ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO TÉCNICO DE TELECOMUNICACIÓN, ESPECIALIDAD EN SONIDO E IMAGEN

Título del proyecto:

ESTIMACIÓN DE LA POSICIÓN DE LA CABEZA EN TIEMPO REAL CON KINECT

Ignacio San Román Lana

Arantzazu Villanueva Larre

Pamplona, 3 de septiembre de 2012

PROPUESTA
PROYECTO FIN DE CARRERA

"ESTIMACIÓN DE LA POSICIÓN DE LA CABEZA EN TIEMPO REAL CON KINECT"

Departamento de ingeniería eléctrica y electrónica

Alumno: Ignacio San Román Lana

Tutor: Arantxa Villanueva Larre

Pamplona, 13 de enero de 2012

INDICE

OBJETO DEL PFC

En este proyecto se pretende implementar y probar un algoritmo en tiempo real que permita identificar la posición de la cabeza, dando los ángulos en los que se encuentra girada ésta, mediante el entorno de desarrollo Matlab y el sistema de imágenes de profundidad de KINECT.

DESCRIPCIÓN DEL PFC

- **Los objetivos principales del proyecto son:**
  - Entender el funcionamiento del sistema Kinect, su configuración, puesta en marcha y comunicación a través del entorno de desarrollo Matlab Así como sus limitaciones y grado de precisión.
  - Entender los procedimientos del algoritmo e implementarlos en Matlab intentando que el código sea sencillo y eficiente en su ejecución.
  - Probar la eficacia del algoritmo mediante pruebas y comparaciones con otros sistemas ya probados anteriormente, buscando posibles fallos en la ejecución.
  - Mejorar el sistema resolviendo esos fallos y hacerlo más eficiente computacionalmente.
  - Probar el nuevo código y dar un rango de funcionamiento y el error de precisión.
  - Elaborar una memoria que contengan todos los datos obtenidos y explique la manera de instalar y ejecutar el sistema.

- **Estructura del PFC:** El proyecto comenzará con una introducción y unos objetivos que marquen el camino que se ha seguido. Después se explicará el funcionamiento del sistema Kinect para obtener un mapa de profundidad, su configuración y la forma de comunicarse con el sistema. A continuación se explicará el algoritmo y su implementación, así como sus limitaciones y errores de precisión. Para terminar se darán unas conclusiones y se dará un pequeño manual para que el experimento sea reproducible.

- **Etapas de desarrollo:** Primero se debe hacer un estudio previo de cómo funciona el sistema KINECT y la forma de configurarlo. Después debe programarse el código según el algoritmo en Matlab. Más tarde se probará la eficacia del algoritmo implementado en Matlab mediante otro sistema ya probado de posicionamiento de la cabeza. Se resolverán los posibles fallos o problemas y se volverá a probar hasta que se pueda dar un programa libre de errores. Para finalizar se realizará la memoria que contendrá todos los datos obtenidos y explicará el funcionamiento y puesta en marcha.

EQUIPOS QUE SE NECESITA UTILIZAR PARA LA REALIZACIÓN DEL PFC:

Los únicos equipos necesarios son el sistema KINECT de Microsoft y un ordenador con el entorno de desarrollo Matlab, en el que se programara el código.

BIBLIOGRAFÍA

- REAL-TIME HEAD POSE ESTIMATI ON USING DEPTH MAP FOR AVATAR CONTROL. Yu Tu, Chih-Lin Zeng and Che-Hua Yeh. National Taiwan University

AGRADECIMIENTOS

Gracias a Arantxa, mi tutora en este proyecto, por todo lo que me ha ayudado desde el primer momento, por todo el tiempo que me ha dedicado y por haberme introducido en un mundo nuevo para mí.

Gracias a mis compañeros de clase, especialmente a Eduardo y a Iñigo, han hecho mucho más fácil estos años en los que hemos estado juntos en clase.

Gracias a mi familia, sobre todo a mis padres, por indicarme el camino correcto con sus consejos y ser los responsables de la persona en la que me he convertido.

Gracias a Estíbaliz, por confiar siempre en mí y estar siempre a mi lado. Por saber ver todo mi potencial y no dejarme desistir. Sin ella no habría llegado hasta aquí.

ÍNDICE

1.- MARCO DEL PROYECTO

1.1 Introducción

Kinect (Figura 1.1.1) es un sistema desarrollado por Microsoft para la consola XBOX 360 y puesto a la venta en Noviembre de 2010 que permite a los usuarios controlar e interactuar con la consola sin necesidad de mandos. Para ello usa una serie de cámaras y micrófonos junto con una tecnología capaz de crear mapas de profundidad. Éstos nos permiten saber lo alejado que se encuentra cada punto de la escena y como consecuencia conocer su posición en las tres dimensiones del espacio.

A partir de estos mapas de profundidad, mediante una interfaz de usuario, el sistema reconoce personas, caras, gestos, objetos, etc. Además la interfaz de usuario también puede reconocer voces y órdenes gracias al micrófono que lleva incorporado.

Figura 1.1.1: Sistema Kinect de Microsoft

Aunque fue diseñada en un principio para la consola, hoy en día existen distintos sistemas y drivers para su utilización en PC. Así mismo, también existen distintos SDK o Kits de desarrollo de software que permiten crear aplicaciones que utilizan esta tecnología.

Utilizando la tecnología de la Kinect, se quiere implementar un algoritmo en Matlab que reconozca la posición de la cabeza y calcule los ángulos en los que se tiene girada ésta en tiempo real. Para ello se parte del trabajo previo de Yu Tu, Chih-Lin y Che-Hua Yeh en su trabajo “Real-Time head pose estimation using depth map for avatar control” en la National Taiwan University [1].

En este proyecto, utilizando únicamente los mapas de profundidad, se calculan los ángulos roll, pitch y yaw (Figura 1.1.2), que permiten controlar un avatar de forma que éste gira la cabeza al unísono con el usuario.

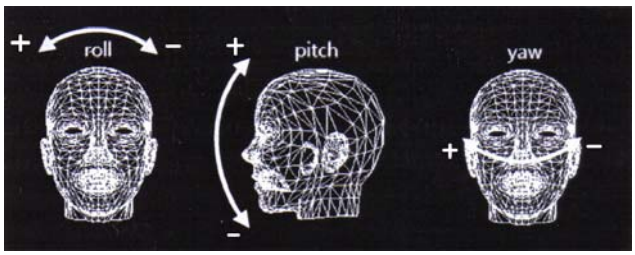


Figura 1.1.2: Ángulos roll pitch y yaw

1.2 Objetivos

El objetivo principal de este proyecto es la implementación y evaluación del algoritmo de Yu Tu y otros [1] que permite identificar la posición de la cabeza mediante el sistema Kinect. El programa localiza la cabeza y después calcular los ángulos en los que se encuentra girada (Pitch, yaw y roll). Para ello se han seguido los siguientes pasos:

1. Como punto de partida es fundamental hacer un estudio previo del sistema Kinect. En él se debe investigar cómo se crean los mapas de profundidad mediante el patrón de puntos proyectado y la cámara infrarroja. También se debe estudiar los distintos drivers y soluciones para controlarla con un ordenador con sistema operativo Windows XP.
2. Una vez el sistema Kinect esté conectado al ordenador y operativo, se buscará la forma de controlarlo a través del entorno de desarrollo Matlab. El objetivo principal es poder importar las imágenes de profundidad directamente a Matlab. También se buscará el poder controlar los distintos parámetros, configuraciones y opciones directamente desde este entorno.
3. Cuando se pueda establecer la comunicación entre Matlab y Kinect, se estudiará el algoritmo.
4. El algoritmo se implementará tanto para un sistema online como para uno offline.
5. Cuando ya esté programado el algoritmo se realizarán pruebas en laboratorio con cabeza de maniquí de porexpan. En este proceso se calculará el rango de funcionamiento tanto de la profundidad como los ángulos máximos soportados y el error cometido entre varias mediciones de una misma situación, lo cual permite calcular la resolución máxima y la robustez del sistema.
6. Se implementará un nuevo algoritmo tanto online como offline que solucione los problemas que pueda tener el propuesto por Yu Tu y otros [1].
7. Se realizarán las mismas pruebas a este algoritmo propuesto que al de Yu Tu y otros [1] para poder comparar las mejoras.

2.- EL SISTEMA KINECT DE MICROSOFT

2.1 Descripción del sistema Kinect

El sistema Kinect es una barra de unos 30 cm sobre una base que cuenta en su parte frontal con un proyector de láser infrarrojo de clase 1 con un patrón de puntos y dos sensores CMOS. Uno monocromo para capturar las imágenes infrarrojas y otro RGB para las de color. Además cuenta con un micrófono de cuatro receptores dispuestos bocabajo, uno en un lado y los otros tres en el otro (Figura 2.1.1). El encargado de recibir y procesar todos los datos obtenidos es un chip de PrimeSense (Figura 2.1.2). Éste se encarga de controlar todos los dispositivos y de crear los mapas de profundad. Este chip está conectado mediante USB 2.0 a un procesador que ejecuta un programa patentado de Microsoft para reconocimiento facial, reconocimiento de voz, seguimiento de personas y más aplicaciones que usa la XBOX 360 en sus juegos. Además dispone de un motor con varios engranajes que puede mover la cámara arriba y abajo para seguir al jugador y un acelerómetro que le indica la posición en la que se encuentra.

Figura 2.1.1: Partes del sistema Kinect

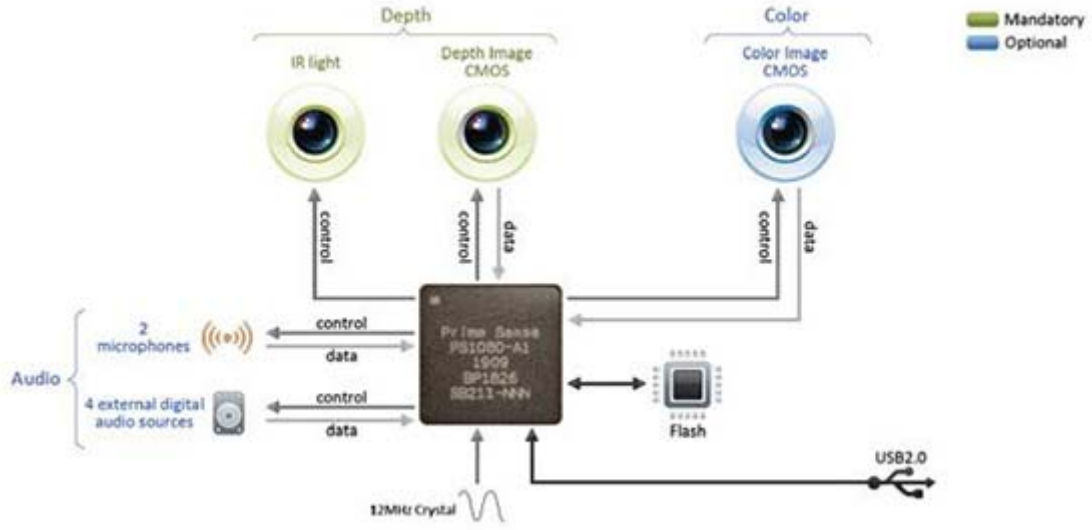


Figura 2.1.2: Conexiones internas del sistema Kinect con el chip de PrimeSense

Los sensores CMOS trabajan a un máximo de 30 fps y tienen una resolución de 640x480 pixeles. El campo de visión de cada cámara es en horizontal de 57º y en vertical de 43º aunque el motor tiene un rango de inclinación de 27º. El sistema de profundidad tiene una resolución de 320x240 pixeles en horizontal y vertical y de 1cm en profundidad. Trabaja desde los 50 cm hasta los 10 m, aunque se estima que sólo puede hacerlo hasta los 4 m con la resolución de 1 cm. El audio se capta a 16 bit por muestra a 16 KHz.

En cuanto a las conexiones, la Kinect dispone de un conector especial, que puede verse en la figura 2.1.3, por el cual discurre una conexión USB 2.0 normal más un cable de alimentación. Si se posee una XBOX 360 SLIM se puede conectar directamente el cable de datos al puerto especial que tiene para ello. En caso contrario, este cable se debe conectar a otro que provee de un enchufe de alimentación y un puerto USB 2.0 por separado, como puede verse en la figura 2.1.4. Esta conexión es necesaria ya que la Kinect necesita una alimentación superior a la que puede proporcionar un USB 2.0, por lo que debemos usar este sistema para la conexión a PC.

Figura 2.1.3: Conector específico de la Kinect

Figura 2.1.4: Cable extra para conexión en PC

2.2 Creación de los mapas de profundidad

El procesamiento para la creación de los mapas de profundidad se hace mediante lo que han denominado “codificación de luz” que consta de dos fases. La primera es una calibración (Figura 2.2.2) que consiste en proyectar el patrón de puntos (Figura 2.2.1) sobre una superficie variando la distancia a la que se encuentra en unas posiciones conocidas. Por cada posición se saca una imagen infrarroja que se guardará en la memoria de la Kinect y servirá de referencia. De esta forma, mediante triangulación, conociendo la distancia que separa el proyector de la cámara infrarroja (75 mm), la focal de ésta (750 mm) y la distancia de la cámara a los puntos proyectados, se puede saber dónde se proyectaría cada punto en el proyector si éste fuese en realidad una

cámara infrarroja igual a la otra. De esta forma se sabe exactamente la posición en la que se proyecta cada punto conociendo su patrón.

Este proceso se hizo mediante la interfaz de XBOX 360 debido a la facilidad y rapidez de este sistema.

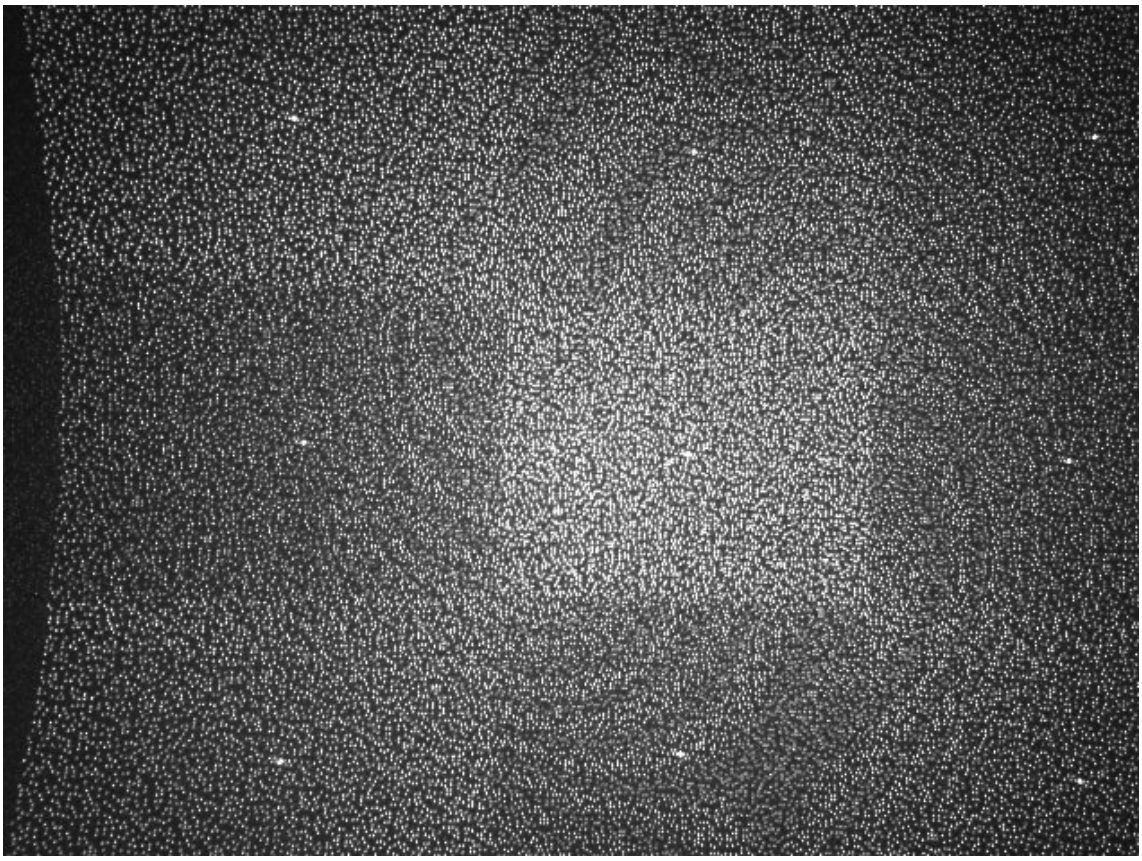


Figura 2.2.1: Imagen de los puntos infrarrojos proyectados en una pared

En la segunda fase, la de funcionamiento (Figura 2.2.3), todo se reduce a un problema de visión estéreo. El chip de PrimeSense busca los puntos en la imagen infrarroja y compara la posición de los puntos proyectados (mediante las imágenes de referencia) y la posición de los puntos en la proyección de la imagen obtenida, calculando cuanto se ha desplazado cada punto. Para ello utiliza una matriz de correlación de 9x9 píxeles que da la disparidad entre los puntos de ambas escenas. Este método crea unas sombras de 5 pixeles en todo el borde de la imagen debido a que para esos píxeles parte de la máscara cae fuera de la imagen, por lo que no existe información suficiente para hacer la correlación. Después mediante triangulación activa calcula la distancia a la que se encuentra proyectado cada punto. Finalmente con la información de todos los puntos se construye la imagen.

Esa triangulación activa para calcular la profundidad (z), utiliza la siguiente ecuación.

$$z = b * f / (\frac{1}{8} * (doff - kd))$$

Donde b es la distancia entre el proyector infrarrojo y la cámara IR (75 mm), f es la focal de la cámara IR (750 mm), kd es la disparidad entre los puntos de ambas imágenes y doff es la disparidad característica de la Kinect (1090) [7].

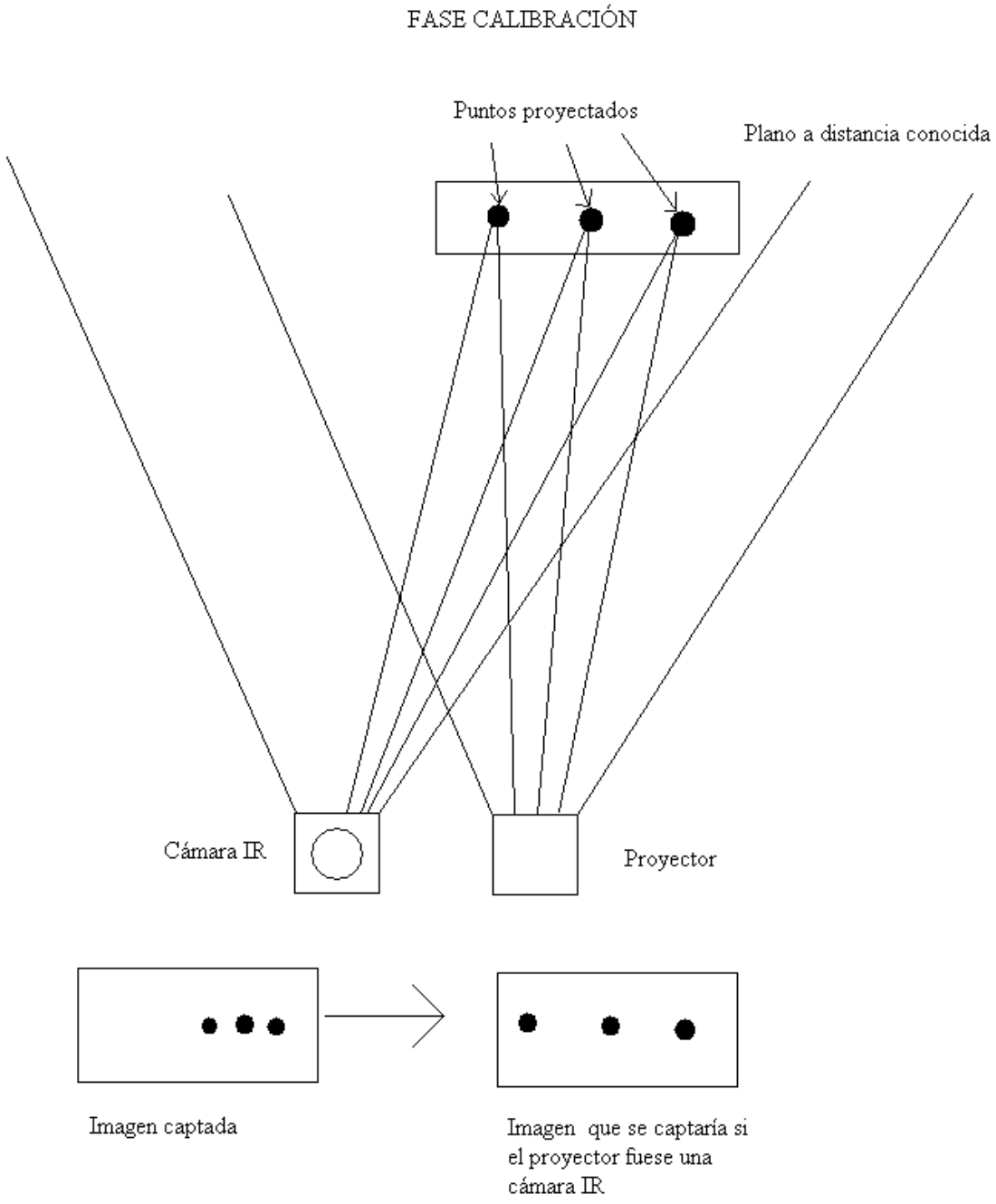


Figura 2.2.2: Fase de calibración del sistema de mapas de profundidad de la Kinect

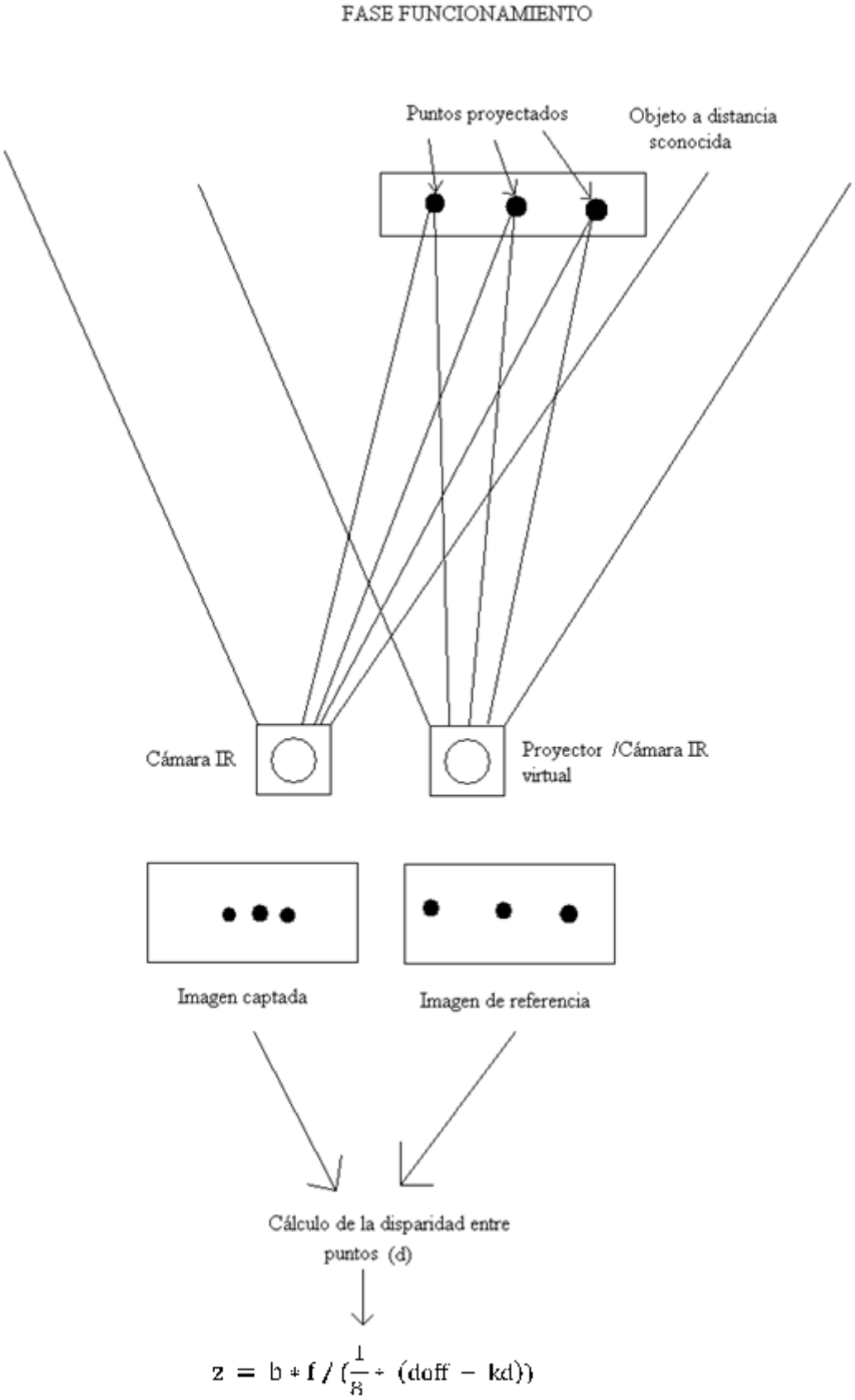


Figura 2.2.3: Fase de funcionamiento del sistema de mapas de profundidad de la Kinect

2.3 Puesta en funcionamiento de Kinect, drivers y funciones en Matlab

En este proyecto se ha decidido utilizar Windows XP con la versión de Matlab R2009a. Como se trata de un Windows anterior al Windows 7, no existen drivers oficiales para la Kinect, pero, sin embargo, hay varias soluciones posibles. Se ha elegido el sistema creado por OpenNi ya que con éste es más fácil interactuar con la Kinect a través de Matlab ya que está enteramente programado en C.

Este sistema dispone de tres partes o paquetes, y aunque se pueden instalar los tres, en este proyecto sólo hacen falta dos. Por un lado está el llamado SensorKinect, que son los drivers propiamente dichos de las cámaras, el proyector infrarrojo, el micrófono y el motor. La segunda parte, llamada OpenNi es una interfaz intermedia que incluye un servidor con el que comunicarse entre Kinect y el ordenador de forma sencilla en tiempo real y un SDK o Kit de desarrollo de software para crear aplicaciones nuevas. Es el encargado de organizar la información que provee la Kinect y de configurarla, de forma que en la programación sólo haya que hacer una llamada para obtener una imagen o mandar un archivo de configuración. La tercera parte, con nombre NITE, y que no es necesario instalar, es un conjunto de aplicaciones sencillas que sustituyen al software propietario de Windows y que hacen cosas como seguimiento de objetos, detección de rostros y personas… etc. A estas aplicaciones se les llama Middleware porque se encuentran entre la adquisición de datos y la aplicación final como puede verse en la figura 2.3.1.

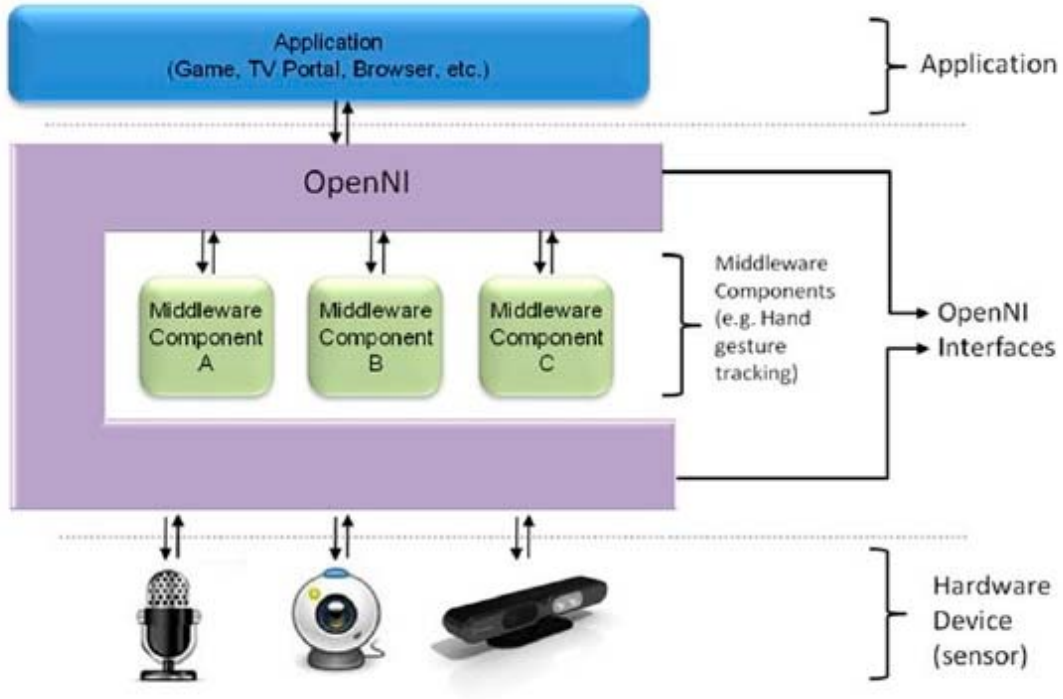


Figura 2.3.1: Funcionamiento drivers OpenNi para Kinect

Para la comunicación desde Matlab se han utilizado las funciones del proyecto Kinect-Mex. Éstas son una serie de funciones escritas en C con las cuales se puede configurar la Kinect, obtener imágenes de profundidad, IR y RGB y obtener y recibir otra serie de datos de la Kinect. Son muy importantes, ya que proporcionan la información necesaria directamente al entorno Matlab.

2.4 Configuración de Kinect

El sistema Kinect tiene una gran cantidad de opciones que se pueden configurar. Todas éstas deben ser mandadas primero al servidor de OpenNi, que se encargará de configurar la Kinect según las especificaciones.

Antes de poder enviar o recibir datos de la Kinect hay que inicializar el servidor y la Kinect con un archivo de inicialización. Este archivo se puede crear desde Matlab con la función *mxNiCreateContext* de Kinect-Mex. Para ello hay que crear una estructura que contenga la resolución de las imágenes que se quieren obtener, la cantidad de frames por segundo a la que se quiere capturar y si se desea utilizar la opción espejo. Las opciones que se pueden elegir son 640x480 a 30fps o 1280x1024 a 15fps. En el caso de que la resolución sea mayor que la que puede proporcionar la Kinect, el servidor de OpenNi se encarga de reescalarlas.

La configuración elegida para este proyecto ha sido de 640x480 a 30fps y los comandos que hacen tal cosa son los siguientes:

```
info.image_node.width = 640;
info.image_node.height = 480;
info.image_node.fps = 30;
info.image_node.mirror = true;
context = mxNiCreateContext(info, 0);
```

En el caso de que se quieran alinear las imágenes RGB y de profundidad, como ha sido el caso en este proyecto, ya que así se pude representar la información obtenida a partir de la imagen de profundidad sobre la imagen RGB para que se vea todo más claro, se puede crear otra estructura y usar la función *mxNiupdateContext* cada vez que se vayan a adquirir las imágenes. Los comandos que hacen esta operación son los siguientes:

```
option.adjust_view_point = true;
mxNiUpdateContext(context, option);
```

Con la función *mxNiSetProperty* se pueden configurar otras muchas opciones. En este proyecto sólo se ha utilizado la de *'OutputFormat'* que permite que en la imagen de profundidad los puntos más cercanos sean los que se representen como más oscuros o más claros según se le dé el valor 1 o 0. En este caso se le ha dado el valor 1, con lo cual, los puntos más cercanos serán más oscuros que los lejanos. Esto se ha hecho así por comodidad a la hora de procesar las imágenes.

```
mxNiSetProperty(context, 'OutputFormat', 1);
```

El resto de opciones que pueden configurarse son:

- *Gain:* Para aplicarle una ganancia a la imagen de profundidad. Por defecto tiene valor 1 (sin ganancia)
- *HoleFilter:* Un filtro para quitar agujeros en la imagen de profundidad.
- *MinDepthValue*: El mínimo valor que se representará en la imagen de profundidad. Sirve para establecer un rango de profundidad de funcionamiento. Por defecto tiene el valor 0.
- *MaxDepthValue:* El máximo valor que se representará en la imagen de profundidad y como el *MinDepthValue* sirve para establecer un rango de funcionamiento. Por defecto tiene el valor 10000 (10 metros).
- *WhiteBalancedEnabled:* Ajuste el balance de blancos automáticamente en la cámara RGB.

3.- ALGORITMOS IMPLEMENTADOS

3.1 Algoritmo de Yu Tu, Chih-Lin Zeng y Che-Hua Yeh[1]

Yu Tu, Chih-Lin Zeng y Che-Hua Yeh proponen un algoritmo en el que proceso para calcular el ángulo roll es independiente del proceso para el ángulo pitch y yaw. Para el ángulo roll se mide la inclinación de una elipse creada a partir de los contornos de la cara, mientras que los ángulos pitch y yaw se calculan a partir del vector perpendicular de un plano creado a partir de unos puntos cercanos a la nariz y pertenecientes a la cara.

El algoritmo propuesto por Yu Tu y otros [1] empieza adquiriendo la imagen de profundidad (Figura 3.1.1), para lo que previamente hace falta haber inicializado la Kinect como se indica en el apartado 2.4. Después se realiza un preprocesado de la imagen en el que se convierten las distancias horizontales y verticales medidas en pixeles (ejes x e y) en coordenadas cartesianas medidas en milímetros, donde el pixel superior izquierdo de la cámara infrarroja de la Kinect es el propio origen tal y como se muestra en la Figura 3.1.2.

Figura 3.1.1: Imagen de profundidad capturada desde la Kinect

Figura 3.1.2: Origen coordenadas en Kinect

Para el proceso de conversión en coordenadas cartesianas, se utiliza la distancia focal (f), que es de 575 mm, y la profundidad (z), calculada según la fórmula del apartado 2.2, ya que se calculan las distancias reales de la imagen (x, y, z) a partir de la proyección en el plano focal (X e Y) como se ve en la figura 3.1.3 según la siguiente ecuación derivada de las ecuaciones generales de lentes convergentes.

$$p(x, y, z) = (z\frac{X}{f}, z\frac{Y}{f}, z)$$

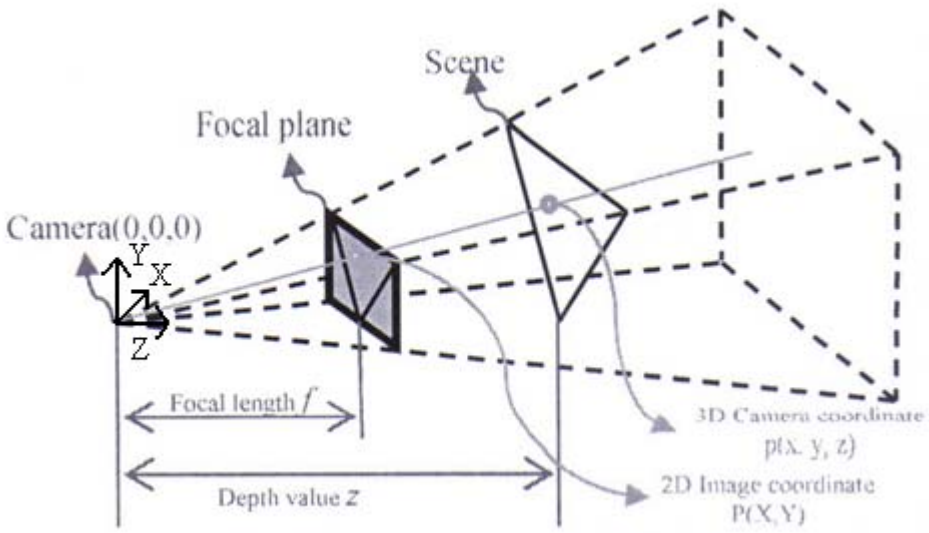


Figura 3.1.3: Proyección de la escena en el plano focal

Además, para trabajar con comodidad en la imagen, se ha decidido pasar toda la imagen de profundidad a un rango de 0 a 1, comprimiéndola dentro de estos valores, de forma que el máximo pase a ser 1, y se han eliminado las sombras negras (0 en la imagen) pasándolas a blanco (1 en la imagen) para que después no se confundan con el punto más cercano, que será el pixel con menor valor de la imagen según la configuración establecida en el apartado 2.4. Estas sombras son creadas por la falta de información al

encontrarse el objeto más próximo en medio de la trayectoria de los puntos proyectados en el fondo con la cámara infrarroja. El resultado de ésta operación se puede ver en la figura 3.1.4.

Figura 3.1.4: Imagen de profundidad después de quitarle las sombras negras y pasarlas a blanco

Después de este proceso hay que segmentar la cabeza, y aunque en el artículo no se explica cómo, lo más fácil, y por lo tanto lo que se ha hecho, es una función que haga una especie de umbralización y un stretching (Figuras 3.1.5 y 3.1.6) que coge desde el punto más cercano (asumiendo que no hay ningún objeto entre la cabeza del sujeto y el sensor) hasta unos cuatro centímetros por detrás, o lo que es lo mismo desde el mínimo de la imagen hasta 0.0009 por detrás. Esta medida de 4 centímetros no se puede medir con exactitud ya que el rango 0 a 1 de la imagen no representa una distancia fija, sino variable según el punto más alejado que se haya encontrado en la imagen de profundidad original. El valor de 0.0009 resultó el más acertado tras haber probado con distintos valores en condiciones estándar de trabajo.

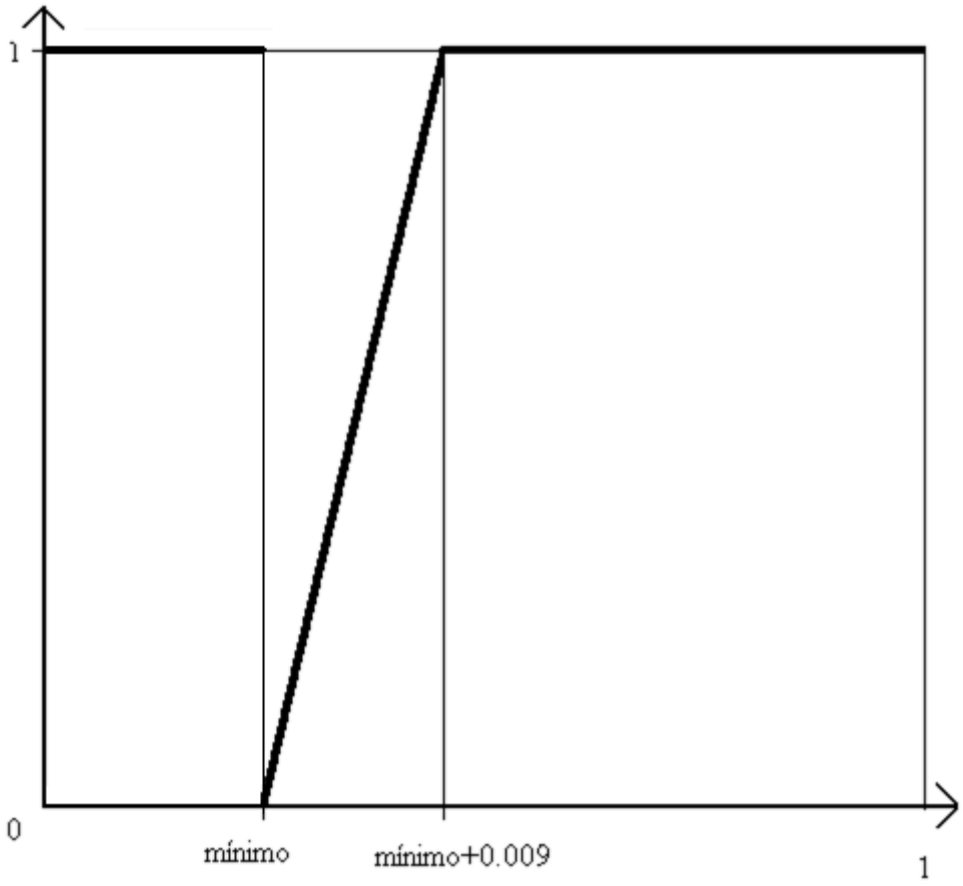


Figura 3.1.5: Función que se aplica a la imagen para la segmentación de la cabeza

Figura 3.1.6: Imagen de la cara segmentada mediante la función propuesta

A partir de este punto el algoritmo se divide en dos. Por un lado se procede a calcular el ángulo denominado de rotación o roll y por otro a calcular los ángulos pitch y yaw.

Para el roll primero hay que hacer un gradiente para obtener los contornos de la cabeza según el trabajo de Yu Tu y otros [1]. Como no especifican qué tipo de gradiente, se han utilizado un proceso de dos pasos para obtener el contorno. El primero es una umbralización al 70% con el fin de separar la cara del fondo. Se he elegido ese valor para quitar el cuello o al menos parte de éste, ya que en el artículo en ningún momento se menciona cómo debe eliminarse y éste hace que la elipse no se ajuste bien a la cara. El segundo es un filtro LoG (Laplaciano del Gaussiano) de máscara cuadrada de 3 pixeles y desviación típica de 0.1, cuyo resultado se observa en la figura 3.1.7.

Figura 3.1.7: Imagen de los bordes de la cara

Posteriormente se calcula una elipse por error cuadrático medio (mediante una función ya implementada en Matlab) a partir de los puntos del contorno obtenidos, cuyo resultado se muestra en la figura 3.1.8. Para que no afecte el ruido a la creación de la elipse, en el algoritmo de Yu Tu y otros [1], se propone hacer un filtrado paso bajo de la lista de puntos del contorno, tal y como muestra la siguiente ecuación.

$$(x_i, y_i) = \frac{\sum_{k=i-l}^{i+l}(x_k, y_k)}{2l+1}$$

A efectos prácticos no se han notado diferencias entre hacer este filtrado y no, por lo que el valor que se ha cogido de l ha sido 0, de forma que no filtra ningún punto.

Una vez se tiene la elipse, se calcula su inclinación mediante el ángulo comprendido entre el semieje menor y la horizontal de la imagen.

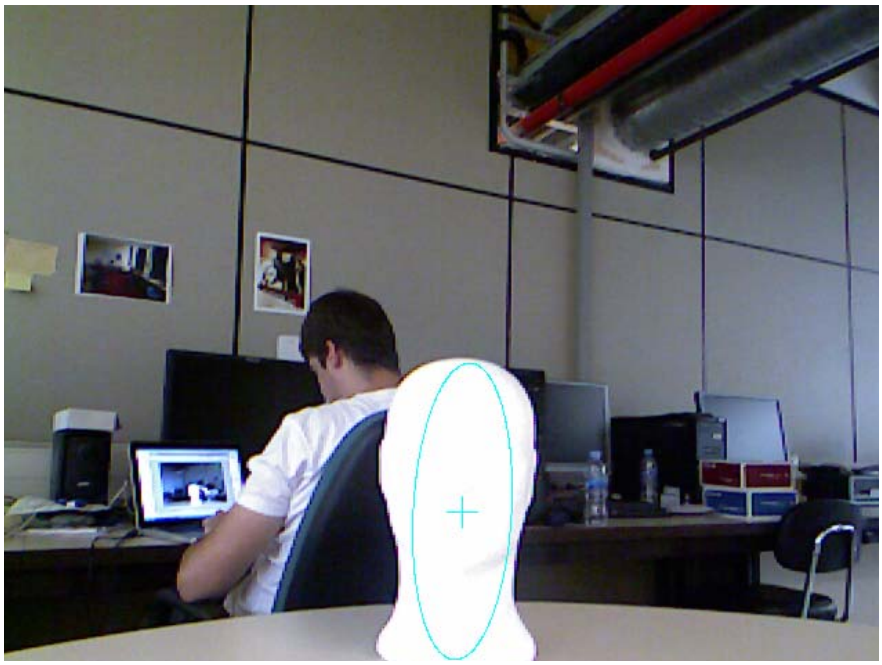


Figura 3.1.8: Imagen de la elipse de la cara

Para el cálculo de los ángulos pitch y yaw, primero hay que buscar la nariz. Para ello se coge el punto más cercano a la Kinect, por lo que se entiende que la cabeza no debe estar girada. Después se coge una nube de 300 puntos aleatorios de alrededor de la nariz pertenecientes a la cara y se calcula el plano por error cuadrático medio (mediante una función ya implementada en Matlab), como se ve en la figura 3.1.9, cuya representación matemática es la siguiente.

$$Ax + By + Cz = 0$$

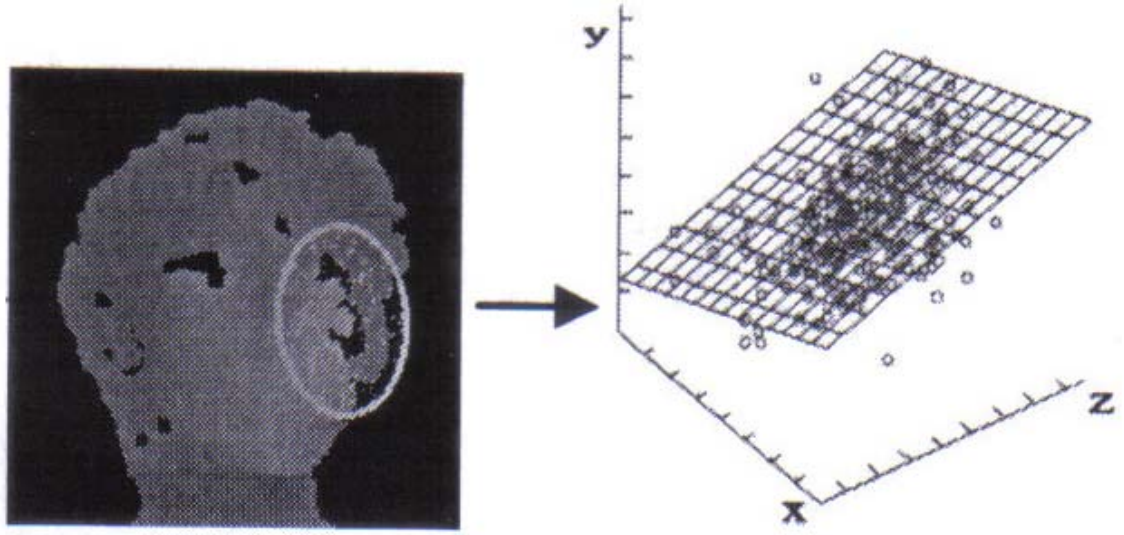


Figura 3.1.9: Nube de puntos y creación del plano por error cuadrático medio

Como en el trabajo de Yu Tu y otros [1] no se indica qué área se considera “alrededor de la nariz” se ha escogido un área de 21x21 píxeles con la nariz en el centro como se observa en la figura 3.1.10, y se han generado los 300 puntos dentro de ésta.

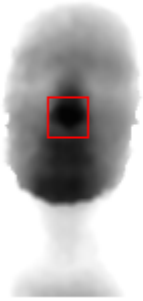


Figura 3.1.10: Área de 21x21 píxeles donde se ha creado la nube de puntos

Para calcular los ángulos se toma el vector perpendicular al plano (A, B, C) y se normaliza para que C sea igual a -1 (-A/C, -B/C, -1). De esta forma ya sólo queda calcular el ángulo que se forma entre las proyecciones del vector en los planos xz e yz (Figura 3.1.11) con los ejes x e y respectivamente. Esto se puede hacer mediante las siguientes fórmulas.

$$Pitch = \cos^{-1}\frac{1}{\sqrt{\left(\frac{-A}{C}\right)^2 + 1^2}} \qquad Yaw = \cos^{-1}\frac{1}{\sqrt{\left(\frac{-B}{C}\right)^2 + 1^2}}$$

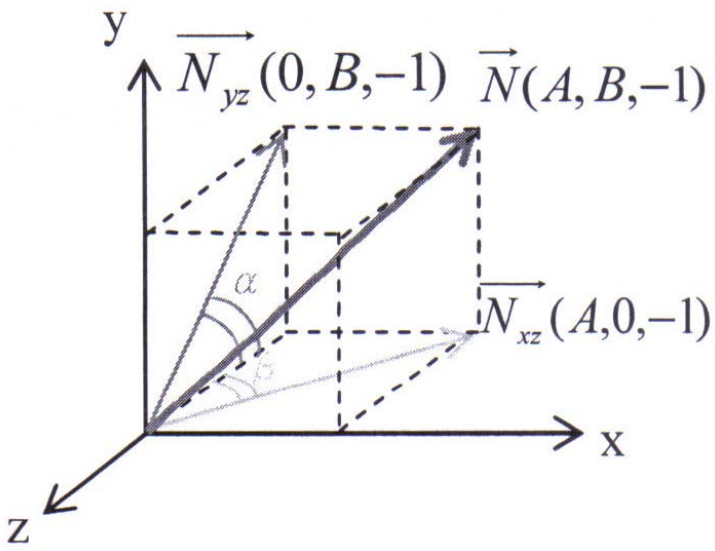


Figura 3.1.11: Proyección del vector perpendicular al plano normalizado en los planos xz e yz

En este punto ya tenemos los tres ángulos obtenidos, pero sin embargo, no tiene sentido, ya que hemos supuesto que la cabeza no se encuentra girada para poder calcular la posición de la nariz. Este proceso es el seguido para el primer frame, pero para los siguientes, para que se pueda girar la cabeza, se debe corregir la rotación en la imagen tridimensional, de forma que la cabeza se quede mirando al frente, para que la nariz sea el punto más cercano a la Kinect y poder continuar con el mismo proceso. Suponiendo que el ángulo entre frame y frame no puede haber cambiado mucho, se usan los valores de pitch y yaw del frame anterior para esa corrección en la rotación. Para ello se utilizan dos matrices de rotación multiplicadas entre ellas cuya fórmula es la siguiente.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos(yaw) & 0 & -sin(yaw) \\ 0 & 1 & 0 \\ sin(yaw) & 0 & cos(yaw) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(pitch) & sin(pitch) \\ 0 & -sin(pitch) & cos(pitch) \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Para que la transición al girar la cabeza sea más suave se debe hacer un filtrado de media de los resultados obtenidos con los frames anteriores. Como el tiempo de ejecución era muy alto, sólo se han cogido dos frames para el filtro.

El diagrama de todo el proceso puede verse en la siguiente figura 3.1.12

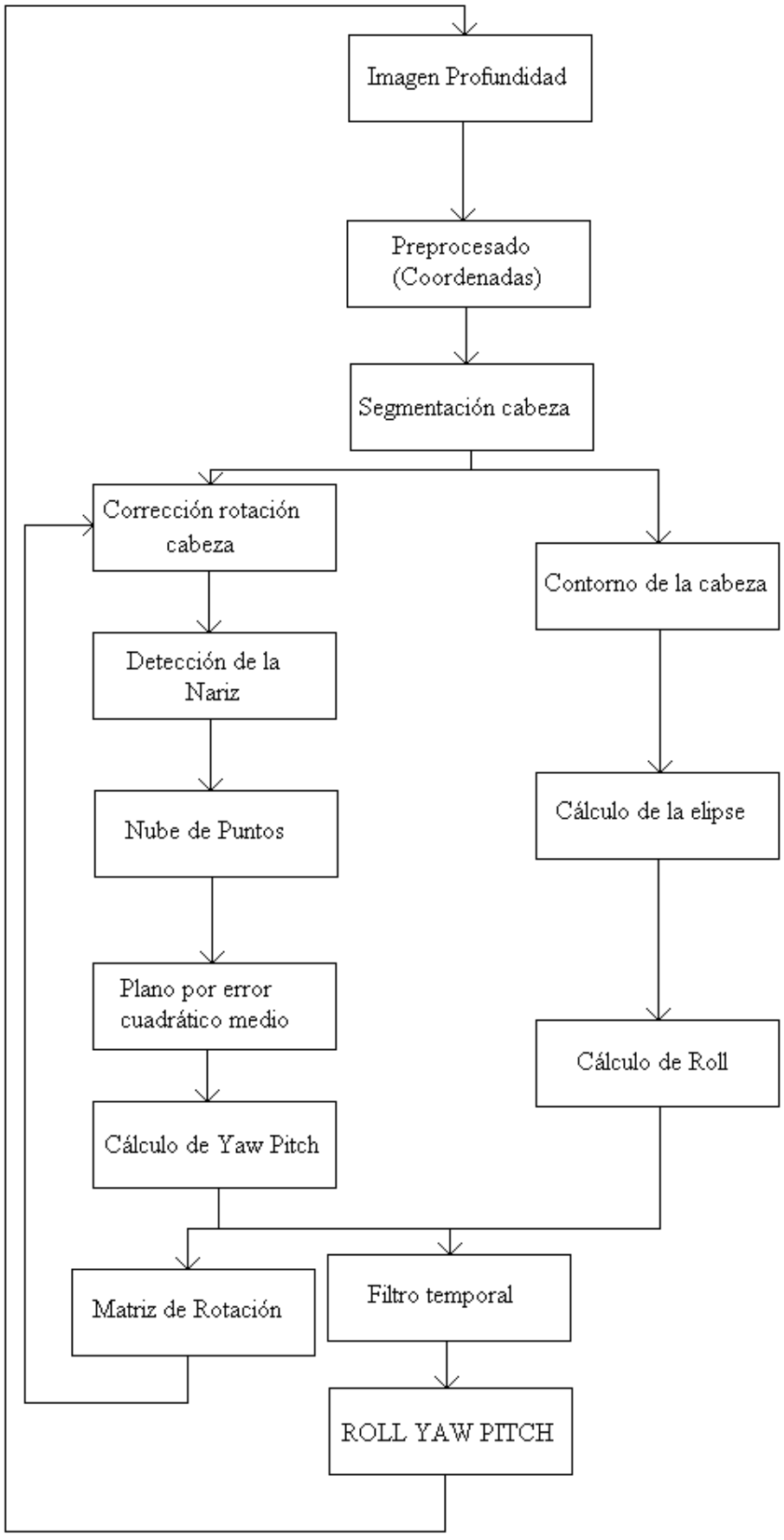


Figura 3.1.12: Diagrama del algoritmo de Yu Tu, Chih-Lin Zeng Y Che-Hua Yeh [1]

3.2 Algoritmo propuesto

Tras un breve análisis del algoritmo de Yu Tu, Chih-Lin Zeng y Che-Hua Yeh 1] se observó que no se cumplía una de las premisas que hiciese que funcionase correctamente, al menos en la implementación realizada. Esta premisa era que el tiempo entre frame y frame era muy pequeño, lo suficiente como para que el ángulo de rotación de la cabeza apenas hubiese cambiado. Esto es debido a que Matlab es un lenguaje muy lento en su ejecución, por lo que se intentó modificar parte del algoritmo para su correcto funcionamiento. Además la elipse no se ajustaba correctamente a la cara, ya que el algoritmo no era capaz de eliminar el cuello, ni siquiera después de la umbralización al 70% que se realizaba por iniciativa propia.

El algoritmo propuesto empieza inicializando la Kinect, como se explica en el apartado de configuración (2.4). Después se procede a obtener las imágenes de profundidad (Figura 3.1.1) y de color, igual que en el algoritmo de Yu Tu otros [1]. A partir de aquí se hace un preprocesado de la imagen de profundidad en el que se pasa toda la imagen a un rango de 0 a 1 y se eliminan las sombras negras (figura 3.1.4) tal y como se ha procedido en el otro algoritmo (apartado 3.1).

A continuación se hace un pequeño filtro paso bajo de media de 5 pixeles para quitar parte del ruido (Figura 3.2.1), a la vez que se calculan las coordenadas en milímetros dentro de los tres ejes, tal y como se explica en el apartado 3.1 algoritmo de Yu Tu y otros [1], que servirán para calcular después los ángulos pitch y yaw.

Figura 3.2.1: Imagen de profundidad después del filtro paso bajo de 5 pixeles

Después si la imagen contiene algún objeto (la imagen no es blanca, lo que supondría que no hay nada que se encuentre dentro del rango de funcionamiento de la Kinect), se procede a una primera segmentación de la cabeza y parte del cuerpo mediante una especie de umbralización y un stretching (Figuras 3.2.2 y 3.1.5) que coge desde el punto más cercano hasta unos cuatro centímetros por detrás, tal y como se ha hecho en el algorimo de Yu Tu y otros en la sección 3.1.

Figura 3.2.2: Imagen de la cara segmentada mediante la función propuesta

A partir de aquí se empieza con el proceso para calcular el roll. En este proceso el objetivo es crear una elipse por error cuadrático medio de los puntos del contorno de la cabeza y sacar el ángulo roll mediante la inclinación de ésta. Para ello primero hay que hacer un preprocesado para coger los contornos que nos interesan.

En este preprocesado primero se hace una umbralización al 99.9% de forma que separamos el fondo de las formas tal y como se ve en la figura 3.2.3. Después, para separar la cabeza del resto del cuerpo, se procede a un cierre con un objeto estructurante circular cuyo diámetro para el primer frame es 30 pixeles pero que después se adapta según lo alejado que esté el individuo mediante una proporción del semieje menor de la elipse del frame anterior (66% del semieje menor es con el que mejores resultados se han obtenido) tal y como se puede observar en la figura 3.2.4. Bajo la premisa de que el cuerpo se encuentra pegado al borde, se hace una limpieza de éstos mediante 4-conectividad de forma que queda sólo la cabeza. Esta premisa no es del todo cierta, ya que la imagen de profundidad tiene una sombra de 5 píxeles en todo el contorno de la imagen, que en el preprocesado de la imagen se ha pasado a blanco, de forma que no se distingue del fondo. Para conectar el cuerpo al borde, simplemente hay que volver a crear el borde inferior de color negro justo antes de la limpieza de bordes, tal y como se muestra en la figura 3.2.5. Después se sacan los bordes mediante un filtro LoG

(Laplaciano del Gaussiano) de máscara cuadrada de 3 pixeles y desviación típica de 0.1 (Figura 3.2.5) y se calcula la elipse que mejor se adapta a estos puntos del contorno por error cuadrático medio (Figura 3.2.6). Para terminar se mide el ángulo del semieje menor con la horizontal, el cual corresponde con el ángulo roll.

Figura 3.2.3: Imagen de la cara después de la umbralización al 99,9%

Figura 3.2.4: Imagen de la cara después del cierre con elemento estructurante circular

Figura 3.2.5: Imagen de la cara después de poner la sombra del borde

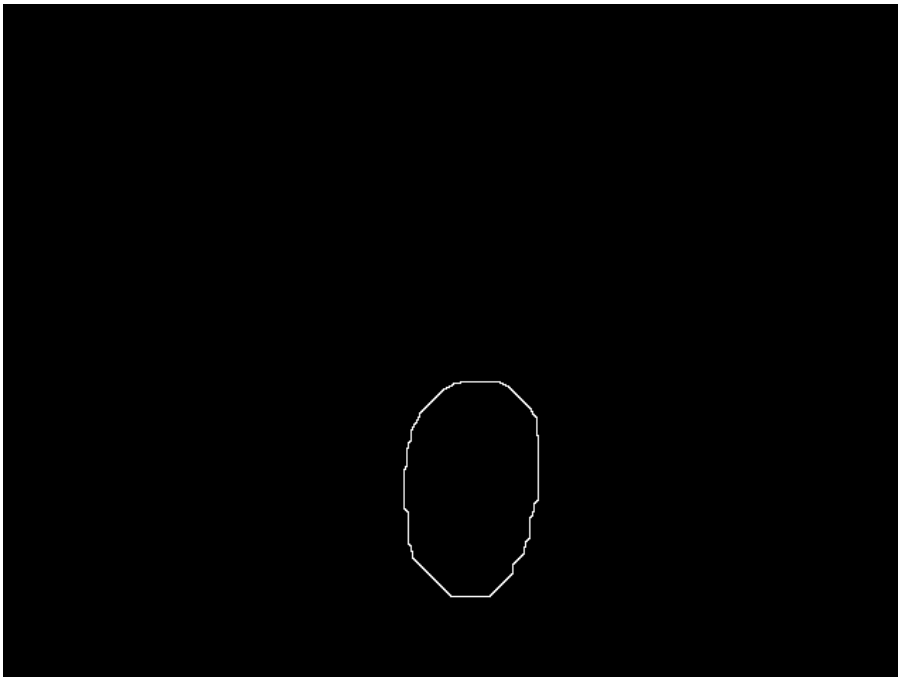


Figura 3.2.5: Imagen de los bordes de la cara con un filtro LoG

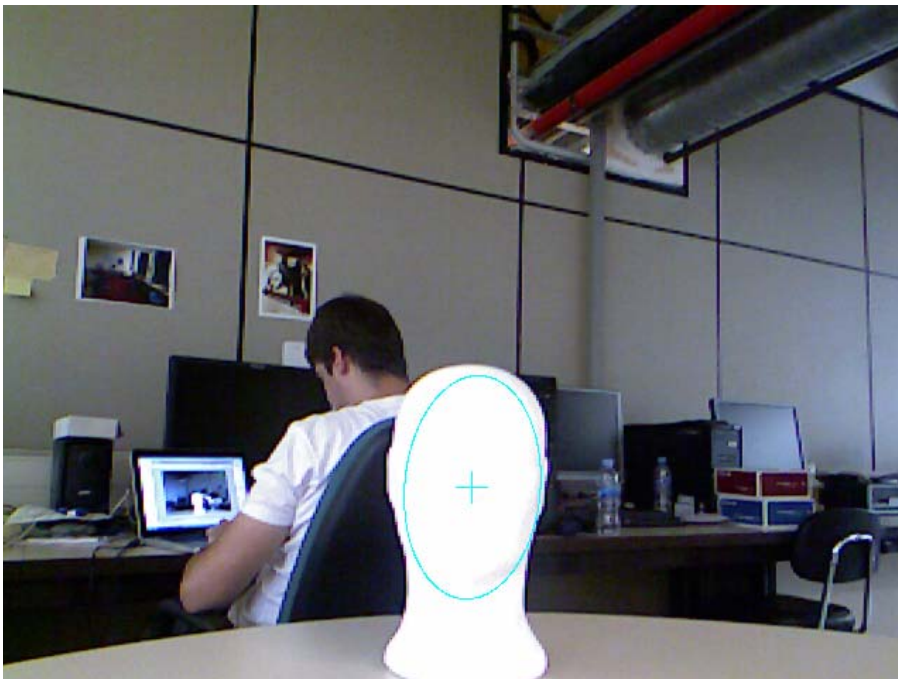


Figura 3.2.6: Imagen de la elipse de la cara

Para calcular los ángulos pitch y yaw se empieza buscando la nariz mediante el punto más cercano. Como en el algoritmo de Yu Tu, Chih-Lin Zeng y Che-Hua Yeh [1], en el primer frame se supone que la nariz es el punto más cercano, pero en las siguientes se hace una búsqueda en torno a la posición de la nariz del frame anterior (posición de referencia). Para ello se calcula un área de búsqueda que es proporcional a los semiejes de la elipse calculados en el frame anterior (40% del semieje mayor de alto por 66% del semieje menor de ancho han sido los valores con los que mejor funcionaba el algoritmo) para que se adapte según la distancia a la que se encuentre el individuo (Figura 3.2.7). El objetivo es encontrar el punto más cercano dentro de esa área, ya que aunque la nariz no es el punto más cercano de toda la imagen sí que sobresale dentro de ese rango.

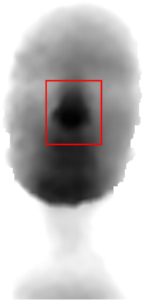


Figura 3.2.7: Imagen con el área de búsqueda para la nariz

Con el objetivo de que en el próximo frame la posición de referencia no sea inservible por movimientos de la cabeza en cualquiera de los tres ejes del espacio (hacerlo invariante a traslación), se pone como origen de coordenadas de este punto el punto central de la elipse (ya que este se mueve con la cabeza y permanece siempre a la misma altura) mediante una simple resta de ambos puntos, y se divide las coordenadas de este punto entre la mitad del semieje mayor y menor. Si las coordenadas del punto de referencia son (X, Y), el centro de la elipse tiene como coordenadas (A, B) y se quiere calcular las posiciones invariantes a traslación (X', Y') las ecuaciones que se deben utilizar son las siguientes.

$$X' = \frac{X - A}{semieje_{mayor}/2}$$

$$Y' = \frac{Y - B}{semieje_{menor}/2}$$

De esta forma en el siguiente frame, una vez calculado el punto central de la elipse, el punto de referencia se recalcula para volver a tener su origen de coordenadas en el origen de coordenadas de la imagen multiplicando por el semieje correspondiente y sumando las coordenadas del punto central de la elipse, con lo que nos queda el lugar en el que se encontraba la nariz dentro de la cara aún cuando ésta se ha movido. A partir de ese punto se establecerá la región de búsqueda de la nariz.

Una vez se tiene la nariz encontrada, se creará una nube de puntos pertenecientes a la cara y de alrededor de la nariz, con la que se creará un plano por error cuadrático medio. Para calcular esta nube de puntos lo primero es buscar el área en la cual se creará. Para ello se vuelve a coger una proporción de los semiejes mayor y menor para que sea invariante a la distancia del individuo. Además la proporción del semieje menor depende del ángulo del frame anterior, porque si la cara está de frente conviene coger más área para que el resultado sea más exacto. Sin embargo si está girada conviene coger un área menor ya que como la cara es redondeada podría calcularse un plano erróneo formado por los puntos de uno de los lados de la cara. Los valores escogidos han sido 86% del semieje mayor, 90% del semieje menor para yaw anterior menor a 20º y 80% del semieje menor para ángulo mayor a 20º, como se ve en la figura 3.2.8.

Figura 3.2.8: Imagen con el área en la que se creará la nube de puntos para yaw menor a 20º

Al área en la que se creará la nube de puntos puede parecer grande, sobre todo si se compara con el área que se ha escogido en la implementación del algoritmo 3.1 (21x21 píxeles). La razón es que en el artículo de Yu Tu y otros [1], se hablaba de puntos alrededor de la nariz, por lo que se decidió coger un área pequeña, sin embargo se ha comprobado que un área mayor hace que disminuya la variabilidad.

A partir de aquí se calculan una cantidad de puntos, cuya cantidad dependerá de la proximidad del individuo (máximo 700), uniformemente distribuidos y aleatorios con cuyas coordenadas cartesianas, calculadas al principio del algoritmo en la fase de preprocesado de la imagen de profundidad, se creará el plano por error cuadrático medio. Una vez obtenido el plano hay que calcular los ángulos que forma el vector perpendicular del plano (A, B, C) con los planos xz e yz. Para ello lo mejor es normalizar las coordenadas de forma que C sea igual a -1 (-A/C, -B/C, -1) y calcular a

continuación los ángulos mediante las ecuaciones descritas en el algoritmo de Yu Tu y otros [1] (apartado 3.1)

Una vez obtenidos los tres ángulos se hace un filtro temporal de media con el frame anterior para que la transición sea más suave y se representa la imagen RGB junto con la elipse y el punto que ha detectado como la nariz y los ángulos obtenidos.

En las ocasiones en las que no crea la elipse por no detectar cara, o no puede crear un plano por insuficiencia de puntos, los ángulos no calculados serán NaN y la nariz que se tomará como referencia en el próximo frame será el de la última posición conocida.

El diagrama de todo el proceso seguido se puede ver en la figura 3.2.9

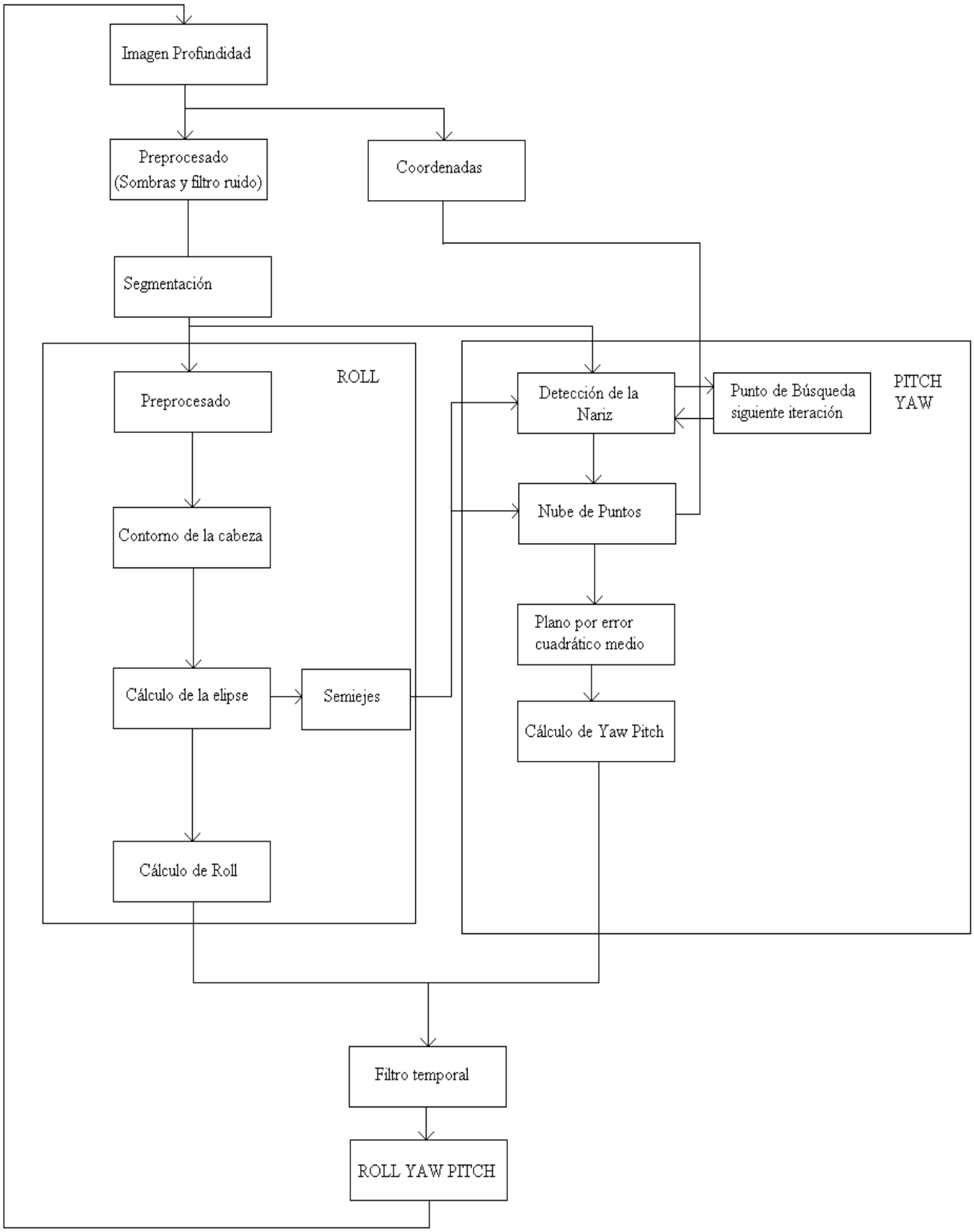


Figura 3.2.9: Diagrama del algoritmo propuesto

3.3 Explicación de los algoritmos offline

Para poder comparar los dos algoritmos sin el problema del tiempo excesivo entre frame y frame se ha procedido a hacerlo de forma offline. Para ello se ha tenido que hacer una función que grabase los frames de profundidad y RGB a 15 frames por segundo y los metiese en una variable.

La función es muy sencilla, simplemente inicializa la Kinect y a continuación empieza a pedir al servidor de OpenNi los frames cada 0,7 s durante el tiempo especificado. Mientras, va creando dos variables llamadas *depth_video* y *rgb_video* en los que va metiendo esos frames a la vez que se muestran en dos figuras. Para que el flujo de datos sea estable se ha utilizado la función pause con el tiempo que le queda a cada frame para terminar.

Para la implementación offline, se ha conservado la misma estructura en cada uno de los algoritmos, pero se han cambiado algunos valores para mejorarlos y adaptarlos al sistema offline. El objetivo de la implementación offline es evitar el tiempo que se consume entre frame y frame en la versión online al realizar la representación. Si el objetivo final del algoritmo va a ser simplemente registrar posiciones lo que se concluya para el algoritmo offline será igual de válido que para el online. Sin embargo, si el objetivo de la aplicación va a ser la representación del algún tipo de avatar u objeto este afectaría al tiempo de ejecución inter frames y a las suposiciones que se han realizado.

Debido a que el tiempo entre frame y frame offline es menor que en online, se ha cambiado el número de frames para el filtro temporal de dos a tres para que las transiciones sean aún más suaves. Además el área de búsqueda de la nariz en el algoritmo propuesto es menor, ya que al ser el tiempo menor la nariz no ha podido moverse tanto. Los valores son 25% del semieje menor y 18% del semieje mayor.

Para la visualización de los resultados, las funciones generan videos llamados “video.avi” y “video_rotacion.avi”.

4.- EVALUACIÓN DE LOS ALGORITMOS

4.1 Proceso seguido en el laboratorio

Al no disponer de un "ground truth" no podemos medir la exactitud del algoritmo propuesto. Con el fin de comprobar su robusted se han realizado una serie de pruebas en el laboratorio tanto de forma online como offline. Para ello se ha utilizado la cabeza de un maniquí de porexpan que se ha ido moviendo según convenía. Estas pruebas han sido las siguientes:

- Comprobar el rango en profundidad en el que el algoritmo puede funcionar. Para ello se ha puesto la cabeza de maniquí pegada a la Kinect, y se ha ido alejando poco a poco hasta que el algoritmo la encontraba. Entonces se registraba la distancia a la que se encontraba de la Kinect y se seguía alejando hasta que la perdía, para volver a registrarla.
- Comprobar la repetibilidad del resultado entre varias medidas para una misma posición de la cabeza con el fin de poder conocer la resolución máxima a la que puede funcionar. Para ello se ha colocado la cabeza a tres distancias distintas y por cada distancia en dos posiciones distintas, una de frente (ángulo yaw 0º) y otra girada (ángulo yaw de 30º). Sin mover la cabeza se han registrado los distintos datos para después compararlos y ver la variabilidad entre éstos. Este proceso se ha realizado primero estando tanto la cabeza como la Kinect apoyadas en una mesa y después estando ambas elevadas de forma que no pudiese interferir la mesa y la Kinect quedase a distinta altura respecto de la cabeza.
- Comprobar los ángulos máximos de rotación que puede registrar el algoritmo para comprobar el rango en el que funciona. Para ello se ha colocado la cabeza a tres distancias distintas y en cada posición se ha ido girando la cabeza en ambos sentidos hasta que el algoritmo daba datos erróneos o dejaba de encontrar la cabeza. Entonces se guardaban los datos de los ángulos máximos obtenidos. Este proceso se ha realizado también tanto con la Kinect y la cabeza sobre la mesa como ambas elevadas.
- Se ha calculado el tiempo de ejecución de cada frame para comprobar si se cumplía la premisa de que entre frame y frame el ángulo de giro no había cambiado en gran medida.

Para ello se ha utilizado una cabeza de maniquí de porexpan, que puede verse en la figura 4.1.1, un sistema Kinect y un ordenador con las especificaciones que se muestran en el apartado 6.1.

Figura 4.4.1: Cabeza maniquí de porexpan

4.2 Resultados obtenidos online

En el algoritmo de Yu Tu, Chih-Lin Zeng y Che-Hua Yeh, el alto tiempo de ejecución y sobre todo la frecuencia con la que se pierde la nariz, ha hecho inviable que se realice el estudio de robustez. Por ello sólo se ha realizado el estudio del algoritmo propuesto.

Observaciones generales:

- Con luz alta, sobretodo solar no es posible capturar imágenes buenas en profundidad, tienen mucho ruido e incluso falta de información.
- Existe un ruido notable en las imágenes de profundidad que se percibe al comparar distintos frames de la misma escena.
- El tiempo de ejecución medio es de 0.4s aunque oscila entre 0.38 y 0.42s. La representación tarda alrededor de 0.15s, el preprocesado 0.12 s, el cálculo del ángulo roll 0.12 s y la búsqueda de la nariz y cálculo de los ángulos yaw y pitch ambos 0.01 s.
- La nariz se pierde con movimientos bruscos (más de 15º entre frame y frame)
- El rango de funcionamiento va desde los 50 cm hasta los 130 cm, aunque detecta la cabeza hasta los 170 cm.

A continuación se muestran los resultados obtenidos para cada una de las tres distancias y cada una de las dos posiciones de la cabeza.

Cabeza y Kinect sobre la mesa. A 55cm			
Angulo	Variabilidad	Máximo	Mínimo
ROLL	2º	45º	-45º
PITCH	2º	20º	-20º
YAW (0º)	9º	40º	-40º
YAW (30º)	6º		

Tabla 4.2.1: Resolución y máximo y mínimo para cada ángulo con la cabeza y la Kinect sobre la mesa a una distancia de 55 cm de la Kinect

Observaciones: La Kinect coge como punto más cercano la barbilla en vez de la nariz en el primer frame. La elipse se pierde fácilmente con ángulos mayores de 20º de pitch en ambos sentidos. La elipse coge parte del cuello.

Cabeza y Kinect sobre la mesa. A 80cm			
Angulo	Variabilidad	Máximo	Mínimo
ROLL	1º	45º	-45º
PITCH	1º	30º	-30º
YAW (0º)	6º	50º	-50º
YAW (30º)	5º		

Tabla 4.2.2: Resolución y máximo y mínimo para cada ángulo con la cabeza y la Kinect sobre la mesa a una distancia de 80 cm de la Kinect

Observaciones: A veces el punto de la nariz se desvía hacia la barbilla.

Cabeza y Kinect sobre la mesa. A 130cm			
Angulo	Variabilidad	Máximo	Mínimo
ROLL	1º	45º	-45º
PITCH	1º	20º	-20º
YAW (0º)	6º	40º	-40º
YAW (30º)	5º		

Tabla 4.2.3: Resolución y máximo y mínimo para cada ángulo con la cabeza y la Kinect sobre la mesa a una distancia de 130 cm de la Kinect

Observaciones: El punto de la nariz se desvía hacia la barbilla con movimientos rápidos en el ángulo pitch en el sentido positivo (más de 10º entre frame y frame)

Cabeza y Kinect elevadas. A 55cm			
Angulo	Variabilidad	Máximo	Mínimo
ROLL	2º	45º	-45º
PITCH	2º	20º	-20º
YAW (0º)	9º	40º	-40º
YAW (30º)	6º		

Tabla 4.2.4: Resolución y ángulos máximo y mínimo para cada ángulo con la cabeza y la Kinect elevadas a una distancia de 55 cm de la Kinect

Observaciones: La elipse se pierde fácilmente con ángulos mayores de 20º de pitch en ambos sentidos.

Cabeza y Kinect elevadas. A 80cm			
Angulo	Variabilidad	Máximo	Mínimo
ROLL	1º	45º	-45º
PITCH	1º	30º	-30º
YAW (0º)	6º	50º	-50º
YAW (30º)	5º		

Tabla 4.2.5: Resolución y ángulos máximo y mínimo para cada ángulo con la cabeza y la Kinect elevadas a una distancia de 80 cm de la Kinect

Observaciones: Funciona correctamente.

Cabeza y Kinect elevadas. A 130cm			
Angulo	Variabilidad	Máximo	Mínimo
ROLL	1º	45º	-45º
PITCH	1º	20º	-20º
YAW (0º)	6º	40º	-40º
YAW (30º)	5º		

Tabla 4.2.6: Resolución y ángulos máximo y mínimo para cada ángulo con la cabeza y la Kinect elevadas a una distancia de 130 cm de la Kinect

Observaciones: Funciona correctamente.

4.3 Análisis de los resultados obtenidos online

El rango de profundidad en el que funciona el algoritmo no es muy amplio (80 cm), sobre todo si se compara con el rango de funcionamiento de la Kinect (3.5m con resolución de 1 cm), aunque es aceptable si se desea poner la Kinect en un puesto de trabajo (encima o debajo de una pantalla de ordenador) y con la persona sentada.

Con luz fuerte los mapas de profundidad tienen mucho ruido y falta de información debido a que la cámara infrarroja no puede captar los puntos porque son enmascarados por el resto de luz infrarroja que emite el sol o las lámparas. Este fenómeno, que puede observarse en la figura 4.3.1, puede ocasionar serios problemas a la hora de encontrar la elipse correctamente y puede afectar también, aunque en menor medida la búsqueda de la posición de la nariz.

Figura 4.3.1: Ejemplo de la cara segmentada con una luz solar fuerte desde un lado

En cuanto a los datos obtenidos se puede observar que no existen diferencias de rango de funcionamiento ni repitibilidad entre las dos posiciones. El mejor punto de funcionamiento es en torno a los 80 cm ya que los ángulos que puede llegar a detectar son mayores en esta posición, siendo ±45º para el roll, ±30º para el pitch y ±50º para el yaw y la variabilidad entre medidas es la misma que a 130 cm (ver Figuras 4.3.2 y 4.3.3). En el documento de Yu Tu y otros [1] se habla de que ellos consiguieron un rango de ±45º en los tres ángulos, por lo que se ha conseguido 10º más en el yaw, aunque en el pitch se ha obtenido 20º menos.

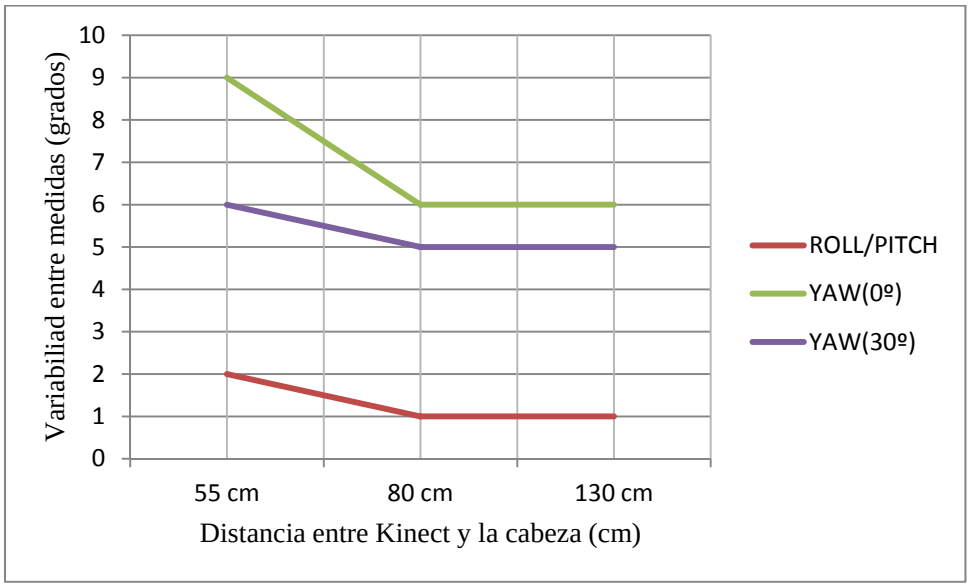


Figura 4.3.2: Gráfica de la variabilidad entre medidas

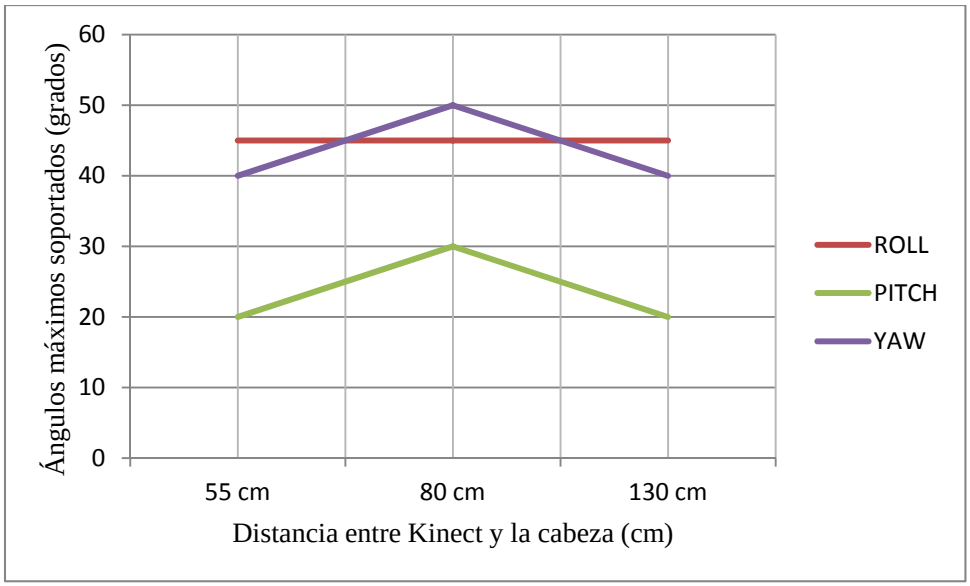


Figura 4.3.3: Gráfica de los ángulos máximos soportados

La resolución en los ángulos de roll y pitch es bastante buena, menor a 5% del rango de funcionamiento, mientras que el yaw puede llegar hasta el 12%. Esto es debido a que la cara es más redondeada en horizontal que en vertical, por lo que es más difícil ajustar el plano correctamente para el ángulo yaw.

Cuando la cabeza se encuentra muy cerca (55 cm), con ángulos grandes de pitch (mayores a 20º), tal y como puede verse en la figura 4.3.2, parte de la cabeza se introduce fuera de la zona de visión de la Kinect, es decir entre los 0 y 50 cm, por lo que no se puede detectar elipse o ésta no es correcta. Cuando la cabeza está muy alejada, más de 130 cm y se produce un giro la nariz se pierde como puede observarse en la figura 4.3.3. Esto es porque el área de búsqueda de la nariz es demasiado grande y el mínimo (punto más cercano) en esa área ya no es la nariz. Sin embargo no importa ya que a esas distancias la resolución es tan pequeña que no se puede construir una nube de puntos que de un plano fiable del que calcular los ángulos pitch y yaw.

Figura 4.3.2: Ejemplo de pérdida de información debido a un ángulo pitch muy grande estando la cabeza a 55cm

Figura 4.3.3: Ejemplo de pérdida de la nariz por encontrarse la cabeza más alejada de 130cm

Aunque en los datos obtenidos no se observa diferencia entre las dos posiciones, se ha observado que la nariz se pierde con más frecuencia cuando tanto la Kinect y la cabeza están apoyadas en la mesa. (Figura 4.3.4). Se nota especialmente cuando la cabeza se encuentra cerca de la Kinect (55 cm). Esto es debido a que el punto de la nariz se desvía hacia la barbilla porque la Kinect se encuentra por debajo de la cara y la barbilla es el punto más cercano. Por eso en la segunda posición, en la cual la Kinect se encuentra por encima de los ojos (Figura 4.3.5), se pierde menos. Una vez que el punto de la nariz se

ha trasladado a la barbilla, ya no vuelve a la nariz salvo que se haga un movimiento brusco en el ángulo pitch en sentido negativo.

Además, en esta posición en la que tanto la Kinect se encuentra por debajo de la cara, si el cuello es muy ancho (del orden del de la cabeza), es probable que coja parte de éste dentro de la elipse (Figura 4.3.4), sobre todo si la cabeza se encuentra cerca (55 cm) ya que después de la segmentación de la cabeza, éste apenas desaparece, porque se encuentra muy cerca de la Kinect en relación con el punto más cercano de la cara y no consigue quitarlo con el cierre por su excesiva anchura.

Figura 4.3.4: Ejemplo de mala detección de la nariz y elipse estando la Kinect por debajo de los ojos (Kinect y cabeza encima de la mesa)

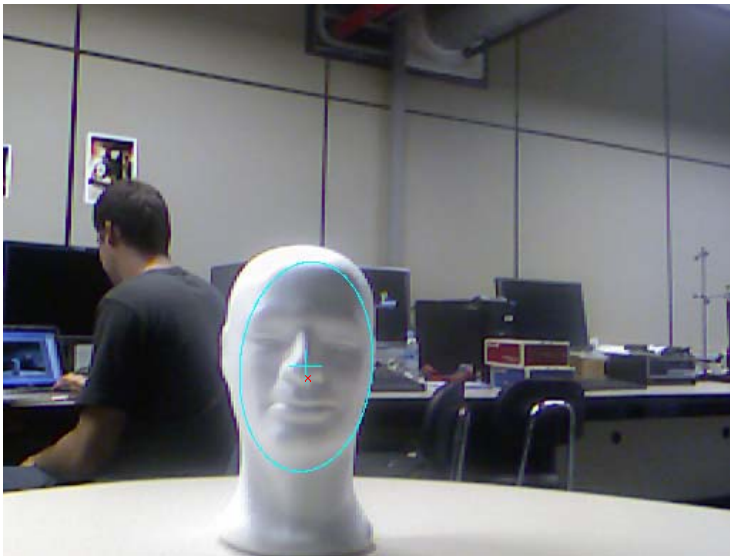


Figura 4.3.5: Ejemplo de buena detección de la nariz y elipse estando la Kinect por encima de los ojos (Kinect elevada y cabeza en la mesa)

El tiempo de ejecución es un poco alto (0.4 s) por lo que la nariz se pierde si se hacen movimientos bruscos (más de 15º entre frame y frame) tal y como se observa en la figura 4.3.1, ya que la nariz ya no se encuentra dentro del área de búsqueda y la premisa de que se encuentra cerca de la posición del frame anterior no es cierta. Teniendo en cuenta que la representación es lo que más tarda (alrededor de 0.15 s), podría mejorarse en este asunto si se hace esta más sencilla de forma que sea más rápida su ejecución. En cuanto al tiempo que tarda en buscar la nariz cabe decir que es realmente rápido, en torno a 0.01 s, lo que hace que el algoritmo se agilice considerablemente.

Figura 4.3.1: Ejemplo de movimiento brusco en el ángulo yaw

4.4 Resultados obtenidos offline

El estudio no se ha podido realizar para el algoritmo de Yu Tu, Chih-Lin Zeng y Che-Hua Yeh [1], ya que aunque en offline se soluciona el problema del tiempo de ejecución, la nariz se sigue perdiendo con mucha frecuencia debido a la mala corrección de la rotación de la cabeza.

En el estudio offline del algoritmo propuesto, todas las observaciones relacionadas con pérdida de la nariz y de la elipse, los tiempos de ejecución y el ruido se siguen manteniendo, puesto que el algoritmo en sí no ha cambiado, sólo algunos valores.

A continuación se muestran los resultados obtenidos para cada una de las tres distancias y cada una de las dos posiciones de la cabeza.

Cabeza y Kinect sobre la mesa. A 55cm			
Angulo	Variabilidad	Máximo	Mínimo
ROLL	1º	45º	-45º
PITCH	2º	20º	-20º
YAW (0º)	6º	40º	-40º
YAW (30º)	4º		

Tabla 4.4.1: Resolución y máximo y mínimo para cada ángulo con la cabeza y la Kinect sobre la mesa a una distancia de 55 cm de la Kinect

Cabeza y Kinect sobre la mesa. A 80cm			
Angulo	Variabilidad	Máximo	Mínimo
ROLL	1º	45º	-45º
PITCH	1º	30º	-30º
YAW (0º)	4º	50º	-50º
YAW (30º)	3º		

Tabla 4.4.2: Resolución y máximo y mínimo para cada ángulo con la cabeza y la Kinect sobre la mesa a una distancia de 80 cm de la Kinect

Cabeza y Kinect sobre la mesa. A 130cm			
Angulo	Variabilidad	Máximo	Mínimo
ROLL	1º	45º	-45º
PITCH	1º	20º	-20º
YAW (0º)	4º	40º	-40º
YAW (30º)	4º		

Tabla 4.4.3: Resolución y máximo y mínimo para cada ángulo con la cabeza y la Kinect sobre la mesa a una distancia de 130 cm de la Kinect

Cabeza y Kinect elevadas. A 55cm			
Angulo	Variabilidad	Máximo	Mínimo
ROLL	1º	45º	-45º
PITCH	1º	20º	-20º
YAW (0º)	6º	40º	-40º
YAW (30º)	4º		

Tabla 4.4.4: Resolución y ángulos máximo y mínimo para cada ángulo con la cabeza y la Kinect elevadas a una distancia de 55 cm de la Kinect

Cabeza y Kinect elevadas. A 80cm			
Angulo	Variabilidad	Máximo	Mínimo
ROLL	1º	45º	-45º
PITCH	1º	30º	-30º
YAW (0º)	4º	50º	-50º
YAW (30º)	3º		

Tabla 4.4.5: Resolución y ángulos máximo y mínimo para cada ángulo con la cabeza y la Kinect elevadas a una distancia de 80 cm de la Kinect

Cabeza y Kinect elevadas. A 130cm			
Angulo	Variabilidad	Máximo	Mínimo
ROLL	1º	45º	-45º
PITCH	1º	20º	-20º
YAW (0º)	4º	40º	-40º
YAW (30º)	4º		

Tabla 4.4.6: Resolución y ángulos máximo y mínimo para cada ángulo con la cabeza y la Kinect elevadas a una distancia de 130 cm de la Kinect

4.5 Análisis de los resultados obtenidos offline

En cuanto al algoritmo de Yu Tu, Chih-Lin Zeng y Che-Hua Yeh, la nariz se pierde muy fácilmente. Esto es debido a que la corrección de la rotación de la cara no es precisa, con lo que el punto más cercano ya no es la nariz. Este problema se acentúa cuando se gira la cabeza en el ángulo yaw, ya que es el que menor resolución tiene. En cuanto a la resolución, no se ha podido realizar el estudio por este problema por lo que no se han podido obtener datos.

Figura 4.5.1: Ejemplo de pérdida de la nariz debida a la poca precisión en la corrección de la rotación

Figura 4.5.2: Ejemplo de pérdida de la nariz al rotar en el ángulo yaw

Además la elipse no se adapta a la cara, debido a que no hay ningún sistema que elimine la parte del cuello, cosa que se ha arreglado con un cierre en el algoritmo propuesto. Este problema puede verse claramente en la figura 3.2.2.

En el algoritmo propuesto offline, los problemas de pérdida de la elipse y la nariz se mantienen igual que en el online puesto que el algoritmo es el mismo. Sin embargo la variabilidad de una misma posición es menor debido al filtrado temporal mayor de los resultados como puede verse en la gráfica de la figura 4.5.3. Se nota especialmente en el ángulo yaw para 0º, en el que la oscilación disminuye hasta tres grado debido al filtrado temporal de media de tres frames. Lo que supone una mejoría del 30%, llegando a tener un 5% menos de variación entre distintas medidas.

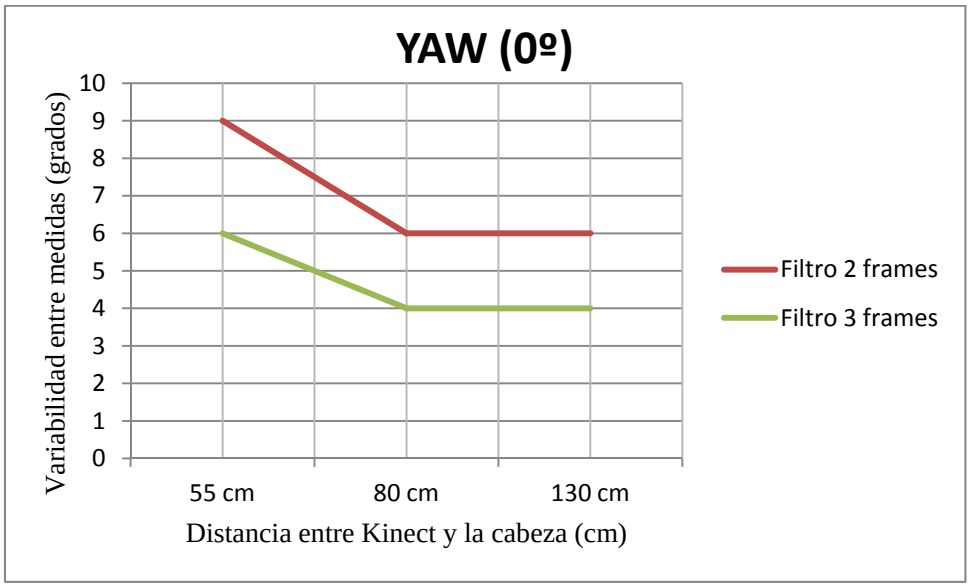


Figura 4.5.3: Gráfica comparativa de la variabilidad entre medidas para un filtro temporal de 2 frames y uno de 3 frames

En cuanto al tiempo de ejecución, que es el mayor problema que se ha tenido en este proyecto, cabe decir que la diferencia es de casi el doble. Mientras que el algoritmo de Yu Tu, Chih-Lin Zeng y Che-Hua Yeh [1] tarda 0.75 segundos por frame, el propuesto sólo tarda 0.4 s, como puede verse en la gráfica de la figura 4.5.4. Por lo que el algoritmo de Yu Tu y otros [1] es inviable para su ejecución de forma online, al menos en la implementación que se ha hecho, ya que el ángulo del frame anterior puede ser muy distinto al de la nueva y el giro de la cabeza ser tan impreciso que el punto más cercano ya no sea la nariz. Con lo que se tendría no solo el problema de la poca precisión en la estimación de los ángulos, sino que también se perdería la nariz muy fácilmente al girar la cabeza.

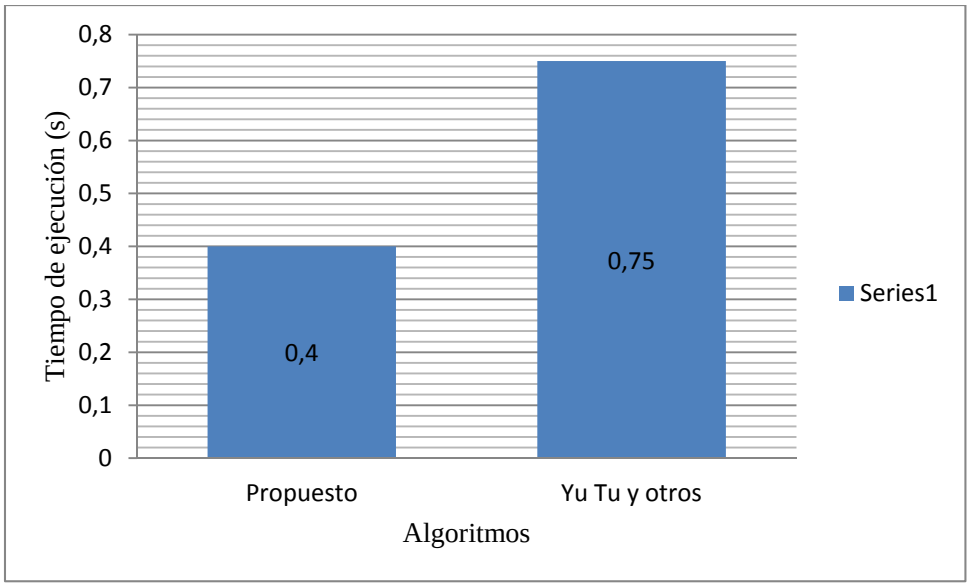


Figura 4.5.4: Gráfica con los tiempos de ejecución de los dos algoritmos

5.- CONCLUSIONES Y LÍNEAS FUTURAS

5.1 Conclusiones

Tras el análisis de Kinect se ha llegado a la conclusión de que es un sistema novedoso, y que permite simplificar muchos problemas de forma fácil, sobre todo derivados de la falta de información en tres dimensiones de las cámaras convencionales. La forma de interactuar con ella, desde el punto de vista de la programación es realmente fácil, sobre todo si se utilizan los SDK y los middlewares disponibles.

En este proyecto se ha implementado en Matlab y analizado el algoritmo de Yu Tu, Chih-Lin y Che-Hua Yeh [1] de detección de la posición de la cabeza mediante los mapas de profundidad. El algoritmo es muy poco robusto debido a la falta de exactitud en las medidas de los ángulos que provoca que no se encuentre la nariz correctamente. Aun arreglándose este problema, la nariz seguiría sin encontrarse correctamente puesto que el tiempo de ejecución es demasiado alto de forma online. Sí podría hacerse de forma offline, aunque el procesamiento sería muy costoso.

Debido al problema del tiempo de ejecución se realizó un algoritmo con mejoras para la detección de la nariz. Tras analizarlo se ha llegado a la conclusión de que no es fácil encontrar la nariz utilizando el método del punto más cercano, ya que la barbilla y la frente también sobresalen del resto de la cara. Los ángulos roll y pitch tienen una buena resolución, menor al 5%, no siendo así con el ángulo yaw que puede llegar al 12%. Esto es debido a que horizontalmente la cara no es plana, sino redondeada, lo que complica el formar un plano sin cometer mucho error. Además el ruido que presentan las imágenes de profundidad de la Kinect es un punto en contra, ligado a que la resolución tanto en horizontal como en vertical y en profundidad puede ser insuficiente a la hora de distinguir rasgos de la cara.

Para poder comparar ambos algoritmos sin que el tiempo de ejecución supusiese un problema se implementaron para vídeo. Para ello hubo que hacer una función que grabase imágenes de la Kinect a 15fps. Tras su análisis se llegó a la conclusión de que un tiempo menor entre frames permite un filtrado temporal mayor, que hace que la resolución mejore considerablemente en el ángulo yaw.

5.2 Líneas futuras

Uno de los principales problemas del proyecto ha sido el tiempo de ejecución. Matlab es un lenguaje de programación que presenta muchas ventajas para crear prototipos pero que es muy lento a la hora de la ejecución. Por ello sería apropiado utilizar un lenguaje más rápido como C o Java. Con esta mejora en el tiempo de ejecución podría hacerse un filtrado temporal mucho mayor, con lo que se mejoraría la resolución del ángulo yaw.

Además sería interesante utilizar algún otro sistema en el cual se pueda obtener una resolución mayor, como Kinect2 o Leap Motion, sistemas ya anunciados para su futura venta, para obtener mejor detalle de la cara.

En cuanto al problema del ángulo yaw, se debería probar a elegir una serie de puntos predefinidos que representen la parte frontal de la cara, en vez de aleatorios, de forma que el plano obtenido se ajuste mejor a esta parte de la cara.

6.- REQUISITOS DEL SISTEMA E INSTALACIÓN

6.1 Hardware

El hardware utilizado en este proyecto ha sido el siguiente:

- Sistema Kinect de Microsoft
- Ordenador Intel ® Core™ 2 DUO 2,25GHz, 2GB RAM

6.2 Software

Los programas y drivers utilizados en este proyecto han sido los siguientes

- Microsoft Windows XP, Service Pack 3
- Matlab R2009a
- SensorKinect
- OpenNi
- Funciones Kinect-Mex

6.3 Instalación

Como primer paso se debe instalar Matlab R2009a. No se ha probado la compatibilidad con otras versiones, aunque no se espera que haya problemas con versiones más recientes.

Para la instalación de los drivers, antes de enchufar la Kinect se debe instalar OpenNi y después SensorKinect en su versión unstable. Éstos se pueden descargar desde la página web de OpenNi [3]

Una vez instalados, se procede a enchufar el sistema a un puerto USB y a la red eléctrica. Saldrá un mensaje de nuevo hardware encontrado y habrá que especificarle que los drivers se encuentran en la ruta

*C: /ArvhivosdePrograma*s*/PrimeSense/SensorKinect*

Una vez hecho esto, sólo hay que ejecutar la función que se desee dentro de Matlab, que se debe encontrar en la misma carpeta que la carpeta de Kinect-Mex.

7.- REFERENCIAS BIBLIOGRÁFICAS

1- Yu Tu, Chih-Lin Zeng, Che-Hua Yeh, Real-time head pose estimation using depth map for avatar control, National Taiwan University

2-Aritz Legarretaetxebarria, Sistema de localización y seguimiento de personas en interiores mediante cámara PTZ basado en las tecnologías Kinect y Ubisense, Universidad del País Vasco

3-Daniel Yuste Padilla, Sistema de reconocimiento de gestos para un visor de imágenes

4-J.R. Ruiz-Sarmiento, C. Galindo, J. Gonzalez-Jimenez, J.L. Blanco, Navegación Reactiva de un Robot Móvil usando Kinect, Universidad de Málaga

5- http://www.openni.org/Downloads/OpenNIModules.aspx

6- http://es.wikipedia.org/wiki/Kinect

7- http://www.ros.org/wiki/kinect_calibration/technical#Depth_calculation

“ESTIMACIÓN DE LA POSICIÓN DE LA CABEZA EN TIEMPO REAL CON KINECT”

IGNACIO SAN ROMÁN LANA

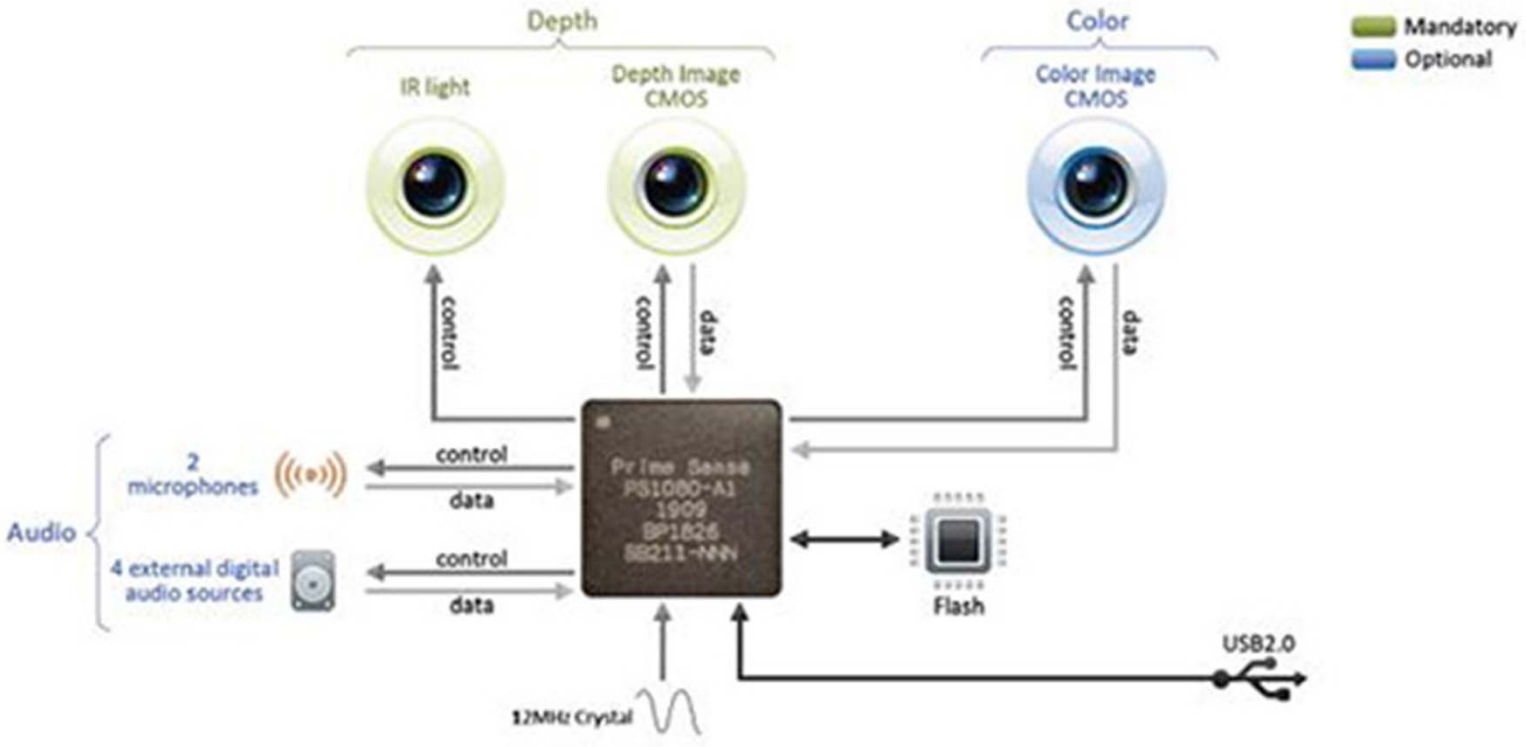
OBJETIVOS

- Estudio del sistema Kinect: Funcionamiento, drivers e instalación
- Estudio de cómo controlar el sistema Kinect desde Matlab
- Estudio e implementación del algoritmo de Yu Tu, Chih-Lin y Che-Hua Yeh
- Análisis del algoritmo implementado
- Propuesta e implementación de un algotimo basado en el de Yu Tu, Chih-Lin y Che-Hua Yeh
- Análisis del algoritmo propuesto

DESCRIPCIÓN DEL SISTEMA KINECT

- Sensor CMOS RGB (480x640 pixeles a 30 fps)
- Sensor CMOS IR monocromo (480x640 pixeles a 30 fps)
- Proyector de puntos IR (laser de clase 1)
- Micrófono de cuatro receptores (16 bits a 16 kHz)
- Chip de PrimeSense (Control y mapas de profundidad)
- Motor para la inclinación
- Acelerómetro
- Procesador (Para la ejecución del software de Microsoft)

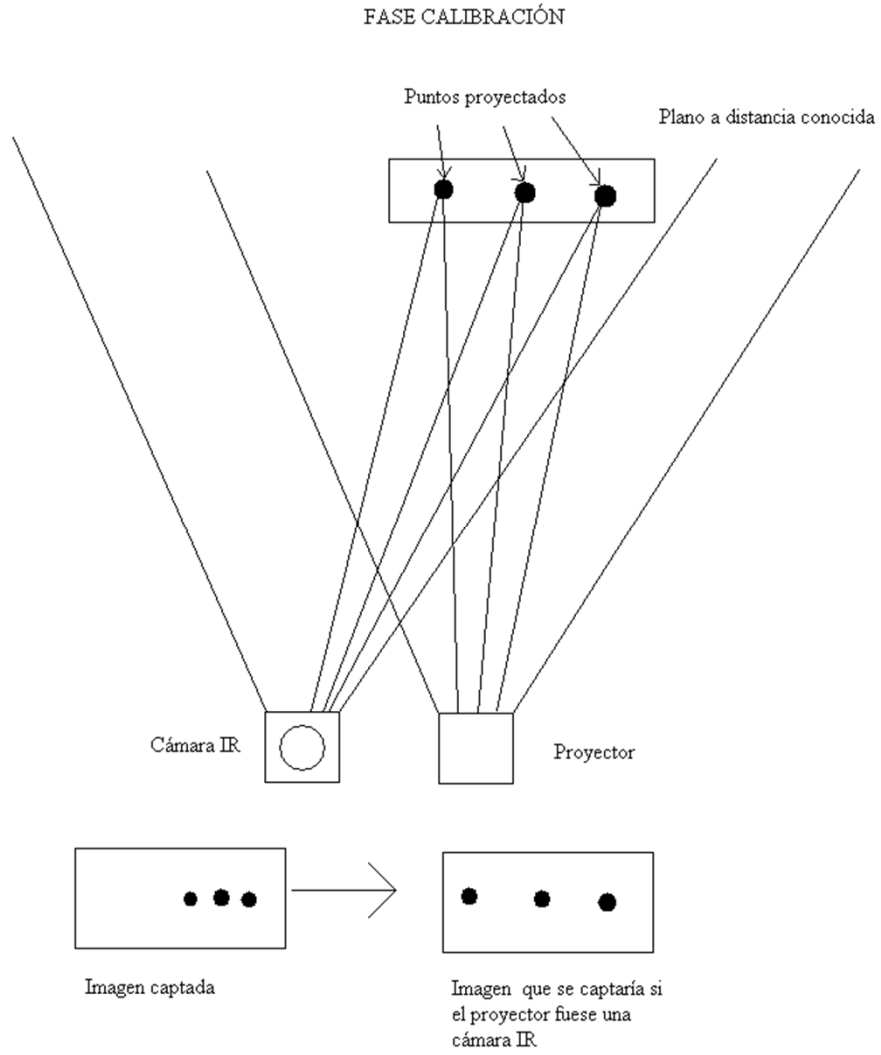
DESCRIPCIÓN DEL SISTEMA KINECT



MAPAS DE PROFUNDIAD

- Creación en dos pasos (codificación de luz)
  1. Calibración
     - Proyección del patrón de puntos sobre una superficie colocada a unas distancias conocida
     - Por cada posición se saca una imagen infrarroja
     - Por triangulación se calcula la proyección de esos puntos en el proyector como si éste fuese una cámara IR
     - Esa proyección servirá de referencia para saber dónde se proyectan los puntos

MAPAS DE PROFUNDIAD



MAPAS DE PROFUNDIAD

2. Funcionamiento

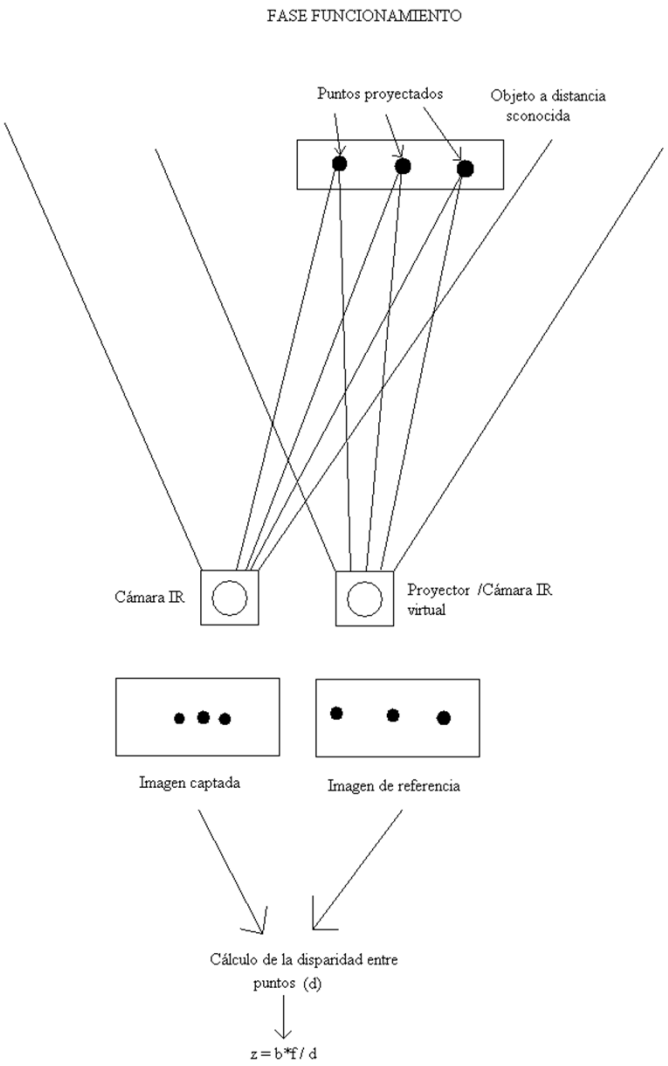
- Proyección del patrón de puntos
- El chip de PrimeSense busca los puntos en la imagen IR
- El chip compara la posición de éstos con la de los puntos proyectados mediante una matriz de correlación 9x9 $\longrightarrow$ Disparidad (doff) entre puntos
- Mediante triangulación activa se calcula la distancia de cada punto

$$z = b * f / (\frac{1}{8} * (doff - kd))$$

b=75 mm kd=1090
f=750 mm

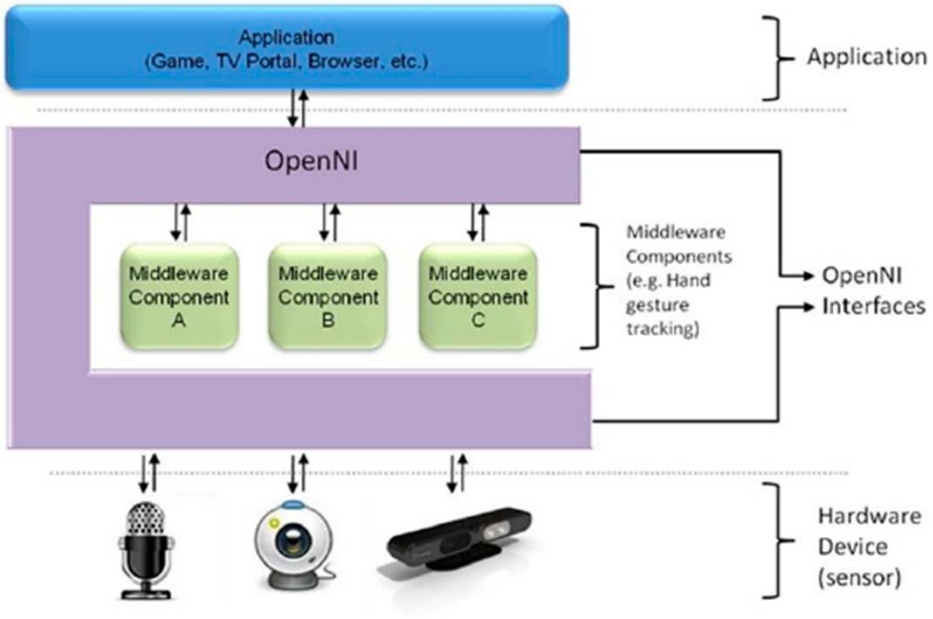
- Se crea la imagen con las distancias obtenidas

MAPAS DE PROFUNDIAD



SISTEMA OPENNI

- Tres paquetes
  - SensorKinect (drivers)
  - OpenNi (servidor y SDK)
  - Nite (Middleware) ⟶ No es necesario



CONFIGURACIÓN KINECT

- Primero hay que inicializarla → Archivo inicialización
  - info.image_node.width = 640;
  - info.image_node.height = 480;
  - info.image_node.fps = 30;
  - info.image_node.mirror = true;
  - context = mxNiCreateContext(info, 0);
- Alineamiento imágenes RGB y profundidad
  - option.adjust_view_point = true;
  - mxNiUpdateContext(context, option);
- Otras configuraciones

  OutputFormat, Gain, HoleFilter,MinDepthValue,MaxDepthValue, WhiteBalanced

  - mxNiSetProperty(context, 'OutputFormat', 1);

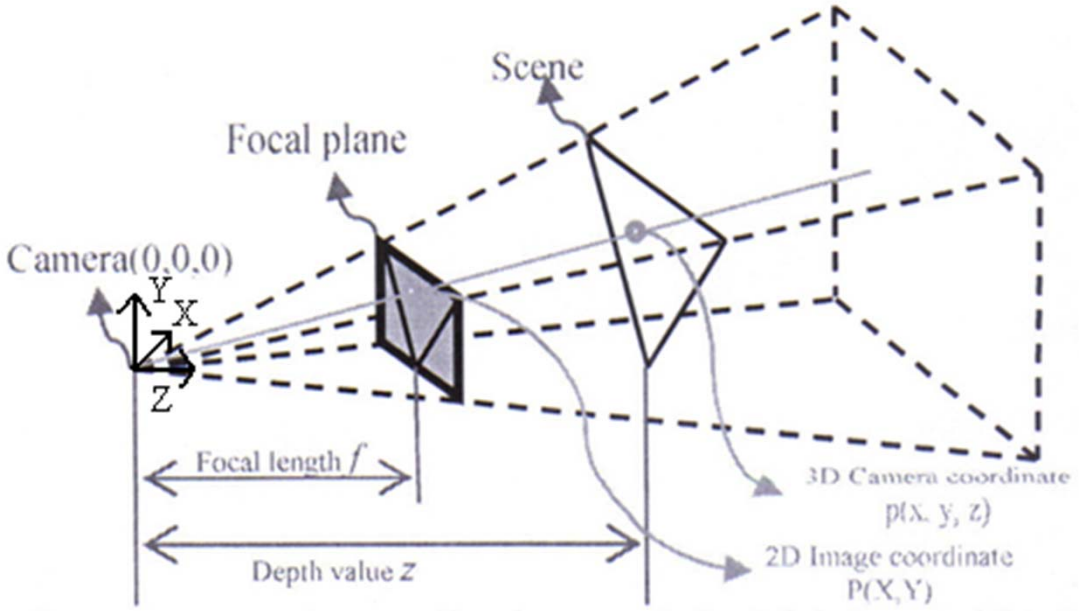
Algoritmo de Yu Tu, Chih-Lin Zeng y Che-Hua Yeh

- Adquisición de imagen

Algoritmo de Yu Tu, Chih-Lin Zeng y Che-Hua Yeh

- Conversión en coordenadas cartesianas

$$p(x,y,z) = (z\frac{X}{f}, z\frac{Y}{f}, z)$$

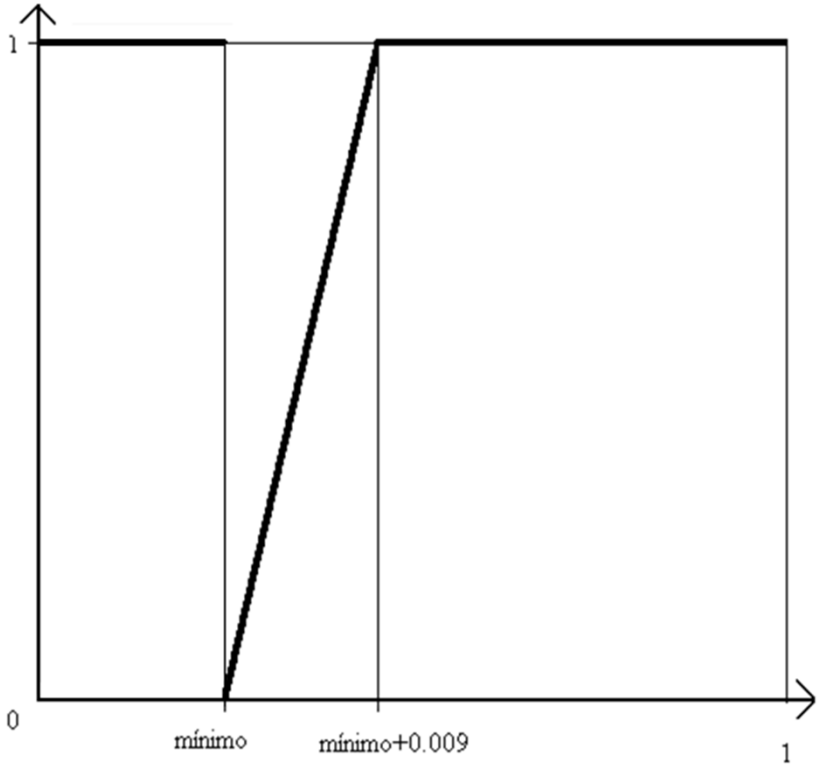


Algoritmo de Yu Tu, Chih-Lin Zeng y Che-Hua Yeh

- Pasar imagen profundidad a un rango entre 0 y 1
- Eliminación de sombras negras

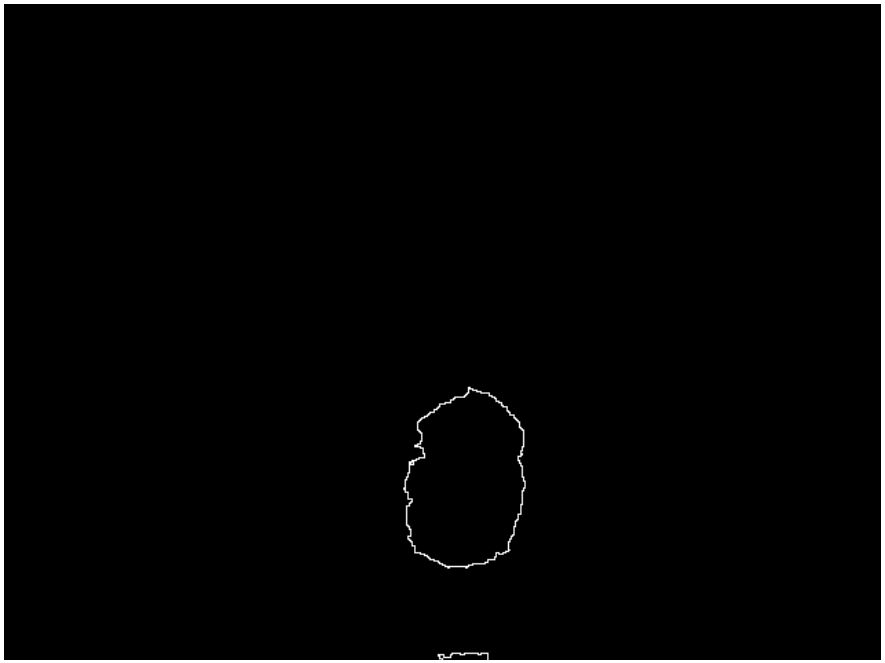
Algoritmo de Yu Tu, Chih-Lin Zeng y Che-Hua Yeh

- Segmentación (Umbralización + Stretching)



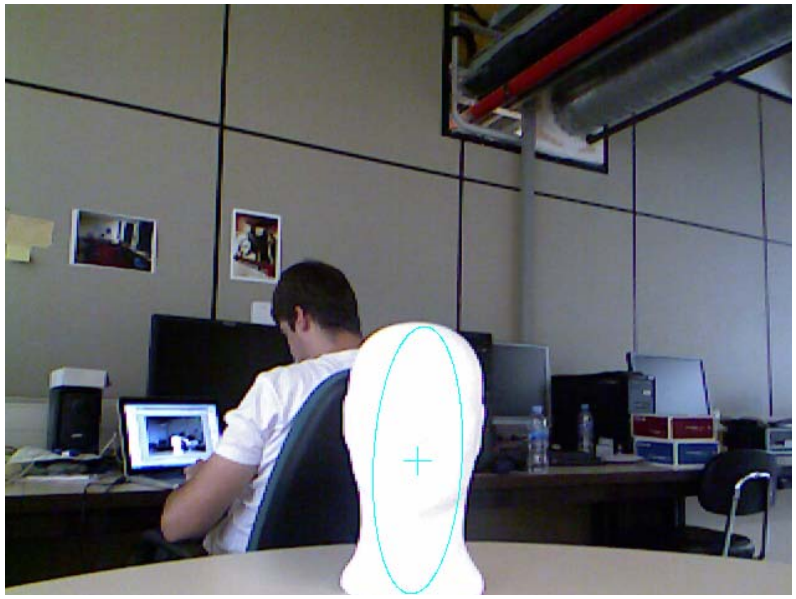
Algoritmo de Yu Tu, Chih-Lin Zeng y Che-Hua Yeh

- Cálculo del ángulo Roll
  - Contornos de la cara
    - Umbralización al 70%
    - Filtro LoG (máscara 3x3 y desviación típica 0.1)



Algoritmo de Yu Tu, Chih-Lin Zeng y Che-Hua Yeh

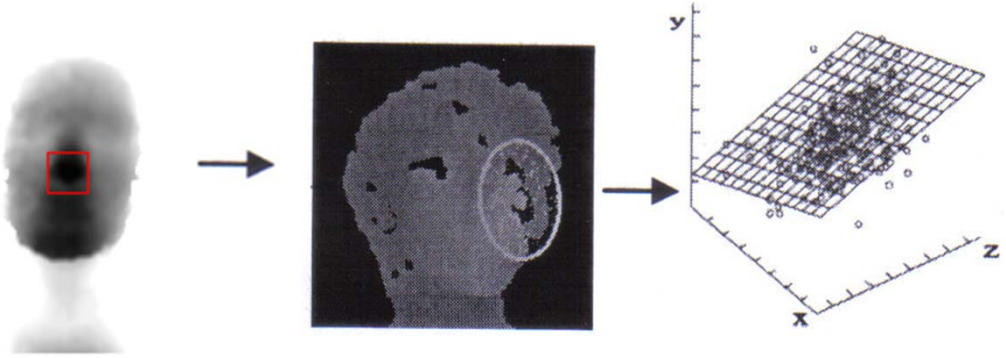
- Filtrado de los puntos
  - $(x_i, y_i) = \frac{\sum_{k=i-l}^{i+l}(x_k, y_k)}{2l+1}$ l=0
- Elipse por error cuadrático medio



- Roll=ángulo entre semieje menor y la horizontal

Algoritmo de Yu Tu, Chih-Lin Zeng y Che-Hua Yeh

- Cálculo de los ángulos Pitch y Yaw
  - Búsqueda de la nariz (punto más cercano)
  - Búsqueda del área alrededor de la nariz(21x21 píxeles)
  - Nube de 300 puntos aleatorios
  - Creación del plano por error cuadrático medio (con las coordenadas cartesianas)



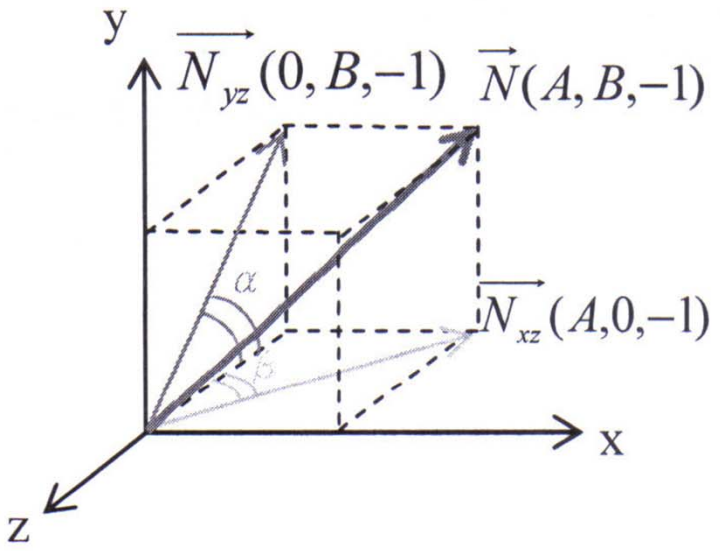
Algoritmo de Yu Tu, Chih-Lin Zeng y Che-Hua Yeh

- Normalización del vector perpendicular

$$\left(\frac{-A}{C}, \frac{-B}{C}, -1\right)$$

- Cálculo de ángulos pitch y yaw y filtrado temporal

$$Pitch = \cos^{-1}\frac{1}{\sqrt{\left(\frac{-A}{C}\right)^2 + 1^2}} \qquad Yaw = \cos^{-1}\frac{1}{\sqrt{\left(\frac{-B}{C}\right)^2 + 1^2}}$$



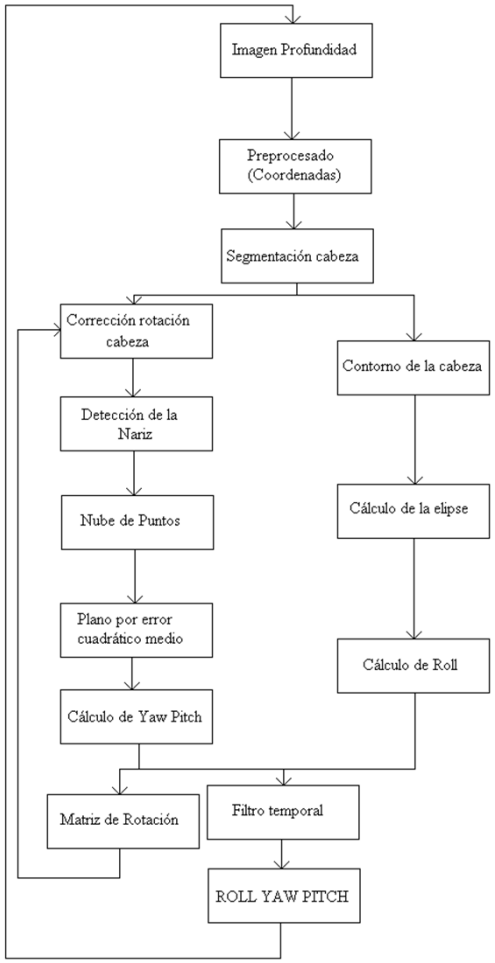
Algoritmo de Yu Tu, Chih-Lin Zeng y Che-Hua Yeh

- Cálculo de la matriz de corrección de la rotación
- En el siguiente frame se multiplica esta matriz por cada punto para poner la cara de frente y que la nariz siga siendo el punto más cercano

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos(yaw) & 0 & -sin(yaw) \\ 0 & 1 & 0 \\ sin(yaw) & 0 & cos(yaw) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(pitch) & sin(pitch) \\ 0 & -sin(pitch) & cos(pitch) \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Algoritmo de Yu Tu, Chih-Lin Zeng y Che-Hua Yeh

- Diagrama



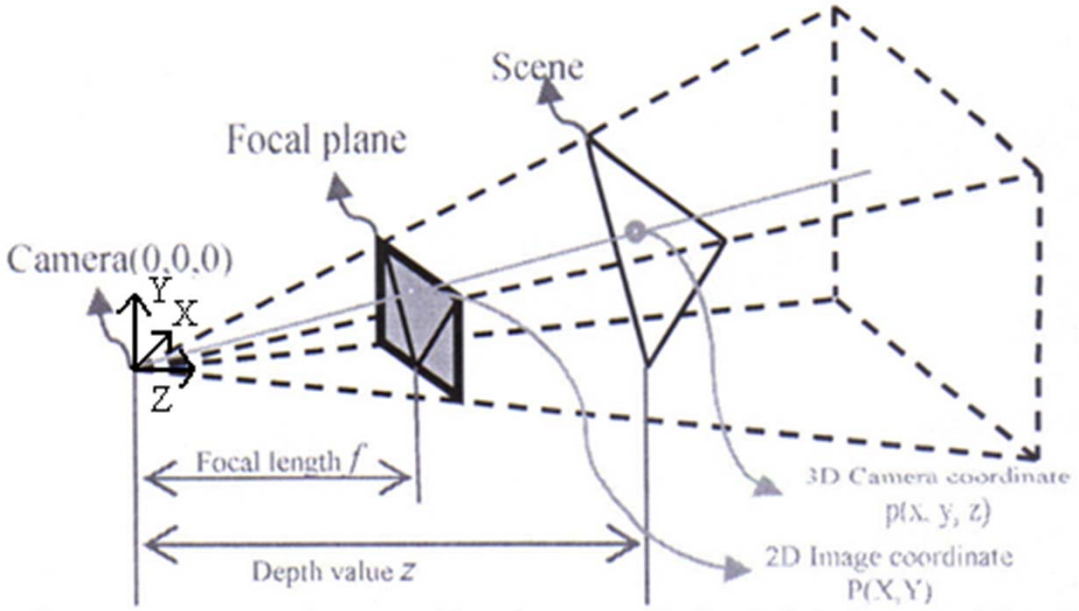
Algoritmo propuesto

- Adquisición de imagen

Algoritmo de Yu Tu, Chih-Lin Zeng y Che-Hua Yeh

- Conversión en coordenadas cartesianas

$$p(x,y,z) = (z\frac{X}{f}, z\frac{Y}{f}, z)$$

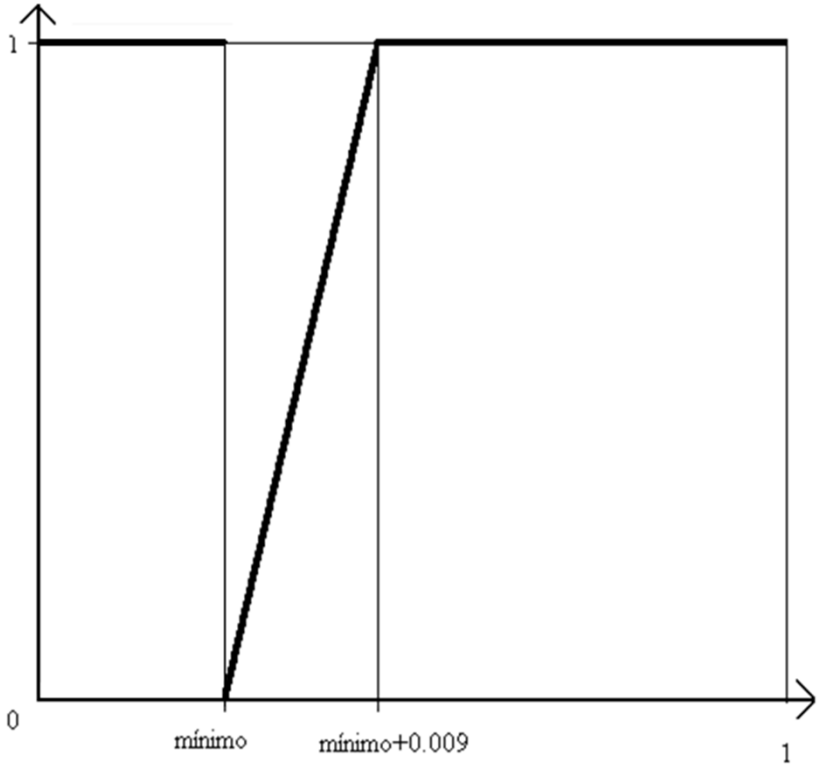


Algoritmo de Yu Tu, Chih-Lin Zeng y Che-Hua Yeh

- Pasar imagen profundidad a un rango entre 0 y 1
- Eliminación de sombras negras
- Filtro paso bajo para quitar ruido

Algoritmo propuesto

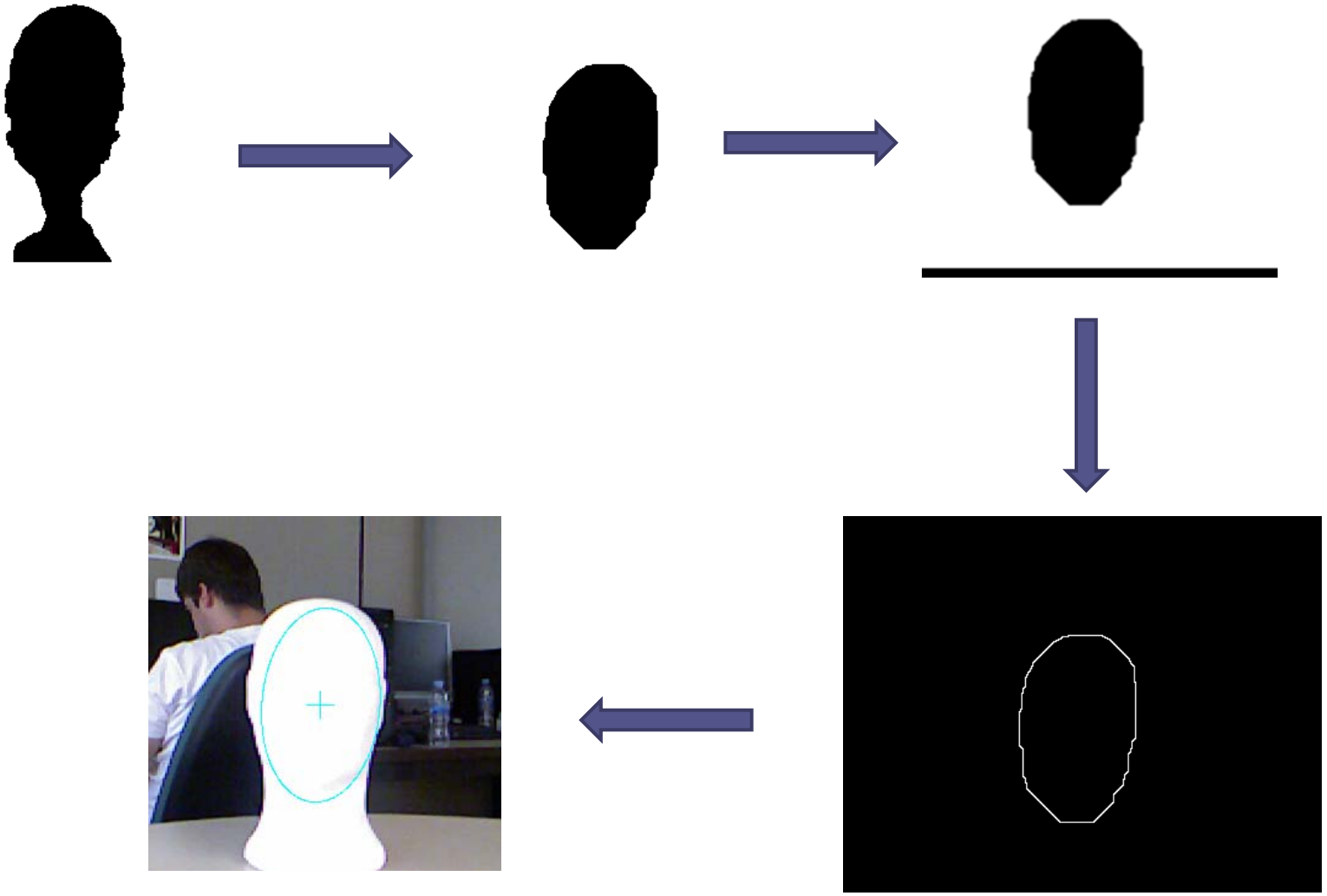
- Segmentación



Algoritmo propuesto

- Cálculo del ángulo Roll
  - Umbralización al 99.9%
  - Cierre con elemento estructurante circular (Diámetro primer frame 30 después 66% semieje menor del frame anterior)
  - Marco 5 píxeles para juntar el tronco con el borde
  - Borrado de bordes
  - Filtro LoG (máscara 3x3 píxeles desviación típica 0.1)
  - Cálculo elipse por error cuadrático medio
  - Cálculo del ángulo Roll

Algoritmo propuesto



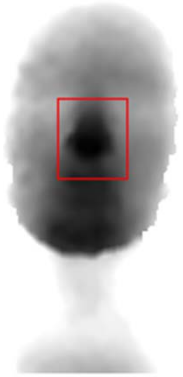
Algoritmo propuesto

- Cálculo de los ángulos Pitch y Yaw
  - Búsqueda de la nariz primer frame (punto más cercano)
  - Cálculo de la posición relativa del punto de la nariz (invariante frente a traslación)

$$X' = \frac{X - A}{semieje_{mayor}/2} \qquad Y' = \frac{Y - B}{semieje_{menor}/2}$$

Algoritmo propuesto

- Búsqueda de la nariz en los siguientes frames
  - Cálculo del punto de referencia en la nueva posición de la cara
  - Cálculo del área en torno a ese punto (40% semieje mayor X 66% semieje menor)
  - Mínimo dentro de esa área ⟶ Punto de la nariz



Algoritmo propuesto

- Búsqueda del área alrededor de la nariz
  - 86% semieje mayor
  - 90% semieje menor para Yaw anterior<20º
  - 80% semieje menor para yaw anterior>20º

- Creación de la nube de puntos
  - Máximo de 700 puntos, depende de la distancia a la que se encuentre

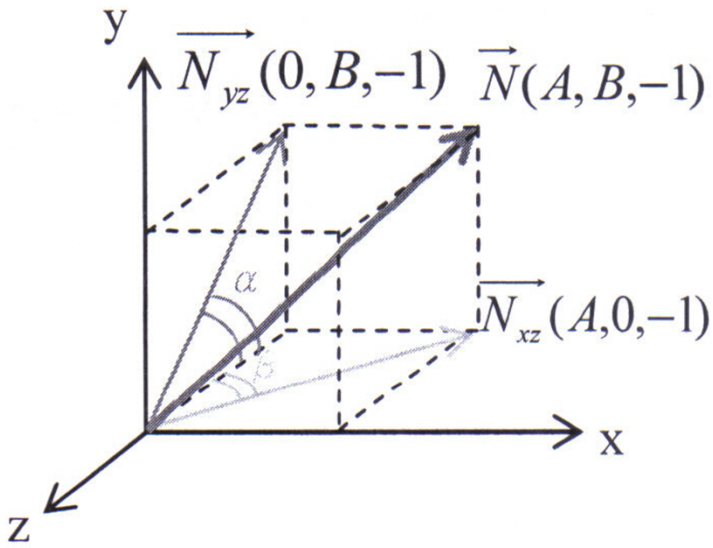
Algoritmo propuesto

- Cálculo del plano por error cuadrático medio
- Normalización del vector perpendicular

$$\left(\frac{-A}{C}, \frac{-B}{C}, -1\right)$$

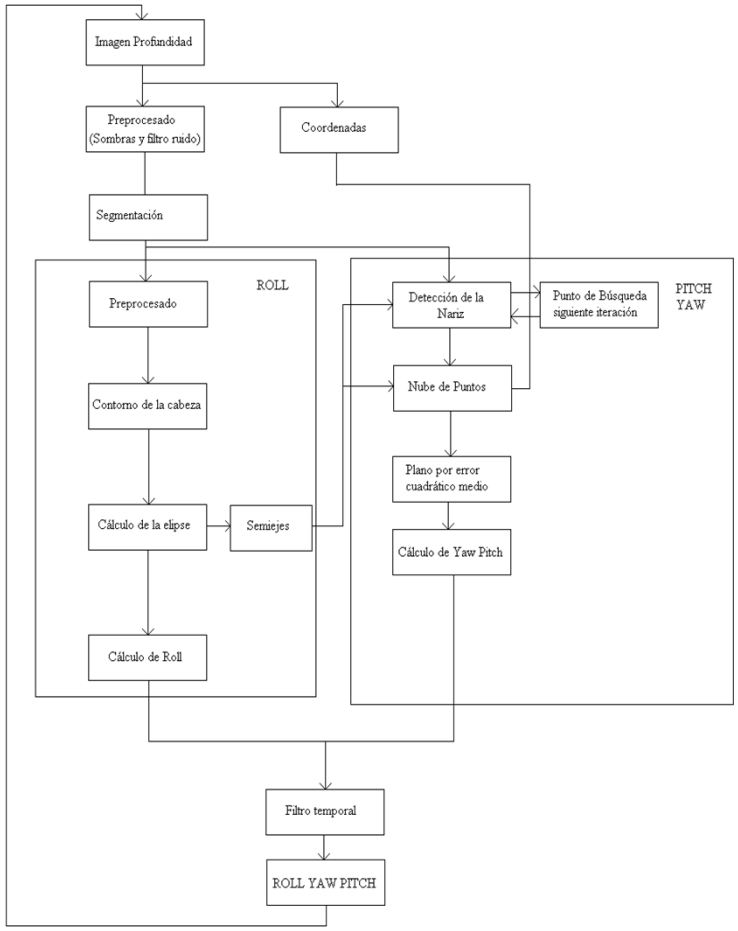
- Cálculo de ángulos pitch y yaw y filtrado temporal

$$Pitch = \cos^{-1}\frac{1}{\sqrt{\left(\frac{-A}{C}\right)^2 + 1^2}} \qquad Yaw = \cos^{-1}\frac{1}{\sqrt{\left(\frac{-B}{C}\right)^2 + 1^2}}$$



Algoritmo propuesto

- Diagrama



Algoritmos offline

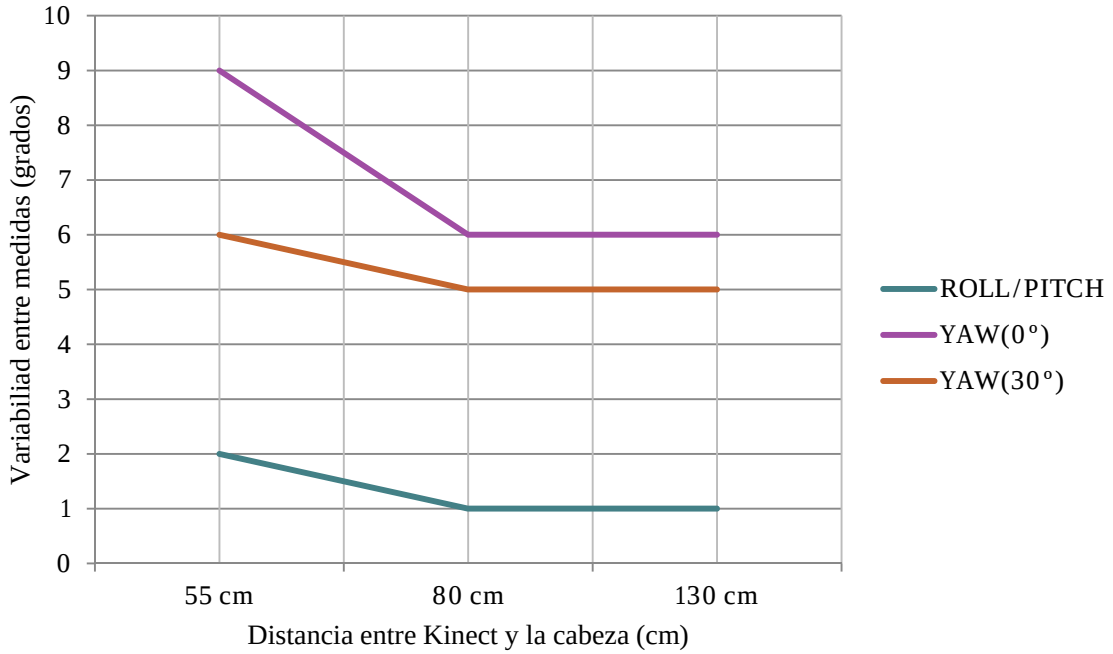
- Función que graba a 15 fps
- Los dos algoritmos son idénticos salvo
  - Filtrado de 3 frames en vez de 2
  - En el algoritmo propuesto el área de búsqueda de la nariz disminuye (18% semieje mayor X 25% semieje menor)
- Crean vídeo para ver los resultados llamados “video.avi” y “video_rotacion.avi”

Proceso en el laboratorio

- Comprobar el rango en profundidad en el que el algoritmo puede funcionar
- Comprobar la variabilidad del resultado entre varias medidas
- Comprobar los ángulos máximos de rotación que puede registrar el algoritmo
- Calcular el tiempo de ejecución de un frame

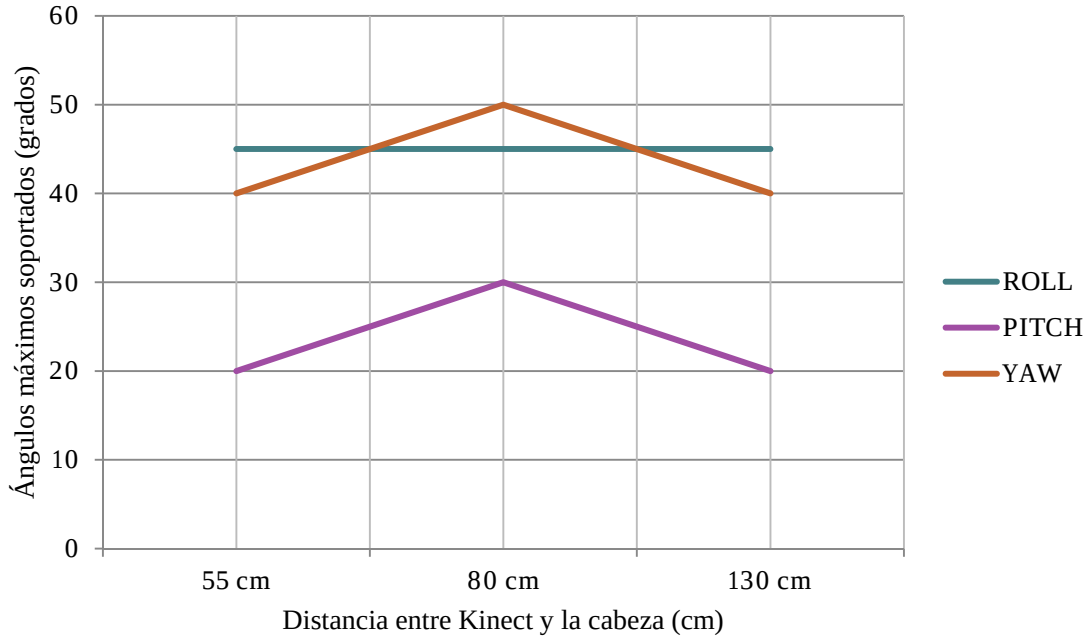
Resultados online

- Rango de 80 cm (50-130 cm)
- Tiempo entre frames 0.4 s
- Variabilidad entre medidas



Resultados online

- Ángulos máximos



Problemas

- Problemas generales
  - Luz fuerte

  - Giros bruscos en ángulo Yaw (>15º entre frames)

Problemas

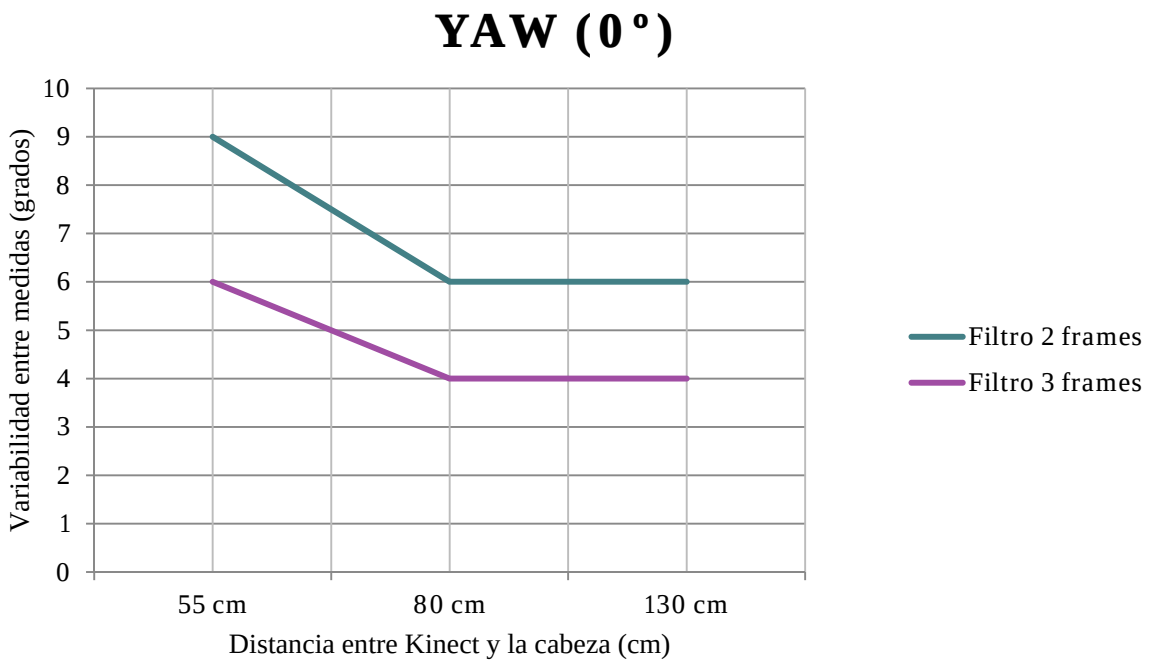
- Problemas al tener la Kinect por debajo de la cara
  - El punto de la nariz se desvía a la barbilla (sobre todo en posiciones cercanas)
  - La elipse no se ajusta bien a la cara si el cuello es muy ancho (sobre todo en posiciones cercanas)

Problemas

- Problemas al estar muy cerca la cabeza de la Kinect (50-55 cm)
  - Con ángulos de Pitch grandes, la cabeza se introduce en una zona sin visión para la Kinect
- Problemas al estar muy alejada la cabeza de la Kinect (>130 cm)
  - Se pierde la nariz con facilidad al mover la cabeza

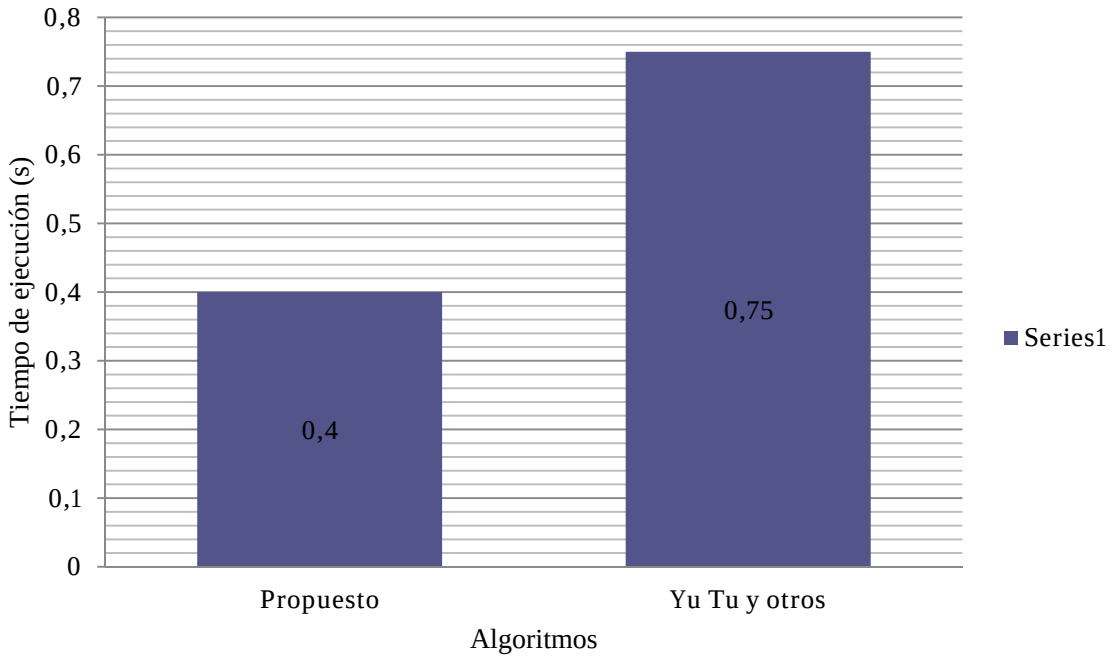
Resultados offline

- Variabilidad de los resultados



Resultados offline

- Tiempo de ejecución
  - Representación 0.15 s
  - Preprocesado 0.12 s
  - Cálculo ángulo roll 0.12 s
  - Búsqueda de la nariz 0.01 s
  - Cálculo ángulos yaw y pitch 0.01 s



Conclusiones

- En la implementación del algoritmo de Yu Tu, Chih-Lin y Che-Hua Yeh el tiempo de procesado y la mala corrección de la rotación de la cabeza impiden encontrar la nariz de forma correcta
- En el algoritmo propuesto la peor resolución la tiene el ángulo Yaw (12% del rango de funcionamiento)
- En el algoritmo propuesto offline, la resolución mejora debido al filtrado temporal de 3 frames
- El punto óptimo de funcionamiento es a 80 cm de la Kinect estando ésta por encima de la cabeza.