comprobaciones propias. Una vez pasado este filtro el resultado es el siguiente, en el buffer ya solo quedan los valores en profundidad de lo que resultan ser las piernas del usuario.

Ilustración 26 Captura con todos los filtros aplicados

8.4 Obtención de los datos esenciales

Una vez aisladas las piernas del resto, me dispuse a extraer la información realmente importante de las piernas. Para entender qué datos tenía hasta el momento y como podía usarlos para afrontar el problema, utilicé la herramienta ImajeJ. Retocando debidamente alguna de las capturas con los filtros aplicados y utilizando le herramienta de surface plot de ImageJ obtuve lo siguiente.

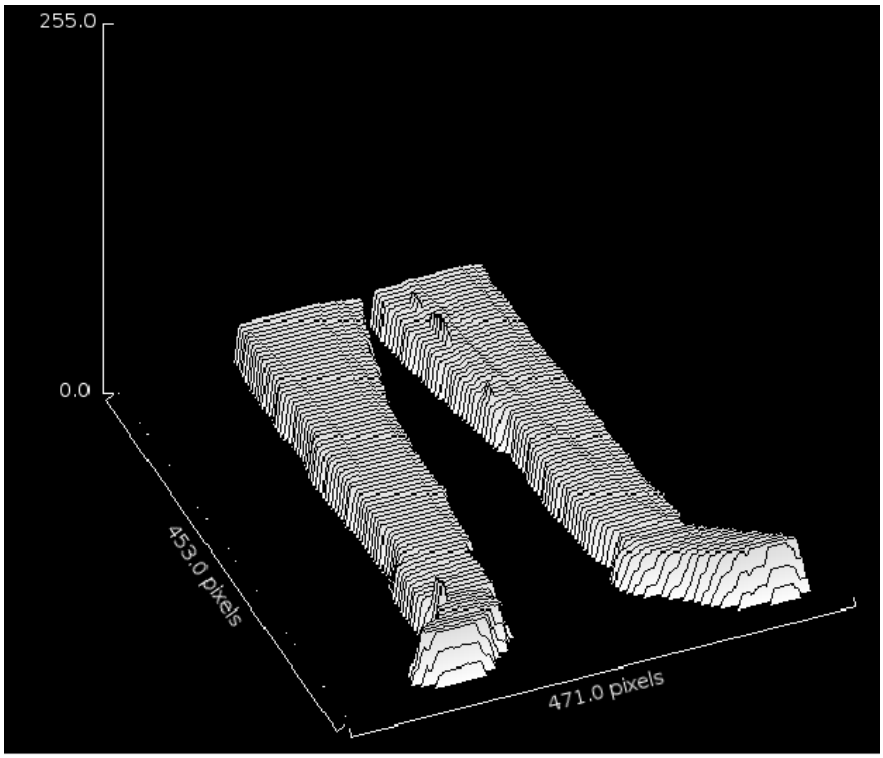


Ilustración 27 Surface plot

Una vez interpretados los datos que se obtenían una vez aplicado los filtros, había que comenzar a extraer la información relevante. Uno de los datos a extraer era la anchura de cada pierna. Para ello se implementó un procedimiento que hacía un barrido horizontal para saber en qué pixel comenzaba y en que pixel acababa ese fragmento de la pierna. Una vez obtenido el inicio y el fin de cada pierna obtener el punto medio entre estos fue tarea trivial. Sin embargo, esta solución para obtener los puntos medios no es fiable completamente en el caso de que las piernas estén juntas. Por causa de esto, se decidió enfocar la obtención de este dato de otra forma.

Distinguir si dos piernas están totalmente juntas se puede conseguir mediante la asunción de que la anchura obtenida con el barrido es mucho mayor. Siguiendo este razonamiento se acoto un tamaño máximo empírico para el ancho de una pierna. Exceder ese ancho implica que las piernas están juntas o bien que la pierna en si es muy ancha. En el caso de que la pierna sea muy ancha el modelo seguido no funcionaria, ya no solo por el algoritmo seguido sino por la forma física del andador. El enfoque esperado del sensor ya viene delimitado por la geometría del andador por lo que en este caso unas piernas muy anchas no entrarían en el área de enfoque.

Puesto que la imagen obtenida no deja de ser una representación 3D, me dispuse a utilizar la información de la profundidad para obtener más datos relevantes. Partiendo de la base de que cuando dos piernas se juntan se forma una concavidad entre ellas, me dispuse a implementar un algoritmo para obtener este punto medio. Aproveché la información obtenida del anterior barrido horizontal, para volver a hacer una búsqueda horizontal entre los pixeles iniciales y finales, con el propósito de encontrar la concavidad.

Una vez encontrada la concavidad, ya se consiguió distinguir las piernas. Con los valores de pixel inicio y pixel fin actualizados, se pudo obtener los puntos medios de cada pierna. En la siguiente ilustración se puede apreciar cómo se trazan las líneas de puntos medios, inclusive si las piernas están juntas.

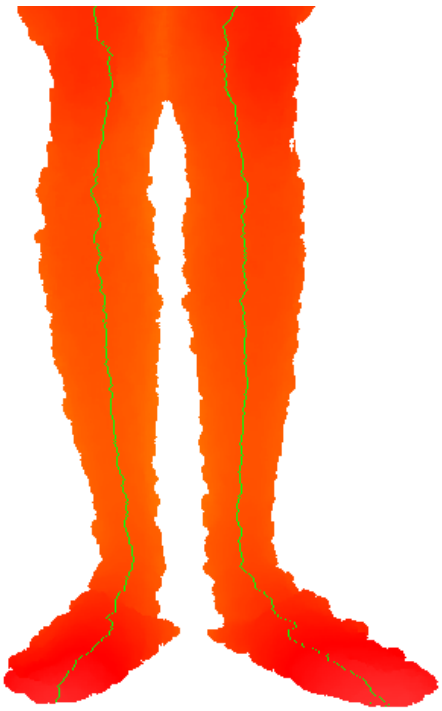


Ilustración 28 Mostrando puntos centrales

Conseguido lo anterior, otro dato requerido para ser extraído era el de los puntos más cercanos al sensor. Con los obtenidos ejes centrales de las piernas y sus puntos más cercanos, ya se podía reconstruir los movimientos. Para obtener estos puntos se siguió la misma mecánica. Mediante barridos horizontales se buscó el punto más cercano al sensor entre los pixeles de inicio y fin de cada pierna. Seguidamente se pueden observar los puntos más cercanos junto con los ejes centrales.

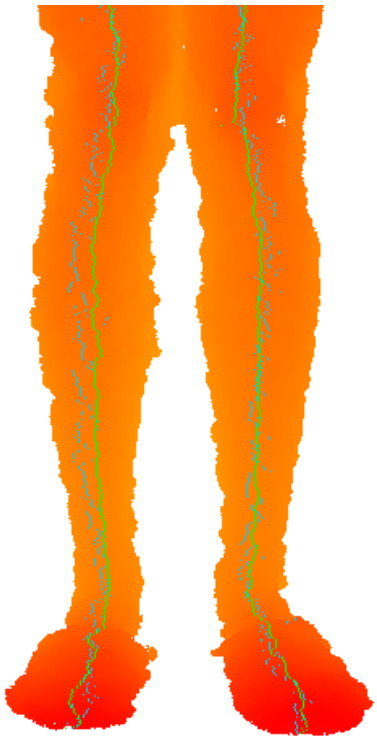


Ilustración 29 Puntos centrales y más cercanos al sensor

8.5 Calibración del sensor

Una vez obtenidos los datos relevantes, había que ajustarlos a valores en distancias con respecto del sensor. Para ello había que seguir los métodos de calibración que existen para este sensor. Esto es un tema bien investigado, por lo que no me costó mucho encontrar el modo de llevarlo a cabo. Entendiendo como el sensor representa los datos; usando algunas constantes referentes a Kinect que ofrece la librería, y siguiendo las siguientes formulas, se puede obtener las coordenadas en el espacio relativas al sensor. Las distancias finalmente obtenidas no resultaron tener una precisión elevada, pero sí suficiente para la tarea requerida.

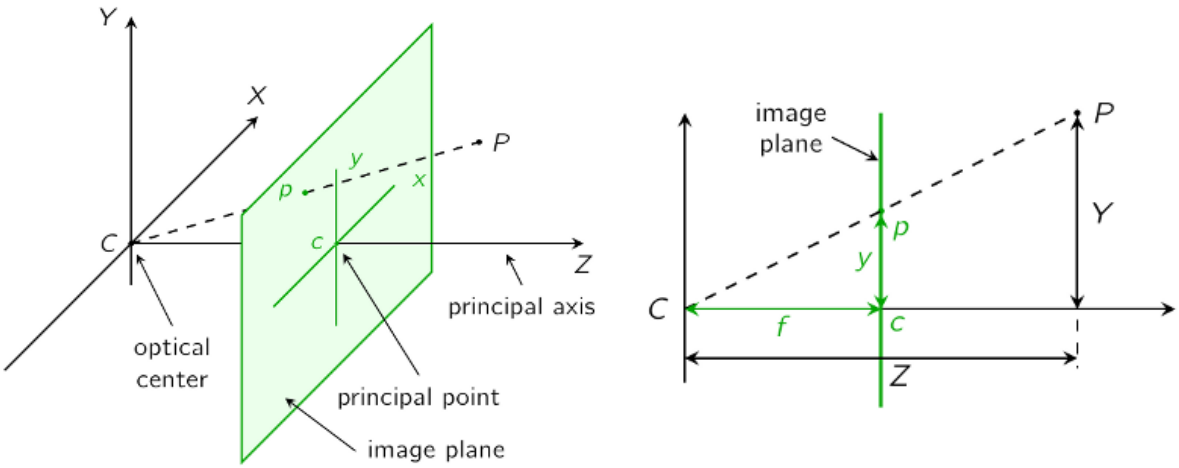


Ilustración 31 Esquema de calibrado

$$Z_{\text{meter}} = \frac{1}{Z_{\text{raw}} \cdot (-0.0030711016) + 3.3309495161}$$

$$\begin{pmatrix} x_d \\ y_d \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X_d \\ Y_d \\ Z_d \\ 1 \end{pmatrix}$$

$$\Leftrightarrow \begin{pmatrix} x_d \\ y_d \\ 1 \end{pmatrix} = \begin{pmatrix} f_x X_d + c_x Z_d \\ f_y Y_d + c_y Z_d \\ Z_d \end{pmatrix} = \begin{pmatrix} f_x X_d / Z_d + c_x \\ f_y Y_d / Z_d + c_y \\ 1 \end{pmatrix}$$

$$X_d = \frac{(x_d - c_x) \cdot Z_d}{f_x}, \quad Y_d = \frac{(y_d - c_y) \cdot Z_d}{f_y}$$

Ilustración 30 Ecuaciones de calibrado

8.6 Segmentación de las piernas

Con la extracción de datos relevantes anterior, ya se reduce considerablemente la información. Hemos pasado de tener un conjunto de imágenes por segundo, que obviamente aportan información de cómo camina el usuario, a un conjunto de datos mucho menor que pueden reconstruir perfectamente el movimiento de piernas de un usuario. La mejora fue evidente aunque aún podía ser mayor; para ello se enfocó el problema para reducir el número de puntos relevantes. Con el sensor colocado en el caminador lo que capta viene a ser la parte inferior de la pierna, es decir de rodilla hasta el pie. Partiendo de este punto, solamente son necesarios tres puntos para reconstruir el movimiento de una pierna; un punto seria la rodilla, otro el tobillo y por último la punta del pie.

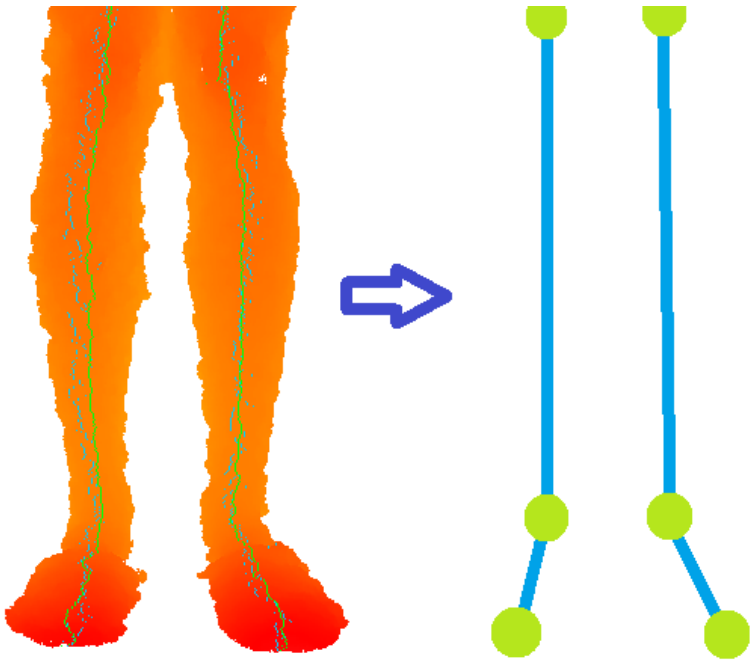


Ilustración 32 Representación de la extracción del modelo deseado

Obtener los puntos de la punta del pie fue tarea sencilla. Aproveché las estructuras de datos anteriores, y los obtuve haciendo una búsqueda de los puntos más cercanos al sensor de la parte correspondiente al pie. Este algoritmo es viable, porque se parte de la base de que el pie no va a estar en posiciones extrañas. En resumen que la puntera del pie siempre será el punto más cercano al sensor.

Obtener la rodilla y el tobillo fue más complejo; ambos puntos comparten la característica de que se encuentran en el mismo eje, por lo que hubo que encontrar tales ejes. Primeramente se abordó la obtención del tobillo, para ello hubo que tener en cuenta como mínimo dos tipos de posiciones de los pies. Tal y como muestra la imagen, una de ellas es cuando la puntera enfoca hacia el sensor y la otra es cuando la parte interior del pie se enfoca al sensor.

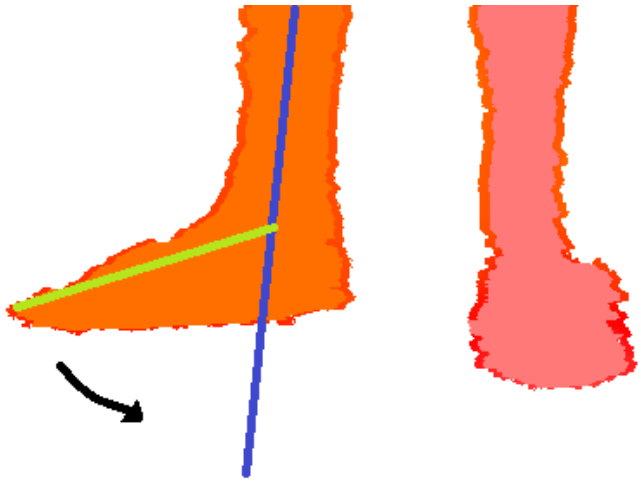


Ilustración 33 Posibles posiciones de los pies

El algoritmo obtenido también funciona para los casos intermedios; es decir, cuando la parte interior del pie no está totalmente perpendicular al eje de visión del sensor. Sin embargo, casos tales como en el que la puntera esta elevada con respecto el talón y la parte interna del pie está enfocada al sensor, no se tuvieron en cuenta. Esa y otras posibles posturas son poco realistas para el caso del andador, por lo que no hizo falta tratarlas.

El algoritmo en cuestión, por una parte realiza una búsqueda del punto más alejado del sensor. Se comienza desde el punto más próximo al suelo hasta una distancia limitada de la pierna; entre ese intervalo se considera que debe de existir un tobillo. Al mismo tiempo, comenzando desde la parte inferior también; se busca la anchura máxima y si se encuentra una variación considerable con el punto máximo encontrado, se deduce que el pie estaba con el interior enfocando al sensor. Por lo tanto, en esa altura es posible que se encuentre el tobillo.

Discernir si el pie estaba en una posición u otra se comprueba mirando la media de anchuras. Si esta es inferior a 20cm se considera que la punta miraba hacia el sensor. El fragmento de código siguiente muestra cómo se realiza la búsqueda del tobillo en una pierna.

```
for (i = leg1[2].y; i > (leg1[2].y-80); i--) {
    if (nearpts[i].z > far1.z) {
        far1.x = centerpts[i].x;
        far1.y = i;
        far1.z = nearpts[i].z;
    }
    meanw1 += widths[i];
    if (widths[i] > maxw1) {
        maxw1 = widths[i];
    }
    else if ((widths[i]+0.05) < maxw1 && !ank1) {//margen de 5cms
        ank1 = 1;
        ankle1.x = centerpts[i].x;
        ankle1.y = i;
        ankle1.z = nearpts[i].z;
    }
}
if (meanw1/80 < 0.2) { //el interior no mira hacia el sensor
    ankle1.x = far1.x;
    ankle1.y = far1.y;
    ankle1.z = far1.z;
}
```

Con los tobillos obtenidos, aún faltaban las rodillas y los ejes dónde ajustar estos cuatro puntos. Obtener la rodilla fue tarea trivial puesto que partimos de la base de que tal y como está situado el sensor, la parte superior delas piernas ya coincide con las rodillas. A causa de esto solo quedaba obtener los ejes de las piernas. Estos ejes no son más que las rectas que mejor se ajustan a la nube de puntos centrales de cada pierna; en otras palabras, realizar la regresión lineal simple del conjunto de puntos centrales entre la rodilla y el tobillo y proyectar la recta de regresión. Utilizando las denominadas ecuaciones normales se modela la relación entre la variable dependiente ***y***, la variable independiente ***x*** y un término aleatorio ***ε*** .

$$\beta = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - (\sum x)^2}$$

$$\varepsilon = \frac{\sum y - \beta(\sum x)}{n}$$

$$y = \beta x + \varepsilon$$

Realizar la regresión lineal con todos los puntos centrales de cada pierna podía ser muy costoso; por lo que se decidió buscar otro enfoque que diese resultado y fuese suficientemente robusto. El enfoque tomado fue el de segmentar la nube de puntos centrales en pequeños fragmentos. De cada fragmento, se obtiene la media en el eje de las abscisas de los puntos centrales. Con cuatro fragmentos, de un pequeño conjunto de puntos, fue suficiente para obtener cuatro puntos con los que obtener, mediante la regresión lineal, un eje fiable.

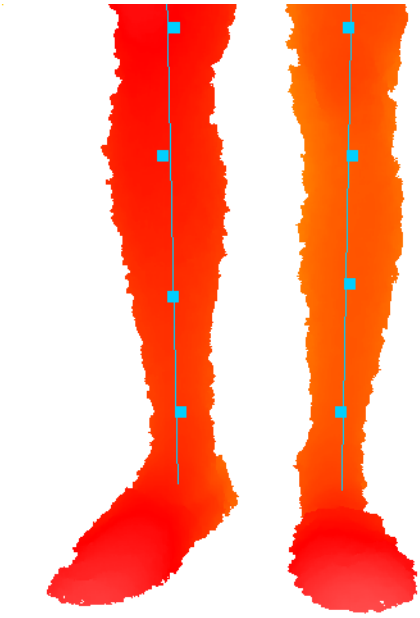


Ilustración 34 Conjunto de puntos y respectiva recta de regresión

En la anterior imagen se observan los puntos obtenidos de cada fragmento, y la recta de regresión lineal resultante. Una vez obtenidos los ejes mediante las rectas de regresión, únicamente quedaba ajustar los puntos de las rodillas y tobillos sobre tales ejes; esta tarea no implicó ninguna dificultad y en la siguiente imagen se pueden ver todos los puntos requeridos. Rodillas y tobillos aparecen sobre sus respectivos ejes y los puntos correspondientes a las punteras de los pies denotan el punto del pie más cercano al sensor.

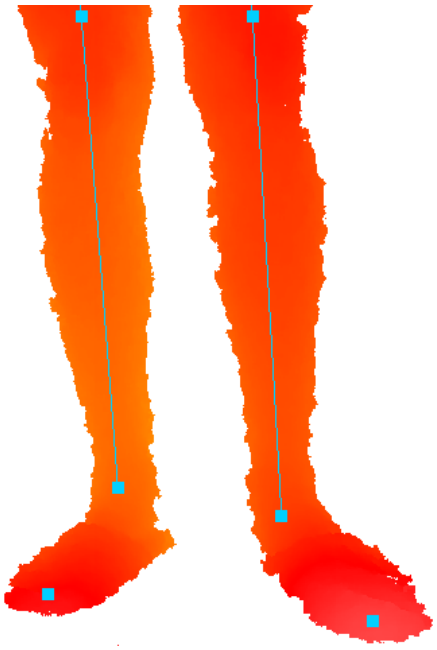


Ilustración 35 Puntos requeridos y ejes

8.7 Conexión a la base de datos remota

Para cumplir con uno de los objetivos del proyecto, había que enviar la información obtenida por WiFi. Para llevarlo a cabo, se utilizó una conexión remota a una base de datos utilizando el sistema de gestión de datos MySQL. Puesto que este gestor acepta varias interfaces de programación, integrarlo en el código escrito en C no resultó ningún problema. Los datos a enviar son los seis puntos convenientemente calibrados y cada vez que aparece o desaparece un usuario; todo ello indicando el Unix timestamp, para que después la información se pueda sincronizar con el resto de información que aporta el robot.

En el caso de indicar si aparece un usuario o desaparece, la inserción en la base de datos consta del Unix timestamp y un mensaje de persona o no persona. Sin embargo, para enviar los puntos se decidió no hacer una inserción por cada tupla de seis puntos. En vez de eso, el programa acumula en un buffer los puntos de veinte capturas; a partir de ese momento si se puede realizar la conexión con la base de datos y enviar el conjunto de puntos.

Realizando esto se reducen el nivel de conexiones remotas; teniendo en cuenta que los puntos de veinte capturas solo tienen un tamaño de 1440 Bytes, es más conveniente realizar tal envío en vez de abrir y cerrar una conexión cada vez para enviar 72 Bytes. Otra opción sería dejar la conexión abierta desde el primer momento; esto supondría un problema al tener varios robots funcionando e insertando datos sobre el mismo servidor remoto.

9 Descripción de la evaluación de los resultados

La evaluación del resultado final será una valoración cuantitativa. Esto se debe a que el proyecto trata sobre que un hardware productor de datos, los envía a la plataforma escogida, y ésta los trata y envía. Por ello para comprobar que el funcionamiento es el correcto, se tendrá que realizar una serie de pruebas dónde se demuestre que la información generada se corresponda a la realidad.

Se trata de extraer un conjunto de puntos de las piernas del usuario. Estos puntos se encuentran a cierta distancia del sensor, así que para comprobar si son válidos simplemente hay que generar una serie de modelos donde las piernas aparezcan en diferentes posturas. Midiendo las distancias de los puntos requeridos del modelo físico con el sensor ya se obtiene los patrones de distancias reales. A continuación se repiten las mismas posturas con el algoritmo funcionando y se comparan los datos obtenidos con los valores fijados por el patrón.

10 Planificación

10.1 Consideraciones

El proyecto empezó el 28 de Enero de 2013 y acaba el 13 de Junio del mismo año. Como se ha comentado anteriormente el desarrollo del proyecto ha seguido el modelo de prototipo por ello en la planificación siguiente aparece que muchas subtareas se solapan. Esto es debido a que el sistema prueba y error hace que la tarea de testeo se haga en paralelo con la de implementación, a su vez al mismo momento que se inicia otra etapa de implementación se puede requerir volver a comprobar tareas supuestamente cerradas. Debido a este planteamiento solamente he contabilizado como etapa de verificación aquella que viene después de la implementación finalizada.

10.2 Tareas y subtareas

Tarea	Subtarea	Tiempo(h)
Planificación y viabilidad		
	Alcance del proyecto	14
	Planificación temporal	14
	Presupuesto	10
	Video de presentación preliminar	10
	Contexto y bibliografía	14
	Anteproyecto Ing. Computadores	10
	Presentación	10
	Corrección documento para TFG	16
		100
Investigación de las alternativas		
	Investigación sobre posibles controladores para Kinect	20
	Estudio sobre los diferentes sistemas operativos a usar según la plataforma	20
	Investigar las diferentes librerías de desarrollo para Kinect	40
	Investigar sobre otros sensores 3D	10
		90
Encontrar plataforma óptima		
	Raspberry Pi + Kinect	52
	Netbook + Kinect	38
		90
Implementación		
	Capturar profundidad	30
	Detectar usuario	30
	Separar piernas del resto de la escena	60
	Obtención de datos esenciales	27
	Calibración del sensor	9
	Segmentación de las piernas	54
	Conexión a BD remota	30
		240
Validación/Testeo de todo el sistema		
		30
Documentación		
	Redacción informe de seguimiento	8
	Redacción de la memoria	50
		58
TOTAL		**608**

Tabla 3 Descripción de tareas

10.2.1 Planificación y viabilidad

Esta primera fase corresponde con la asignatura de gestión de proyectos (GEP). Consiste en: describir el problema a resolver, los objetivos del proyecto, su alcance, la planificación, la estimación de costes, describir los principales riesgos y describir el tipo de metodología a usar para desarrollar el proyecto.

10.2.2 Investigación de las alternativas

Consiste en empezar a indagar sobre los aspectos técnicos del hardware a usar, en este caso los sensores, y las posibles plataformas hardware y software; es decir tomar contacto con ellas, entenderlas y hacerlas funcionar parcialmente por separado. Básicamente se trata utilizar los sistemas operativos de las posibles plataformas y ver que funciones y limitaciones tienen. En el caso del sensor Kinect, hacerlo funcionar en un ordenador de sobremesa para entender cómo funcionan sus drivers y que aplicaciones existen para obtener y tratar los datos que este dispositivo produce.

10.2.3 Encontrar plataforma óptima

Una vez investigadas las posibles opciones, se trata de seguir el árbol de decisiones creado, y encontrar que combinación es la factible e idónea para acabar desarrollando el proyecto con ella.

10.2.4 Implementación

En esta fase se trata de implementar toda la solución a los objetivos, se puede diferenciar todo el grueso de visión por computador y la parte de conexión remota y envío de los datos obtenidos por los algoritmos de visión.

10.2.5 Validación de la solución

Esta fase consiste en asegurarse que los datos obtenidos son almacenados correctamente y corresponden a valores reales. Para ello se seguirán los métodos de evaluación explicados.

10.2.6 Documentación

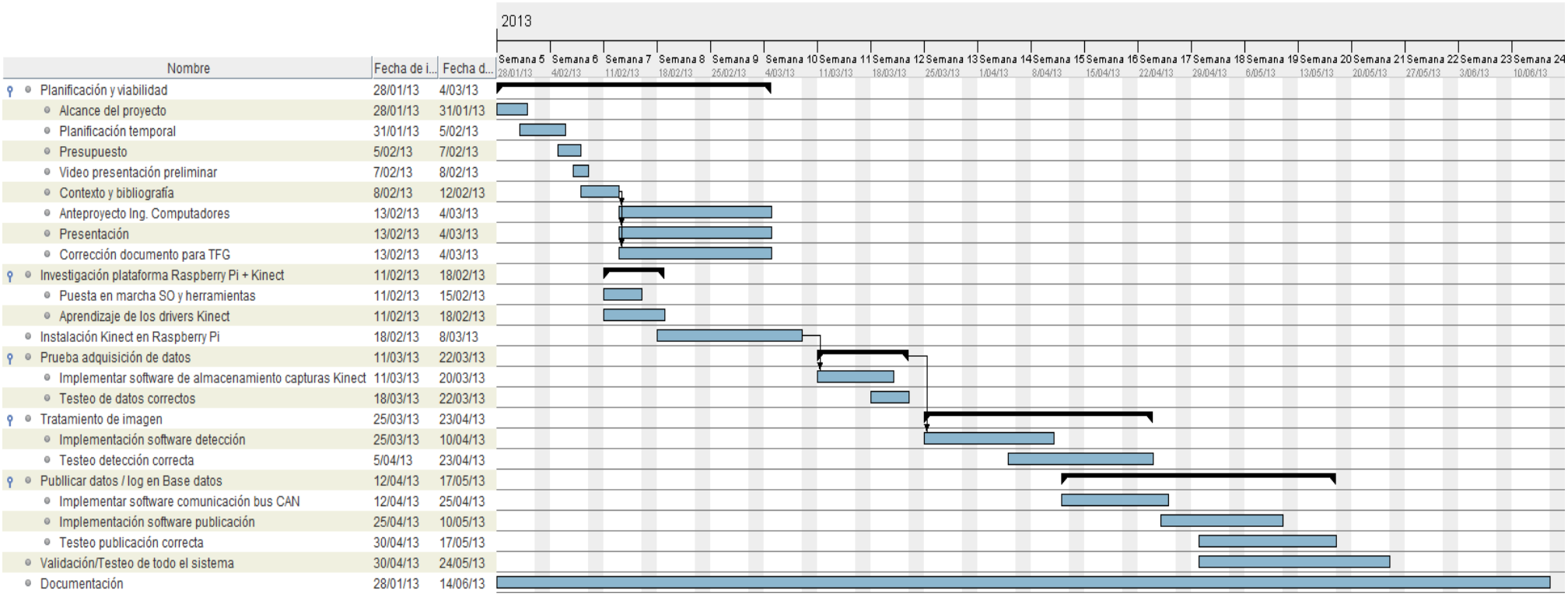
Esta tarea se realiza desde el inicio del proyecto hasta su finalización, consiste en elaborar la memoria del proyecto, incluyendo la planificación y viabilidad inicial y las propias explicaciones de los resultados del proyecto en sí. Además a mitad de cuatrimestre hay que redactar un informe de seguimiento que se ha de entregar al tutor para que vea cómo van los progresos.

10.3 Desviaciones con la planificación inicial

La planificación inicial propuesta en el curso de gestión de proyectos (GEP) varía con respecto a cómo se ha acabado llevando a cabo el proyecto. La planificación inicial se basaba en una suposición idónea en que la plataforma inicialmente escogida Raspberry Pi + Kinect funcionarían y se podría llevar a cabo el desarrollo. Sin embargo finalmente no ha sido así puesto que este binomio no ha sido factible y se ha tenido que optar por otra plataforma. De todas formas esta posible alteración ya se contaba desde el planteamiento inicial puesto que era uno de los riesgos principales y ya se dieron algunas semanas de margen para solventarlo.

Aun así existe una desviación de un mes con respecto al comienzo de la implementación de la solución. Esto se debe a que se supuso que en caso de fallar con la plataforma Raspberry el tiempo para aplicar la alternativa sobre un portátil requeriría poco tiempo. Esta estimación fue errónea puesto que tal y como he explicado hubo que realizar varias investigaciones y comprobaciones sobre diferentes plataformas hasta encontrar la idónea. Por suerte la tarea de implementación finalmente no ha requerido de tanto tiempo quizá por la presión de la fecha final acercándose, o tal vez por el amplio margen que se dio en la planificación inicial. A continuación se muestran los dos diagramas de Gantt el esperado inicialmente y el que finalmente se ha seguido.

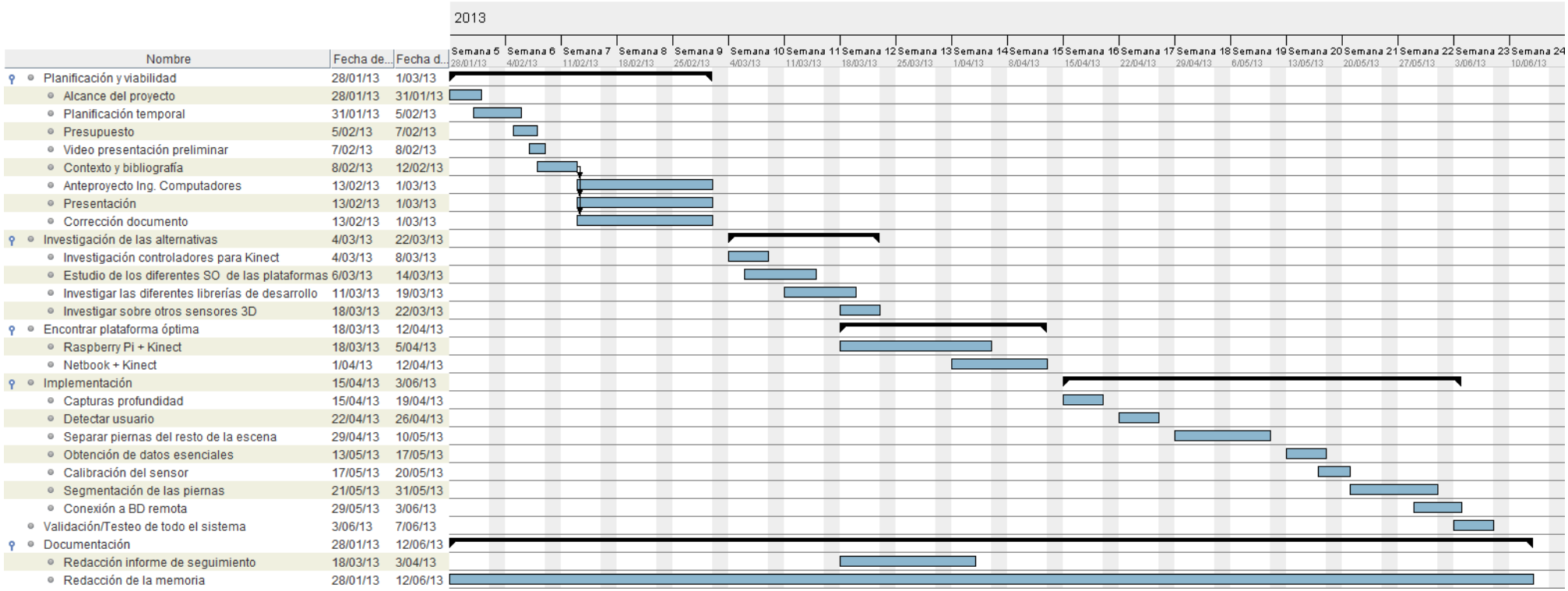
10.4 Diagrama de Gantt inicial



Nombre	Fecha de i...	Fecha d...
Planificación y viabilidad	28/01/13	4/03/13
Alcance del proyecto	28/01/13	31/01/13
Planificación temporal	31/01/13	5/02/13
Presupuesto	5/02/13	7/02/13
Video presentación preliminar	7/02/13	8/02/13
Contexto y bibliografía	8/02/13	12/02/13
Anteproyecto Ing. Computadores	13/02/13	4/03/13
Presentación	13/02/13	4/03/13
Corrección documento para TFG	13/02/13	4/03/13
Investigación plataforma Raspberry Pi + Kinect	11/02/13	18/02/13
Puesta en marcha SO y herramientas	11/02/13	15/02/13
Aprendizaje de los drivers Kinect	11/02/13	18/02/13
Instalación Kinect en Raspberry Pi	18/02/13	8/03/13
Prueba adquisición de datos	11/03/13	22/03/13
Implementar software de almacenamiento capturas Kinect	11/03/13	20/03/13
Testeo de datos correctos	18/03/13	22/03/13
Tratamiento de imagen	25/03/13	23/04/13
Implementación software detección	25/03/13	10/04/13
Testeo detección correcta	5/04/13	23/04/13
Publlicar datos / log en Base datos	12/04/13	17/05/13
Implementar software comunicación bus CAN	12/04/13	25/04/13
Implementación software publicación	25/04/13	10/05/13
Testeo publicación correcta	30/04/13	17/05/13
Validación/Testeo de todo el sistema	30/04/13	24/05/13
Documentación	28/01/13	14/06/13

Ilustración 36 Diagrama de Gantt inicial

10.5 Diagrama de Gantt final



Nombre	Fecha de...	Fecha d...
Planificación y viabilidad	28/01/13	1/03/13
Alcance del proyecto	28/01/13	31/01/13
Planificación temporal	31/01/13	5/02/13
Presupuesto	5/02/13	7/02/13
Video presentación preliminar	7/02/13	8/02/13
Contexto y bibliografía	8/02/13	12/02/13
Anteproyecto Ing. Computadores	13/02/13	1/03/13
Presentación	13/02/13	1/03/13
Corrección documento	13/02/13	1/03/13
Investigación de las alternativas	4/03/13	22/03/13
Investigación controladores para Kinect	4/03/13	8/03/13
Estudio de los diferentes SO de las plataformas	6/03/13	14/03/13
Investigar las diferentes librerías de desarrollo	11/03/13	19/03/13
Investigar sobre otros sensores 3D	18/03/13	22/03/13
Encontrar plataforma óptima	18/03/13	12/04/13
Raspberry Pi + Kinect	18/03/13	5/04/13
Netbook + Kinect	1/04/13	12/04/13
Implementación	15/04/13	3/06/13
Capturas profundidad	15/04/13	19/04/13
Detectar usuario	22/04/13	26/04/13
Separar piernas del resto de la escena	29/04/13	10/05/13
Obtención de datos esenciales	13/05/13	17/05/13
Calibración del sensor	17/05/13	20/05/13
Segmentación de las piernas	21/05/13	31/05/13
Conexión a BD remota	29/05/13	3/06/13
Validación/Testeo de todo el sistema	3/06/13	7/06/13
Documentación	28/01/13	12/06/13
Redacción informe de seguimiento	18/03/13	3/04/13
Redacción de la memoria	28/01/13	12/06/13

Ilustración 37 Diagrama de Gantt final

11 Presupuesto

11.1 Coste Hardware

Se tiene en cuenta solo el hardware utilizado finalmente, otras plataformas no se tienen en cuenta para los siguientes cálculos.

Componentes	Precio	IVA	Precio total
Asus EeePC 901	247,93 €	52,06€	300€
Microsoft Kinect	123,96€	26,03€	150€
TOTAL	**371,9 €**	**78,09€**	**450€**

Tabla 4 Desglose de costes hardware

11.2 Coste del desarrollo

Durante el proyecto han aparecido distintas etapas que se corresponden a diferentes roles.

- La primera etapa coincide con la planificación y gestión inicial del proyecto que es llevada a cabo por el director de proyecto.
- La segunda etapa corresponde a un ingeniero analista, que comprende toda la fase de investigación de plataformas.
- La tercera etapa es la de implementación que corresponde a un programador.

En la siguiente tabla se muestran los roles, las horas de trabajo y sus respectivos costes. El desarrollo se ha llevado a cabo mediante software libre de licencia por lo que tampoco se tiene en cuenta como coste.

Rol	Horas	Coste/hora	Coste
Director de proyecto	100h	60€/h	6.000€
Analista	180h	40€/h	7.200€
Programador	240h	30€/h	7.200€
TOTAL			**20.400€**

Tabla 5 Desglose de costes de desarrollo

11.2.1 Viabilidad económica

Este proyecto es una aplicación de características del proyecto general iWalker. El estudio de viabilidad realizado tiene en cuenta simplemente el trabajo aportado; es decir, se enfoca como un pack hardware + software que realiza las funciones de visión por computador explicadas.

Supondremos que podemos llegar a vender 100 packs de los comentados. El precio al que habría que vender cada pack, corresponde a la suma de coste fijo de hardware, más la amortización del software más un margen de beneficio. Por lo que el precio resultante se calcula como: 450€ + 204€ + 100€ (de beneficio) = **754€.**

Consiguiendo vender 100 packs se conseguiría amortizar el coste de desarrollo y obteniendo un beneficio de **10.000€**

12 Sostenibilidad y compromiso social

12.1 Impacto social

El proyecto desarrollado es una característica del proyecto iWalker el cual es un andador inteligente que va un paso más allá de los andadores convencionales y que permite la comunicación con el usuario, la toma de decisiones y contiene un conjunto de programas inteligentes.

La versatilidad de este andador permite adaptar al producto a las necesidades del usuario y al mercado al que se puede dirigir, que puede ser tanto el área de rehabilitación de hospitales como en un uso doméstico para la ayuda a la movilidad o de rehabilitación ya que permite compensar la pérdida sensorial, motora y las funciones cognitivas provocadas por el paso del tiempo y las enfermedades de los ancianos. También puede ayudar a reforzar y estimular las capacidades humanas y mejorar el bienestar en la vida diaria.

El iWalker incluye un sistema de vigilancia especial que permite detectar la posición del paciente en el hogar y en otros sitios conocidos, como los hospitales y centros de atención primaria. Otro de los sistemas que incluye es el tratado en este proyecto, la detección del movimiento de las piernas del usuario. Este sistema está enfocado sobre todo para que el personal médico especializado tenga un monitoreo de la forma de andar del paciente sin necesidad de estar presente. La implantación al mercado de un aparato de estas características supondría un método de rehabilitación de gran valor añadido si se llega a desarrollar a un precio competitivo.

12.2 Impacto económico

No se ha encontrado ningún producto similar en el mercado, pero sí varios centros y empresas desarrollando prototipos con características similares al iWalker, La empresa DENSO está desarrollando un andador asistido con compensación de fuerzas semejantes al presentado. También está el proyecto de la universidad Carniege Mellon que trata sobre un andador asistido con un sistema de navegación integrado.

La tecnología del andador iWalker es semejante a los anteriores, pero este incorpora unas características, entre ellas la desarrollada en este proyecto, que le dan una ventaja competitiva; sobre todo el potencial de poder ofrecer soporte para la recuperación de enfermedades que provocan problemas asimétricos de

movilidad. Por tanto se considera que el grado de innovación en el mercado es elevado.

12.3 Impacto ambiental

El impacto ambiental no se ha tratado efusivamente con este proyecto, puesto que trata de más de obtener un producto que ofrece un beneficio social real. Sin embargo ciertos aspectos considerados han servido indirectamente para reducir el impacto ambiental.

El proyecto consiste en capturar los movimientos de las piernas del usuario, para ello se utiliza un sensor 3D con el cual finalmente conseguimos reducir la información necesaria a seis puntos. Esta reducción de información disminuye considerablemente el volumen de datos que tendría que enviar el andador por la red. Reduciendo el volumen de datos enviados se consigue un menor consumo y una posible mejor autonomía del andador.

13 Conclusiones y trabajo futuro

El objetivo del proyecto era utilizar una plataforma que se integrase con el andador y principalmente realizase las tareas de visión por computador requeridas. Se han cumplido los objetivos, puesto que se obtiene la extracción de los puntos necesarios para conseguir la reconstrucción del movimiento de las piernas del usuario. Una vez calibrados estos puntos denotan unas distancias reales bastante representativas a la realidad, además estos puntos se envían mediante WiFi a una base de datos remota y también se pueden almacenar a nivel local puesto que la información resultante no es abultada; eso sí, teniendo en cuenta que la capacidad de almacenamiento del netbook es reducida. Por lo tanto el resultado general es satisfactorio.

Aunque el resultado del proyecto es satisfactorio, aún son posibles muchas mejoras y ajustes en general. De la versión actual se pueden realizar varias modificaciones a nivel del software desarrollado. La implementación se puede optimizar en varios puntos puesto que hay muchos bucles que se podrían ajustar mejor para que fuesen más conscientes con la arquitectura utilizada. Se probó durante el desarrollo alguna optimización de paralelismo a nivel de datos utilizando instrucciones *SIMD*. Por desgracia al ser la primera vez que me aventuraba es este tema, no conseguí hacer funcionar correctamente el algoritmo con este tipo de optimizaciones; por lo que decidí posponerlas para trabajo futuro.

Conseguir cierta optimización serviría para reducir la sobrecarga de trabajo añadida y mejorar el *framerate*. Sin embargo, el resultado obtenido ya es bastante satisfactorio; puesto que la sensación que da el sistema es que funciona a tiempo real, sin ningún retraso aparente. Por tanto, más que centrarse en la optimización como trabajo futuro, habría que mejorar la representación de los datos obtenidos. Con el trabajo realizado en el proyecto se obtienen los seis puntos comentados: rodillas, tobillos y punteras de los pies. Estos puntos por si solos aportan información, pero conseguir una interfaz 3D que los represente seria idóneo para entender más rápido como es el susodicho movimiento del usuario.

El proyecto ha servido para obtener una solución viable a integrar con el caminador, y a la vez para encontrar otras soluciones posibles; ya que la primera etapa trató de buscar la plataforma idónea. Al final de este proyecto mi tutor ya dispuso de un nuevo sensor; se trata de una nueva versión del comentado *Xtion.* Por lo que el trabajo futuro real, sería el de integrar este nuevo dispositivo con la plataforma inicial *Raspberry Pi*. Aplicar los procedimientos seguidos sobre esta plataforma, no debería suponer grandes cambios; debido a que los algoritmos principales están desarrollados en *C* y

traspasarlos a la plataforma de desarrollo *OpenNI2* no debería dar problemas aparentes.

A nivel personal este proyecto me ha servido para ratificar todos los conocimientos obtenidos en el grado. Ha sido un trabajo bastante completo, en el que he tenido que tocar aspectos de bastantes ámbitos. Desde temas más hardware y de sistemas operativos, como la investigación y selección de la plataforma, hasta una ligera utilización de bases de datos. Además no hay que olvidar todo el tema de desarrollo de visión por computador, el cual no lo toque durante el grado.

El hecho de ser un tema nuevo para mí me despertó curiosidad y a la vez grandes dudas de por si podía llevarlo a cabo. Pese a mi escepticismo inicial, mi tutor consiguió convencerme que la visión por computador que tenía que realizar no era nada muy alejado a tareas que ya hubiese realizado durante la carrera. Efectivamente él estaba en lo cierto ya que he podido desarrollar una solución viable partiendo prácticamente desde cero, y esa es la experiencia más importante que me llevo.

14 Glosario

Términos ordenados por orden de aparición:

UART: son las siglas de "Universal Asynchronous Receiver-Transmitter". Éste controla los puertos y dispositivos serie.

CAN: acrónimo del inglés Controller Area Network es un protocolo de comunicaciones basado en una topología bus para la transmisión de mensajes en entornos distribuidos.

RGB: es un modelo de color basado en la síntesis aditiva, con el que es posible representar un color mediante la mezcla por adición de los tres colores de luz primarios.

Framerate: es la frecuencia a la cual un dispositivo de imagen produce imagines únicas consecutivas.

ARM: es una arquitectura RISC (Reduced Instruction Set Computer) de 32 bits desarrollada por ARM Holdings.

SDK: un kit de desarrollo de software o SDK (software development kit) es generalmente un conjunto de herramientas de desarrollo de software que le permite al programador crear aplicaciones para un sistema concreto, por ejemplo ciertos paquetes de software, frameworks, plataformas de hardware, computadoras, videoconsolas, sistemas operativos, etc.

Cross platform: es un atributo conferido a los programas informáticos o los métodos de cálculo y conceptos que se implementan e inter-operar en múltiples plataformas.

Callback: es una pieza de código ejecutable que se pasa como argumento a otro código, que se espera para volver a llamar (ejecutar) el argumento en un momento conveniente.

Unix timestamp: es un sistema para describir de tiempo, que se define como el número de segundos transcurridos desde las 00:00:00 Tiempo Universal Coordinado (UTC), del 1 de enero 1970.

15 Bibliografía

15.1 Citas

1. *Sadale, R., Kolhe, R., Wathore, S., Aghav, J., Warade, S., &Udayagiri, S. (2013). Steer-by-wire implementation using kinect*
2. *Zuher, F., & Romero, R. (2012).Recognition of human motions for imitation and control of a humanoid robot.190-195.*
3. *Ning, X., &Guo, G. (2013).Assessing spinal loading using the kinect depth sensor: A feasibility study.IEEE SensorsJournal, 13(4), 1139-1140.*
4. *OpenKinect, "Getting Started" http://openkinect.org/wiki/Getting_Started (15/02/2013)*
5. *Hirotaka Niisato,Natural days, "Raspberry Pi and OpenNI", 6 Enero 2013, http://www.hirotakaster.com/archives/2013/01/raspberry-pi-and-openni2.php (5/04/2013)*
6. *Ariandy,ariandyblog, "Getting Raspberry Pi, OpenNI2, and Asus Xtion Pro Live To Work Together", Febrero 28 2013, http://ariandy1.wordpress.com/2013/02/27/getting-raspberry-pi-openni-and-asus-xtion-pro-live-to-work/ (5/04/2013)*

15.2 Kinect y visión por computador

- Massimo Camplani and Luis Salgado, "Efficient spatio-temporal hole filling strategy for Kinect depth maps", Proc. SPIE 8290, Three-Dimensional Image Processing (3DIP) and Applications II, 82900E (February 9, 2012); doi:10.1117/12.911909
- Lu Xia; Chia-Chih Chen; Aggarwal, J.K., "Human detection using depth information by Kinect," *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on* , vol., no., pp.15,22, 20-25 June 2011;doi: 10.1109/CVPRW.2011.5981811
- Newcombe, Richard A.; Davison, Andrew J.; Izadi, S.; Kohli, P.; Hilliges, Otmar; Shotton, J.; Molyneaux, David; Hodges, Steve; Kim, David; Fitzgibbon, A., "KinectFusion: Real-time dense surface mapping and tracking," *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on* , vol., no., pp.127,136, 26-29 Oct. 2011;doi: 10.1109/ISMAR.2011.6092378
- LEJEUNE, S. PIÉRARD, M. VAN DROOGENBROECK, and J. VERLY. A new jump edge detection method for 3D cameras. In International Conference on 3D Imaging (IC3D), Liège, Belgium, December 2011.

- Kiretu, “Class Description” http://pille.iwr.uni-heidelberg.de/~kinect01/doc/classdescription.html (20/05/2013)

- M.R. Andersen, T. Jensen, P. Lisouski, A.K. Mortensen, M.K. Hansen, T. Gregersen and P. Ahrendt: Kinect Depth Sensor Evaluation for Computer Vision Applications, 2012. Department of Engineering, Aarhus University. Denmark. 37 pp. - Technical report ECE-TR-6

15.3 Otras fuentes

- Jan Bodnar, Zetcode, “MySQL C API programming tutorial”, http://zetcode.com/db/mysqlc/ (27/05/2013)
- ALLSOFT, “SlideShare” Modelos de desarrollo, Enero 2008. *http://www.slideshare.net/inventa2/modelos-de-desarrollo*(28/02/2013)
- Wikipedia,“RaspberryPi” *http://es.wikipedia.org/wiki/Raspberry_Pi* (10/02/2013)
- Praveen Kumar Wilson, Praveen I Tech, “Kinect Overview” http://praveenitech.wordpress.com/2012/01/04/35/ (6/06/2013)

16 Anexos

Adjunto una copia de las instrucciones que encontré para instalar OpenKinect en Windows XP, puesto que es bastante complicado realizar el proceso sin ellas. Extraído de:

http://www.cse.cuhk.edu.hk/~khwong/www2/OSVIP/Microsoft%20Word%20-%20Kinect%20Installation%20Guideline_v0.1.pdf

16.1 Installing Driver

Before compile the libfreenect library, we have to install the hardware driver for Windows. The driver is bundled with the OpenKinect project which also include the libfreenect library source code.

1. Go to GitHub and download the OpenKinect project https://github.com/OpenKinect/libfreenect
2. Choose the stable version and download
3. Extract the files
4. Plug your kinect to your PC and make sure the power supply is on.
5. Windows will report there is an unknown device without driver and ask you to find the driver.
6. Click on "Browse my computer for driver software".
7. Browse and find the folder called “xbox nui motor” in the extracted OpenKinect Project (e.g. D:\OpenKinect\OpenKinect-libfreenect-4a159f8\platform\windows\inf)
8. Continue and Windows will ask for your permission to install the driver.
9. After a few seconds, the driver installation for “xbox nui motor” should be done. Another unknown USB device “xbox nui audio” and “xbox nui camera” will be discovered after the driver installation.
10. Repeat step 6-8 for “xbox nui audio” and “xbox nui camera”.
11. After all the drivers are installed, the LED on Kinect should be flashing.

16.2 Compile the library

Like compiling OpenCV, we use CMake to generate the Visual Studio solution files.

1. Open CMake-gui in the Windows Start menu.
2. Click the “Browse button”.
3. Select the OpenKinect project (e.g. D:\OpenKinect\OpenKinect-libfreenect-4a159f8)
4. Click the “Browse Build”.
5. Find a place to build the project or create a folder for it. (e.g. D:\OpenKinect\OpenKinect-libfreenect-4a159f8\vs2008)
6. Click the “Configure” button.
7. In the drop-down menu of the pop-up window, choose Visual Studio 9 2008 and “finish”
8. After a few seconds, CMake will report error but it is OK.
9. Check the “Group” and “Advance” box under “Browse Build”
10. Expand the “LIBUSB” entries

11. “LIBUSE_1_INCLUDE_DIR” should be pointed to the include folder extracted from libusb (e.g. D:\OpenKinect\libusb-win32-bin-1.2.4.0\include)
12. “LIBUSE_1_LIBRARY” should be pointed to the libush.lib (e.g. D:/OpenKinect/libusb-win32-bin-1.2.4.0/lib/msvc/libusb.lib). DO use the .lib file under the msvc folder.
13. Expand the “THREADS” entries
14. “THREADS_PTHREADS_INCLUDE_DIR” should be pointed to the include folder extracted from pthread-win32 in the pre-built folder (e.g. D:\OpenKinect\pthread\Pre-built.2\include)
15. “THREADS_PTHREADS_WIN32_LIBRARY” should be pointed to pthreadVC2.lib (e.g. D:/OpenKinect/pthread/Pre-built.2/lib/pthreadVC2.lib)
16. Expand the BUILD entries.
17. Uncheck the BUILD_FAKENECT entry as it is not supported in Windows currently.
18. Click the configure button again.
19. Expand the newly added entry: GLUT
20. GLUT_INCLUDE_DIR should be pointed to the include folder of Visual Studio VC (e.g. C:\Program Files\Microsoft Visual Studio 9.0\VC\include)
21. GLUT_glut_LIBRARY should be pointed to the glut32.lib directly (e.g. C:/Program Files/Microsoft Visual Studio 9.0/VC/lib/glut32.lib)
22. Click the configure button again.
23. There should be no red highlighted entry. If you have any problem with the set up, you can reference the image at the end of this guideline.
24. Click the “***Generate***” button.
25. Go to the folder where the Visual Studio Solution file is generated. (e.g. D:\OpenKinect\OpenKinect-libfreenect-4a159f8\vs2008)
26. Open the solution file and choose “***Build***” → “***Build Solution***”
27. Sometimes, you may encounter build error at the end of the compilation. Try clicking the “Build Solution” again. It may solve the problem.
28. After compiling the library, right click the “***INSTALL***” project in the Solution Explorer and choose “***Build***”.
29. After that, the examples, include files and libraries are copied to “C:\program files\libfreenect”
30. Open this folder and go to the “lib” folder.
31. Copy all the .dll files into “C:\windows\system32”
32. Go back to the “***bin***” folder. (e.g. C:\program files\libfreenect\bin)
33. Double click the glview.exe to run the example.
34. After a few seconds, you should see the depth map as well as the RGB image on the screen. You can select the image window and then press “w”, “s” and “x” on your keyboard to control the servo motor in kinects. “w” to tile up and “x” to tile down. “s” to reset the tile angle. “1” to “6” set the LED light and “f” to reset the cameras.
35. If you see a white screen but you can use the keys to control the keyboard, then you may install a newer driver for your display card. It is essential for the machine in SHB122.