TRABAJO FIN DE GRADO:

SEGMENTACIÓN Y RECONSTRUCCIÓN DE SUELO NAVEGABLE MEDIANTE DIVERGENCIA EN LOS ESPACIOS DE COLOR

Autor: Javier Azores Martínez
Tutores: Javier Victorio Gómez González y David Álvarez Sánchez

Índice

Índice de figuras

Índice de cuadros

Resumen

En los últimos años en robótica el área de la visión 3D está en continuo desarrollo gracias principalmente a la aparición de hardware de bajo coste y de librerías de desarrollo open source. Estas herramientas facilitan cada vez más la tarea de modelar el entorno para que robots móviles sean capaces de saber qué zonas son transitables y evitar, de esta forma, los obtáculos que aparezcan a su paso.

Este proyecto parte de uno anterior llevado a cabo por Jose Pardeiro[1]. En él se proponía un algoritmo capaz de detectar suelos lisos transitables haciendo uso de la información recibida mediante una cámara Microsoft Kinect. Para ello se valía tanto de la información 3D que proporciona la cámara como de la información sobre los parámetros de color RGB que captaba del entorno.

Este proyecto se propone, en primer lugar, realizar una serie de pruebas con el algoritmo anterior usando distintos espacios de color. Posteriormente, llevar a cabo la implementación de mejoras a dicho algoritmo para perfeccionar no solo el etiquetado del suelo transitable, sino también el tiempo de ejecución del mismo. Para ello, se optimizará el código y se introducirán nuevos criterios, como la Delta E divergence, que permitirán usar de una forma más efectiva la información del color para la identificación del suelo, evitando al mismo tiempo los obstáculos.

Para tal objetivo se hará uso del lenguaje de programación C++ y de librerías externas como Open CV y PCL.

Abstract

The 3D vision area has become more and more important recently in robotics. It is in continuous development thanks mainly to the new low cost hardware and open source libraries available nowadays. These tools make the task easier to characterize the surroundings so that a mobile robot could recognize the environment and know which are walkable areas. That enables it to avoid the different obstacles that may encounter along its way.

This project is based in a previous one carried out by Jose Pardeiro[1]. That project implemented an algorithm capable of detecting flat walkable floor using for that purpose a Microsoft Kinect camera. The algorithm combined both 3D information and color parameter information (in RGB) provided by the camera to recognize the floor.

The main goals of this project are the following:

Firstly, to carry out several tests on the previous algorithm using three different space colour.

Secondly, to implement various upgrades on the previous algorithm in order to improve its performance: better recognition of the walkable floor and reduction of runtime if possible. To achieve this objectives, the code will be optimized and a new way of computing color divergences wil be implemented. This new criteria to compute color differences, Delta E divergence, will cater for a more effective treatment of color information.

Finally, several tests will be done on this new algorithm to find the best configuration for each environment.

1. Introducción

1.1. Motivación

El algoritmo sobre el que se basa el proyecto era capaz de detectar buena parte del suelo, pero su resultado dependía en gran medida de que las condiciones del entorno le fuesen favorables. Requería de diferencias de color grandes entre el suelo y los obstáculos.

Era incapaz de etiquetar los brillos que aparecen en el suelo, debido a la iluminación artificial en interiores, al producir un cambio importante en la apreciación del color en estos tramos. La utilización del espacio de color RGB para caracterizar el suelo y comparar si un color es suficientemente cercano a otro era también mejorable, pudiendose utilizar otros espacios de color que fuesen más cercanos al modo en que el ojo humano capta los colores.

El tiempo de ejecución se podía reducir optimizando el código en la medida de lo posible.

En general, el algoritmo ofrecía la oportunidad de realizar una gran cantidad de pruebas para examinar el desempeño del mismo con cada uno de los cambios que queríamos introducir. Y de ese modo poder discernir de forma experimental cuál es la forma óptima de aprovechar los parámetros suministrados por la cámara Kinect, con sus limitaciones lógicas, para realizar de la forma más rápida y precisa el etiquetado del suelo transitable.

1.2. Objetivos

El objetivo es crear un algoritmo basándonos en el anterior para que el etiquetado del suelo sea más preciso y de este modo reducir el número de falsos positivos y falsos negativos. Para ello se realizarán toda una serie de pruebas para determinar experimentalmente si los cambios mejoran notablemente los resultados del algoritmo ya existente tanto en la identificación del suelo transitable como en el tiempo de ejecución.

De forma más detallada, se pretende realizar lo siguiente:

- Optimizar el código previo.
- Trabajar con dos espacios de color más al realizar pruebas con el algoritmo anterior, HSV y CIE L*ab.
- Implementar cambios en el algoritmo para realizar la comparación de color de una forma más eficiente para poder etiquetar el suelo transitable de forma más precisa.
- Realizar un amplio número de pruebas en este nuevo algoritmo con distintas nubes de puntos captadas por la cámara variando los distintos parámetros de entrada. Probar las distintas combinaciones posibles y compararlas entre ellas para establecer la óptima.

1.3. Estructura

En este documento se realizará una breve introducción al área de la robótica en la que se ha desarrollado este proyecto y se presentarán las bases teóricas de las herramientas que han sido utilizadas para lograr el objetivo. Así como una descripción a nivel técnico del hardware y las herramientas software necesarias. Posteriormente se detallará la estructura del algoritmo y se mostrará toda una serie de pruebas llevabas a cabo para determinar cuáles son las condiciones y los parámetros de entrada óptimos para obtener el máximo rendimiento del algoritmo.

2. Estado del arte

Se comenzará en esta sección por introducir al lector en el estado del arte de la navegación autónoma de robots y de la visión artificial como principal herramienta.

2.1. Navegación de robots móviles

Hay tres técnicas principales para el desarrollo de la navegación auntónoma de robots:

- Map building and Interpretation: Consiste en la descripción de un escenario usando el marco de referencia de un robot.
- Localisation: Consite en reconstruir el entorno utilizando los sensores disponibles para que el robot sea capaz de conocer su posición respecto del entorno y pueda moverse en consecuencia.
- Path-planning: Utiliza el output del método anterior y una vez conocida ya la posición del robot, requiere también la posición final deseada con respecto al mismo origen de coordenadas.

La navegación que usaremos en este proyecto es la basada en visión. Aquella que usa sensores ópticos para extraer toda la información que sea posible del entorno para poder ubicar al robot. Para logar esto último, es de vital importancia analizar correctamente la información proporcionada por los sensores. Para este proposito existen técnicas como la representación del entorno y algoritmos de localización.

Como muestra, la Universidad de Changzhou[4] (China) creó un sistema de navegación en interiores basado en ultrasonidos. Apoyándose en la información aportada por los sensores de ultrasonidos, junto a otros, le permite generar un mapa del entrono. Con dicho mapa es capaz de conocer las coordenadas de posicionamiento y calcular la mejor ruta para alcanzar el objetivo.

2.2. Visión artificial

Es una de las áreas clave en el desarrollo de la navegación autónoma de robots. Estudia la capacidad para procesar e interpretar la información obtenida por los sensores ópticos.

Se puede dividir el proceso en dos partes, que tendrán lugar de forma continua para cada frame captado por los sensores.

En primer lugar, se recogen todos los datos captados por los sensores. En muchos casos la información procede de cámaras 2D convencionales, aunque también se utilizan infrarrojos o cámaras 3D para este propósito. Dicha información es transmitida habitualmente a la unidad de procesado mediante puertos USB, FireWire o Ethernet.

En segundo lugar, se procesa la información obtenida para traducirla en información utilizable. Existen mutitud de métodos para hacerlo. Entre ellos tenemos la conversión a una escala de grises para facilitar el procesado, segmentación, detección de bordes, reconocimiento de elementos, procesado de aprendizaje mediente redes neuronales o filtrado.

Tras este proceso tiene lugar la toma de decisiones. En el caso de la navegación autónoma de robots, consistirá en elegir la ruta óptima para alcanzar el objetivo.

Este campo ha avanzado mucho tecnologicamente. Sin embargo, el principal handicap es el alto coste de implementar un sistema de navegación basado en visión de calidad. Por tanto, una parte importante de la investigación en este campo se centra en implementar dicha tecnología y obtener buenos resultados con equipos de bajo coste.

Como muestra de esas investigaciones, la Universidad de Almería publicó un sistema para la automatización del riego en invernaderos[5]. Para conseguirlo usaban un robot que poseía todo lo necesario para realizar el riego e integraba también un sistema de navegación. Con estos medios implementaron dos soluciones, por un lado el uso de una cámara web para recibir la información y un equipo empotrado para procesarla, y por otro una cámara inteligente que realiza todos los pasos. A pesar de usar equipos diferentes, ambas soluciones se basaban en la necesidad de localizar el centro del pasillo del invernadero.

También existen variedad de proyectos basados en la navegación en interires, como el desarrollado por la Universidad de Tianjin[6] (China). En él, una cámara colocada en el techo es capaz de segmentar el suelo y convertirlo en mapas de navegación que el robot puede usar para moverse conociendo el entorno.

Con este proyecto se busca contribuir a este estado del arte proporcionando un algoritmo capaz de etiquetar el suelo navegable en interiores por que el que se podría mover un robot sin chocar con obstáculos. Y todo ello con la principal ventaja de poder hacerlo con un hardware de bajo coste.

3. Fundamentos teóricos

3.1. Óptica

Un efecto óptico de utilidad para este proyecto ha sido el efecto pinhole. Consiste en la formación de imágenes invertidas por la luz que atraviesa un pequeño orificio. Es la base de la cámara oscura y por tanto de la cámara fotográfica, e incluso es la base del funcionamiento del ojo humano. Si bien es verdad que tanto nuestro ojo como la cámara fotográfica hacen uso de lentes que enfocan la imagen (el cristalino en nuestro caso y el objetivo en el otro).

Es decir, al dejar pasar la luz primero por una lente y después por un pequeño orificio, la información 3D del mundo se proyecta sobre un plano como una imagen invertida cuyo tamaño es proporcional al de los objetos reales y se puede variar modificando la distancia a la lente. Esta distancia se conoce como distancia focal.

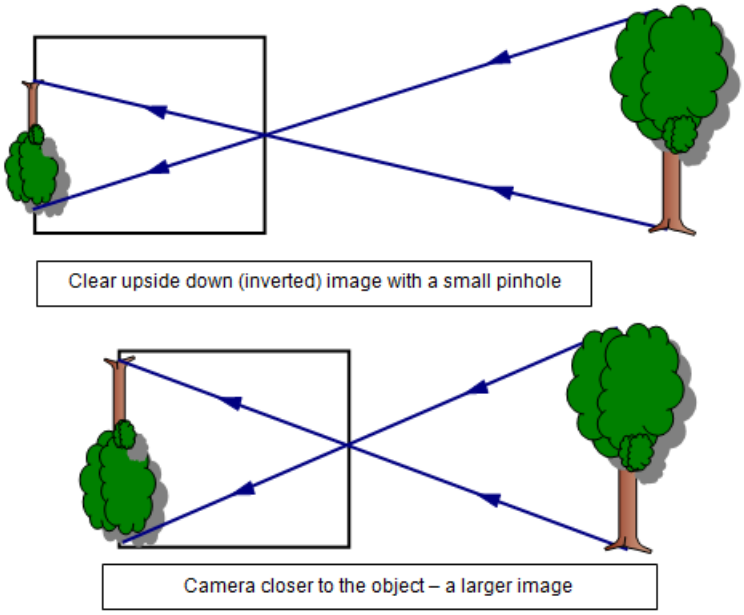

Figura 1: Ejemplo de modelo pinhole. Imagen cortesía de http://www.schoolphysics.co.uk

En medidas cercanas el modelo pinhole posee una gran exactitud que va perdiendo según se aleja el objeto.

Se han creado técnicas que reproducen el efecto pinhole, como *range imaging*. Es un conjunto de técnicas que se usan para producir una imagen en 2D mostrando la distancia de los puntos respecto a uno específico, generalmente asociado al tipo de sensor. El número de píxeles de la imagen obtenida depende de dicha distancia. Si la cámara está bien calibrada, con este método se puede conocer la posición de cada pixel de la imagen en 2D partiendo de la distancia al punto en 3D.

Si se tiene en un cuenta un píxel (xd,yd), una distancia focal (fxd,fyd) y una

profundidad depth(xd, yd), se puede conocer la situación real de cada píxel haciendo uso de las siguientes ecuaciones:

$$P3D.x = (xd - cxd) * depth(xd, yd)/fxd \quad (1)$$

$$P3D.y = (yd - cyd) * depth(xd, yd)/fyd \quad (2)$$

$$P3D.z = depth(xd, yd) \quad (3)$$

Para poder obtener los parámetros, es necesario calibrar la cámara. Para ello, se suele partir de un tablero de ajedrez impreso en A3 como el mostrado a continuación:

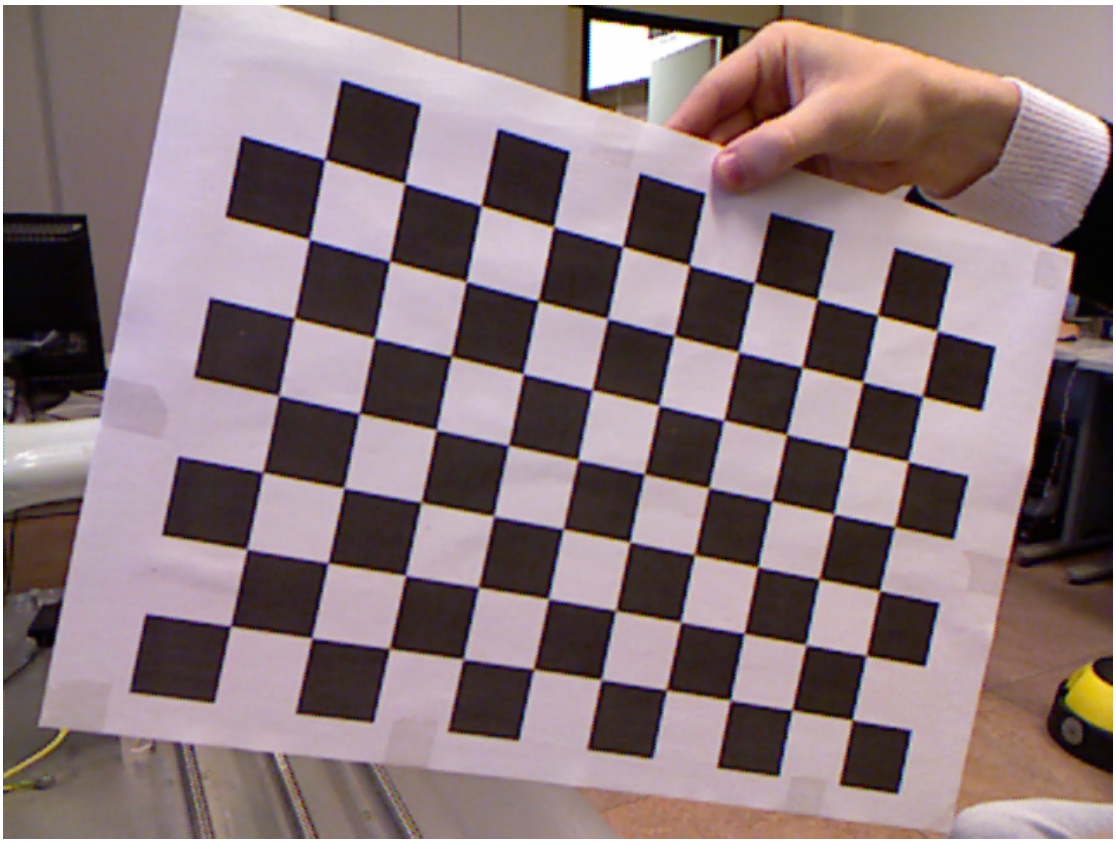

Figura 2: Ejemplo de tablero A3. Imagen cortesía de Nicolas Burrus

Usando ese tablero, de dimensiones conocidas, y colcándolo a distancias también conocidas, se puede calibrar la cámara y obtener los valores de profundidad.

3.2. Espacios de color

Un espacio de color es un sistema que describe los colores matemáticamente. Esto permite reproducir un mismo color de forma exacta. En realidad no es más que un modelo matemático abstracto que pretende describir los distintos colores posibles como un conjunto de números (normalmente usan tres cifras). Esto da lugar a un sistema de coordenadas donde cada punto representa un color distinto.

3.2.1. RGB

El espacio de color RGB utiliza la intensidad de los tres colores primarios, rojo, azul y verde, para generar el resto de colores combinandolos. Se trata de un modelo basado en la síntesis aditiva. Es el más común en sistemas electrónicos o en fotografía.

Cada color primario queda definido por un *byte* de información, y su valor, que va de 0 a 255, implica cuanto afecta en el resultado, siendo 0 que no interviene y 255 que tiene su máxima intensidad. El negro se representa con tres ceros y el blanco con los tres valores a su máximo. Se suele representar en forma de cubo:

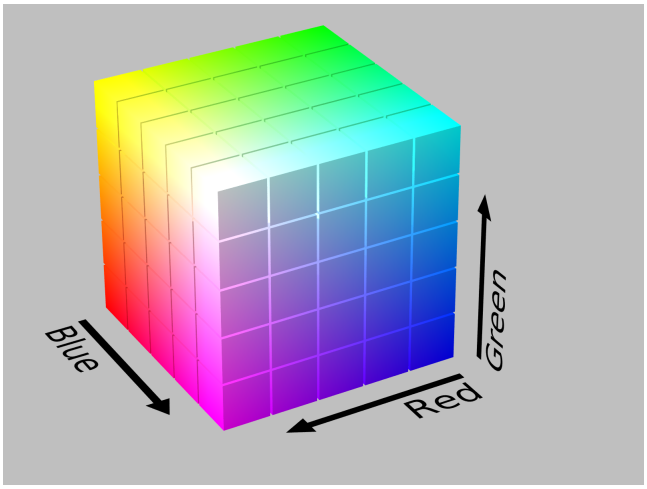

Figura 3: Cubo de color RGB

Este modelo de color no define por sí mismo lo que significa rojo, verde y azul, por lo que se conoce como un espacio de color no absoluto al depender del dispositivo en el que se reproduzca. Sin embargo, se han creado espacios de color absolutos a partir de este, como el sRGB que es en el que trabaja la cámara Kinect, tal como informa Microsoft[2].

Como veremos más adelante, el inconveniente de trabajar con RGB ,o sRGB en nuestro caso, es que este no describe los colores de forma lineal respecto a como lo capta el ojo humano. Es decir, lo que el ojo capta como pequeñas variaciones de color puede producir grandes variaciones en los parámetros RGB y viceversa.

3.2.2. CIE L*ab

Tambien conocido como CIE Lab 1976. Sus tres variables son:

- L*: Representa la luminosidad. Oscila entre 0, negro; y 100, blanco difuso.
- a*: Posición entre rojo y verde.
- b*: Posición entre amarillo y azul.

Este modelo cromático se usa para describir todos los colores que capta el ojo humano. Fue desarrollado especificamente por la CIE (*C*ommission Internationale de l'Eclairage) con el propósito de linelizar las diferencias de color perceptibles por el ojo basándose en el espacio de color CIE XYZ creado anteriormente en 1931.

Al ser mucho más lineal que RGB, nos será más útil a la hora de comparar diferencias de color. Para ello habrá que realizar una transformación desde el espacio sRGB en el que trabaja la cámara hasta este. Para conseguirlo habrá que realizar unas transformaciones intermedias: primero desde sRGB a RGB, después a CIE XYZ y por último a CIE L*ab.

En un primer momento tenemos los valores r, g y b en el espacio sRGB (representados por α) y se transforman a R, G y B (representados por β) en el espacio RGB lineal.

$$if\ \alpha \leq 0,04045 \Rightarrow \beta = \frac{\alpha}{12,92} \tag{4}$$

$$else\ \Rightarrow \beta = \frac{\alpha + 0,055}{1,055} \tag{5}$$

Una vez conocidos los valores RGB, conseguimos XYZ:

$$\begin{vmatrix} X \\ Y \\ Z \end{vmatrix} = \begin{vmatrix} 0{,}412453 & 0{,}357580 & 0{,}180423 \\ 0{,}212671 & 0{,}715160 & 0{,}072169 \\ 0{,}019334 & 0{,}119193 & 0{,}950227 \end{vmatrix} \begin{vmatrix} R/255 \\ G/255 \\ B/255 \end{vmatrix} \tag{6}$$

Por último, se obtienen los valores L*ab:

$$if\ Y \geq 0,008856 \Rightarrow L* = 116 * \sqrt[3]{\frac{Y}{Yn}} - 16 \tag{7}$$

$$else\ \Rightarrow L* = \frac{Y}{Yn} * 903,3 \tag{8}$$

$$a* = 500 * [f(X/Xn) - f(Y/Yn)] \tag{9}$$

$$b* = 200 * [f(Y/Yn) - f(Z/Zn)]$$

Siendo la funcion f(t):

$$if\ t \geq 0,008856 \Rightarrow f(t) = \sqrt[3]{t} \tag{11}$$

$$else\ \Rightarrow t = 7{,}787 * t + 16/116 \tag{12}$$

3.2.3. HSV

Hue (Matiz), Saturation (Saturación) and Value (Valor) o HSV es un espacio de color compuesto por las tres variables que incluye su nombre. Se representa en coordenadas cilíndricas. Para ser más presicos:

- H: Representa un ángulo en grados entre 0º y 360º. Cada valor corresponde a un color.
- S: Se representa como la distancia al eje de brillo negro-blanco. Oscila de 0 a 100 según la saturación. Cuanto menor sea este valor, más grisacio y decolorado se verá el color en cuestion.
- V: Representa la altura en el eje blanco-negro. Varía de 0 a 100 también. Cuanto mayor es, más claro se hace el color.

Para tener una mejor idea, su representación en tres dimensiones es:

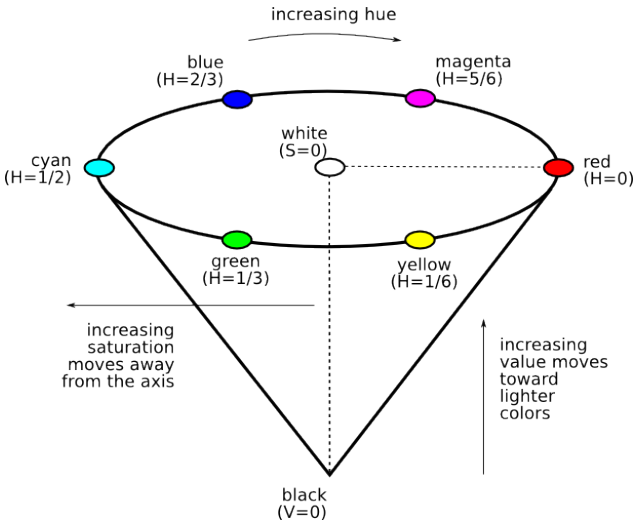

Figura 4: Coordenadas cilindricas HSV

Este espacio de color tampoco tiene una trasformación lineal a RGB, ya que al igual que CIE L*ab intenta acercarse al modo en que el ojo capta las transiciones de los colores.

3.2.4. Delta E divergence

En 1976 la Comisión Internacional de la Iluminación (CIE) propuso una unidad de medida para cuantificar las diferencias entre colores, conocida como Delta E. Fue desarrollada conjuntamente con el primer espacio de color que pretendia ser perceptualmente uniforme, CIE L*ab.

Delta E es un número que representa la distancia entre dos colores. La idea fue hacer que un valor igual a 1 fuese la menor diferencia de color que pudiese apreciar un observador entrenado bajo condiciones ideales de comparación. La realidad es que el espacio L*ab no es ideal, no es completamente uniforme como pretendían y por tanto una misma Delta E no se aprecia exactamente igual para dos colores cualquiera, pero se aproxima bastante.

Matemáticamente la Delta E divergence no es más que la distancia euclídea entre dos puntos distintos - es decir, colores - en el espacio L*ab.

$$DeltaE = \sqrt{(L_1^* - L_2^*)^2 + (a_1^* - a_2^*)^2 + (b_1^* - b_2^*)^2} \tag{13}$$

3.3. Cluster k-means

La separación por grupos resulta clave siempre que se manejen gran cantidad de datos. En este caso, al manejar gran cantidad de puntos captados por la Kinect será necesario usar algún método que permita realizar agrupaciones de puntos similares entre si.

Entre los distintos métodos de clusterin destaca uno de ellos por su sencillez, k-means. El proceso que sigue este método es el siguiente:

Se parte de un número n de datos en los que se quiere realiza k agrupaciones.

En un primer momento se eligen al azar k datos como centro de los grupos y se calcula la distancia de cada dato a cada centro. Se reagrupan los datos según el centro que tengan a menor distancia, y se calcula un nuevo centro para cada grupo que se encontrará en el punto medio del conjunto de datos pertenecientes. Se repetirá el proceso en bucle hasta que de una iteración a la siguiente ningún punto cambie de grupo. En ese momento ya quedarán definidos los k grupos distintos.

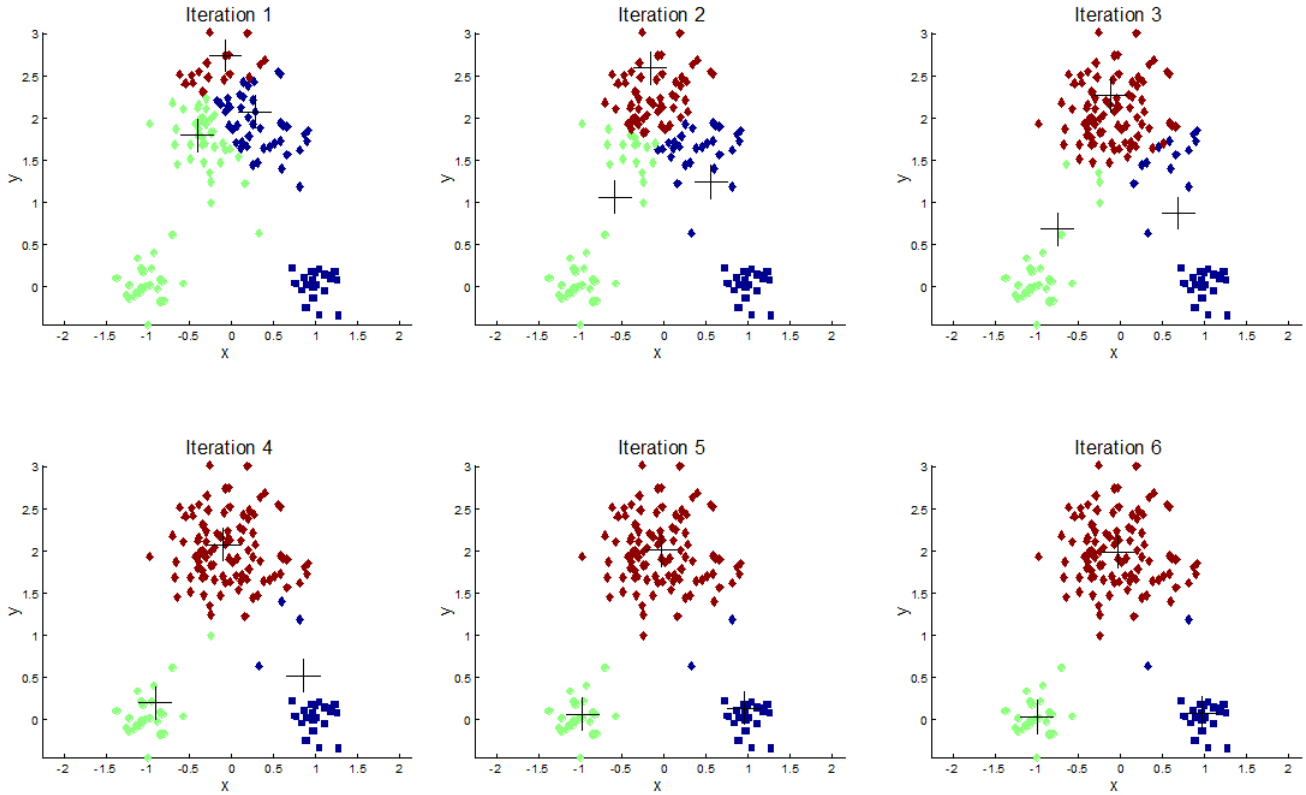

Figura 5: Proceso de clustering k-means

4. Tecnologías asociadas

En este apartado tendrá lugar una breve descripción de la tecnología en la que se ha basado el desarrollo de este proyecto. Principalmente la cámara kinect, el lenguaje de programación utilizado y las librerías adicionales que han resultado de ayuda para llevar a cabo el proyecto.

4.1. Cámara Microsoft Kinect

El principal hardware del que haremos uso será la cámara Kinect. Fue lanzada al mercado a finales del 2010 cosechando un gran éxito no solo en sector de los videojuegos, tanto que Microsoft finalmente accedió a liberar los drivers.

Consta de un sensor de profundidad, una cámara RGB y un microfono multi-array. Con ello es capaz de captar movimiento en 3D y posee reconocimiento facial y de voz.

El sensor de profundidad consiste en dos sensores infrarojos y un sensor CMOS monocromo que es capaz de capturar imagenes en 3D. Sin embargo, el rango de detección de profundidad no es todo lo amplio que nos gustaría y por ello haremos uso en este proyecto de la información que aporta la cámara RGB para ampliarlo. Para distancias muy cercanas es problematico, y demasiado lejos no funciona. También tiene el inconveniente de que al usar laser infrarrojo solo funciona en unas condiciones de iluminación muy concretas que se dan solo en interiores.

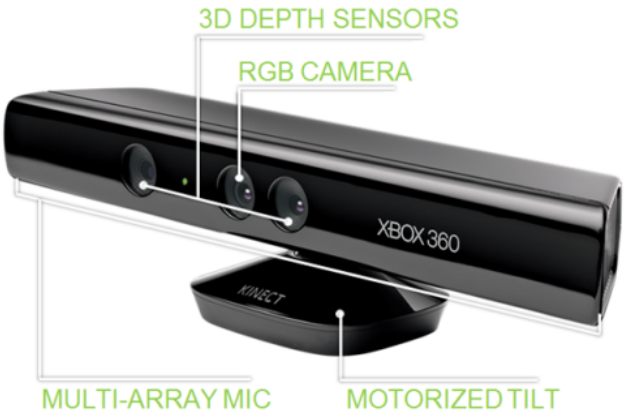

Figura 6: Cámara Kinect

La cámara RGB usa por defecto una resolución VGA de 8 bits (640 x 480) con un filtro de color Bayer y la salida de video funciona a 30 Hz. Su rango de trabajo es de 1,2m a 3,5m y posee un ángulo de visión horizontal y vertical de 57º y 43º respectivamente.

Necesita alimentación externa para funcionar mediante cable USB.

4.2. C++

Se trata de uno de los lenguajes de programación más utilizados en la actualidad. Fue creado Bjarne Stroustrup a mediados de los años 1980 con la intención de extender el lenguaje C con mecanismos para la programación orientada a objetos. Esto dio lugar a un lenguaje híbrido de gran potencia y flexibilidad.

Actualmente existe un estándar, ISO C++, al que se han unido la mayoria de fabricantes de compiladores actuales.

4.3. Herramientas de desarrollo

En esta sección se hará un breve repaso de las herramientas utilizadas para poder programar en el lenguaje C++.

4.3.1. Colección de compiladores GNU

Los compiladores GNU (*GNU Compiler Collection*), GCC, son unas herramientas libres creadas por el proyecto GNU bajo la licnecia GPL. Estos compiladores son básicos en los sistemas basados en UNIX[3].

Actualmente GCC soporta los lenguajes más comunes, como C, C++, Java o Fortran, mientras que de forma no estándar se añaden otros como mercury o VHDL.

4.3.2. CMake

CMake es una herramienta que que se encarga de la generación de código para ayudar a portarlo entre distintas plataformas o compiladores. Para ello genera un proyecto siguiendo una serie de instrucciones marcadas por el usuario, facilitando el desarrollo multiplataforma.

Para poder usar CMake correctamente es necesario crear un fichero de nombre *CMakeLists.txt*, debe contener los parámetros necesarios, como ficheros que forman parte de la compilación, librerías que hay que incluir o el nombre final de la aplicación. CMake lee dicho archivo y genera un proyecto adecuado al Sistema Operativo, la compilador elegido y la ubicación de las librerías que van a ser enlazadas.

4.4. Licencia GPL

Una de las bases del proyecto es la de realizar un código extensible, modular y multiplataforma, que pueda servir como base o complemento para futuros proyectos con Kinect. Por ello será publicado bajo la llamada licencia *GNU General Public License* (GPL).

Esta licencia fue creada por la *Free Software Foundation* en 1989 con la intención de proteger el software libre e impedir la apropiación del código y el robo de libertades.

En términos legales, la licencia GPL actúa como un contrato al ceder algunos derechos al usuario, siendo una licencia reconocida legalmente.

4.5. Bibliotecas

En términos de programación, una biblioteca o librería es el conjunto de subprogramas que se emplean para desarrollar otros programas, proporcionando servicios que pueden facilitar el desarrollo. Los programas se enlazan con las librerías mediante llamadas en el código fuente, lo que hace que a la hora de compilar dichas bibliotecas se enlacen en el ejecutable.

Las librerías son utilizadas constantemente en programación. Todos los lenguajes poseen sus propias librerías que facilitan tareas tan habituales y comunes como la muestra de elementos por pantalla. Una de las grandes facetas de C++ es la de poder utilizar conjuntos de librerías externas al lenguaje, pero programadas en él, creadas por terceros, proporcionando un abanico de posibilidades inmenso, que en este proyecto serán aprovechadas para simplificar el trabajo.

4.5.1. Point Cloud Library

Point Cloud Library (PCL) son un conjunto de librerías que integran multitud de algoritmos para el trabajo con nubes de puntos tridimensionales y procesamiento geométrico. Concretamente, contienen algoritmos variados para filtrado, estimaciones, reconstrucción de superficies, registro, segmentación o adecuación de modelos que son muy útiles en el campo de la robótica. Se desarrolla de forma comunitaria por contribuyentes de todo el mundo, está escrita en su mayoría en C++ y liberada bajo la licencia BSD, permitiendo que puedan ser usadas en cualquier clase de proyecto, no necesariamente libre.

4.5.2. OpenCV

OpenCV se trata de una biblioteca *open-source* de visión artificial creada en sus orígenes por Intel. Está liberada bajo la licencia BSD, la cual permite que se integre en cualquier proyecto siempre que se cumplan las condiciones de la licencia. Es una librería multiplataforma, con versión para Windows, MacOS X y GNU/Linux, y contiene una gran cantidad de funciones dentro del área de la visión.

La librería se encuentra escrita en C y C++ de forma muy optimizada, dando un gran rendimiento. Actualmente se utiliza en multitud de aplicaciones, como seguridad o control de procesos.

5. Algoritmo propuesto para etiquetado de suelo navegable

En esta sección se explicará en detalle el funcionamiento del algoritmo paso a paso. Primero se hará una breve introducción que muestre cuál es el objetivo ideal que se desea implementar y que fue el objetivo con el que este proyecto comenzó. Posteriormente se seccionará el programa en sus distintas etapas para analizarlas por separado y de ese modo tener una visión global del conjunto.

5.1. Introdución

El objetivo con el que comenzó este proyecto fue el de mejorar el algoritmo realizado en un proyecto anterior por José Pardeiro[1]. Se busca tratar de paliar las limitaciones de la cámara kinect para expandir su rango de trabajo más allá de la restringida información 3D que aportaban los sensores laser. Para tal propósito haremos uso de la información de color que proporciona la cámara RBG. Se busca etiquetar una mayor parte del suelo navegable que el algoritmo anterior -aumentar *true positives*- evitando ,al mismo tiempo, cometer más errores y pagar el precio de considerar como suelo lo que son obtáculos -*false positives*-. Ese es el reto, pues siempre es posible aumentar los *true positives* relajando las condiciones de etiquetado, lo que lleva a un aumento también de los *false positives*.

Para entenderlo mejor se muestra gráficamente de qué situación partimos y lo que queremos conseguir.

Figura 7: Captura RGB

Partiendo de un pasillo como el mostrado anteriormente, podemos discernir los puntos que pertenecen al suelo fácilmente usando la información del sensor 3D:

Figura 8: Captura RGB

Pero el problema, como se puede apreciar, es el ya expuesto. El rango del sensor es bastante limitado.

Para intentar solucionarlo haremos uso de la información en 2D que nos aporta la cámara RGB. Usando de forma efectiva en nuestro algoritmo los parámetros de color que nos aporta la cámara, intentaremos llegar a un resultado que se aproxime lo más posible al siguiente:

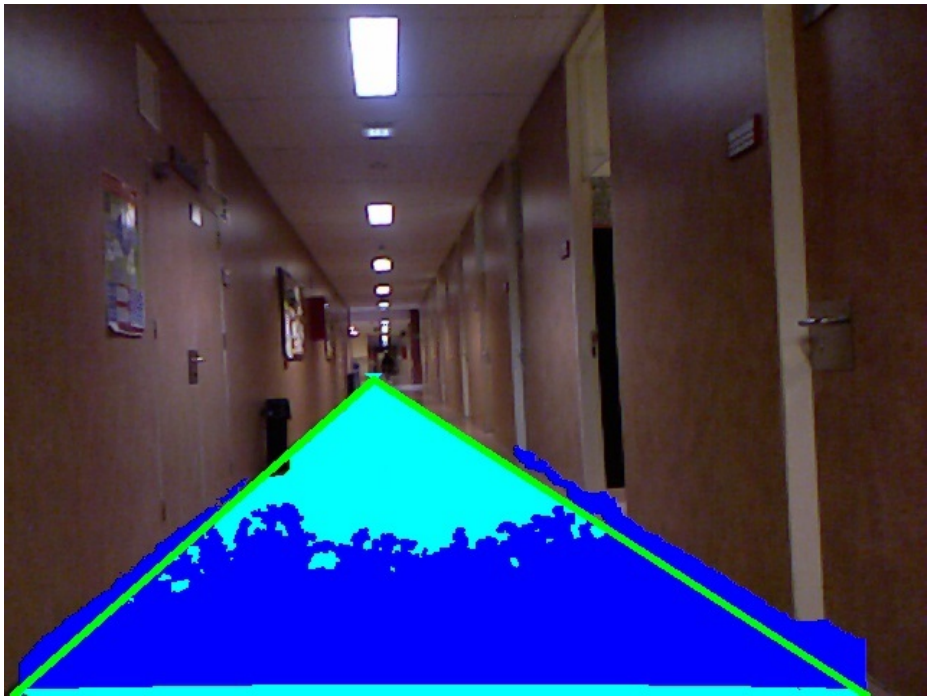

Figura 9: Captura RGB

Es decir, que etiquete como suelo transitable lo máximo que pueda dejando fuera a los obstáculos, como la papelera en este caso.

A parte de conseguir esto, hay que tener en cuenta de dónde partimos. Para poder mejorarlo hay que saber qué resultados arrojaba el algoritmo anterior configurado con los parámetros que se dedujeron como óptimos. En esas circunstancias, se obtenía el siguiente resultado:

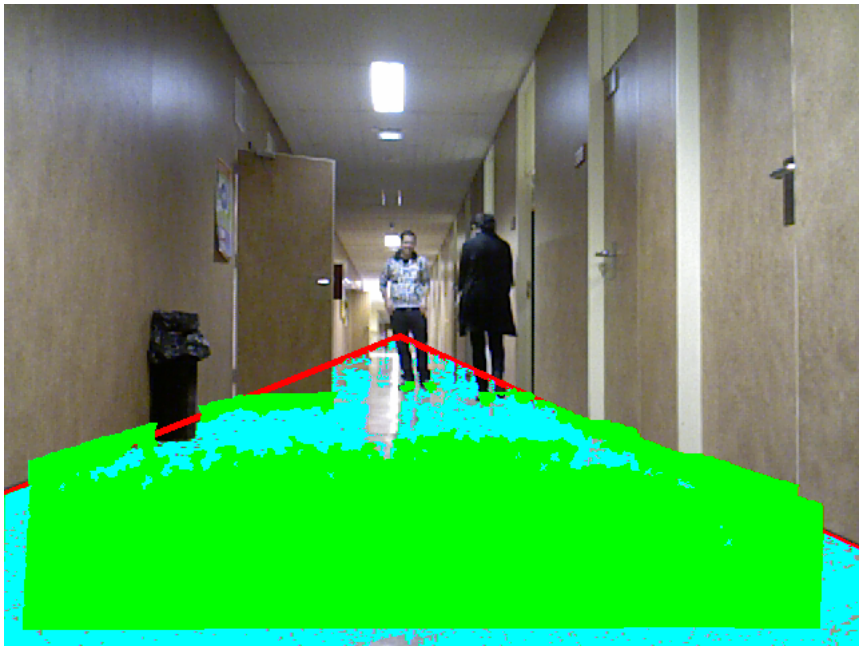

Figura 10: Captura RGB

No es posible en esta situación mejorar el etiquetado del suelo sin evitar que empiece a etiquetar también obstáculos como la puerta, la papelera o las piernas de las personas. Por tanto buscamos mejorar ese programa para ser capaces de llegar a un resultado que mejore lo anterior, como por ejemplo el de la figura 11. Este es, como veremos más adelante, el mejor resultado que podemos obtener con el nuevo algotimo en este entorno, configurado con los parámetros de entrada óptimos.

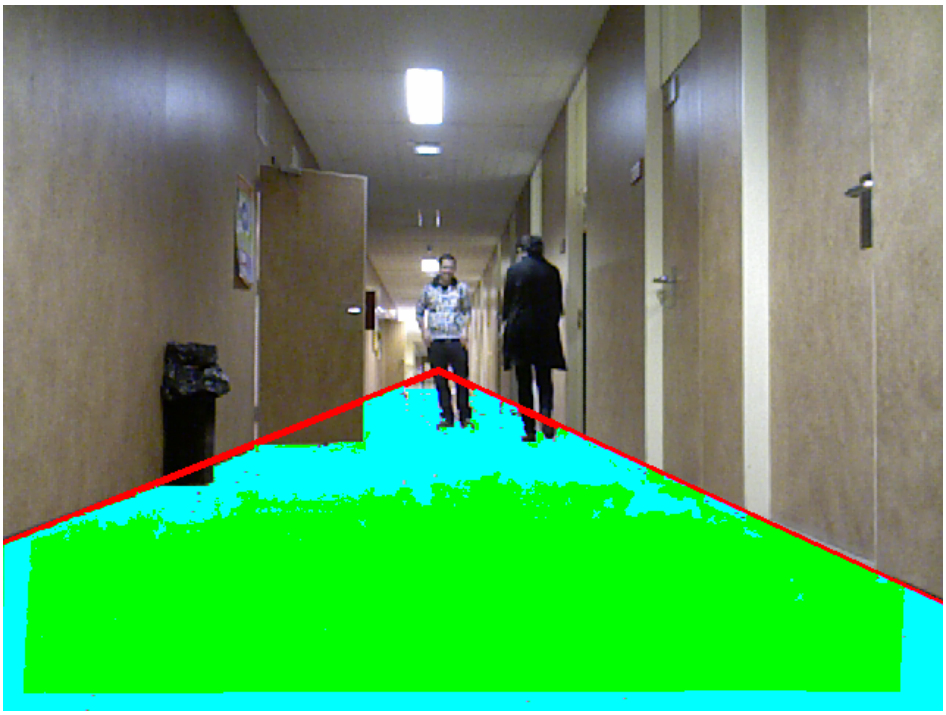

Figura 11: Captura RGB

Port tanto, es evidente la mejora en el etiquetado de este nuevo algoritmo.

El diagrama de flujo del programa es:

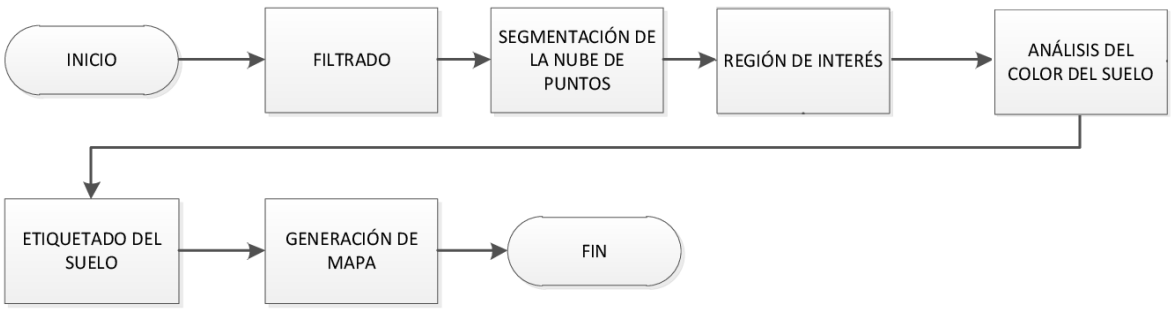

Figura 12: Diagrama de flujo general del algoritmo

En un primer momento se filtra la nube de puntos inicial para trabajar con menos información y agilizar la segmentación posterior. Una vez segmentada la nube filtrada en sus distintos plano (suelo, techo y paredes), se calcula la región de interés. Posteriormente, se realiza un análisis de las propiedades de color de los píxeles pertenecientes a la región de interés para de ese modo poder etiquetar como suelo navegable a los que cumplan ciertas condiciones. Por último se genera una vista aérea del suelo donde se muestra la zona etiquetada.

En la siguiente sección se explica más detalladamente cada una de las etapas.

5.2. Partes del algoritmo

Una vez conocido el objetivo del proyecto, toca describir cada una de las partes que lo conforman en más detalle.

Al comienzo del algoritmo, la kinect nos aporta una nube de puntos con la que tendremos que trabajar. Cada punto de la nube corresponde a un pixel de la imagen (640 x 480) que capta la cámara RBG. Para algunos de esos puntos tenemos la toda la información: posición en x, y, z; y valores RGB del pixel. Pero como ya hemos mencionado esto solo ocurre en un rango muy limitado, en el resto de puntos solo nos asiste la cámara VGA que nos aportará el color en sRGB.

Partiendo de esta base, los pasos que tendrán lugar son:

5.2.1. Filtrado

El filtrado consiste en la eliminación de un exceso de datos tratando de suprimir información redundante para quedarnos con la mínima indispensable. Esto supone un ahorro de cálculos y tiempo para realizar la segmentación más adelante.

Haciendo uso de un filtro tipo *voxel grid* se genera una rejilla de cubos de idéntico tamaño en la nube de puntos. Para cada cubo, que contendrá varios puntos en su interior, se genera un solo punto que sustituirá a todos los anteriores y que se situará en el centroide.

De este modo se obtiene una nube filtrada con menor número de puntos con los que trabajar sin que se pierda la información característica que nos aportaba la nube inicial. Para ello hubo que encontrar empíricamente un tamaño correcto para el cubo del filtrado, un punto intermedio entre ligereza y fidelidad con la nube inicial.

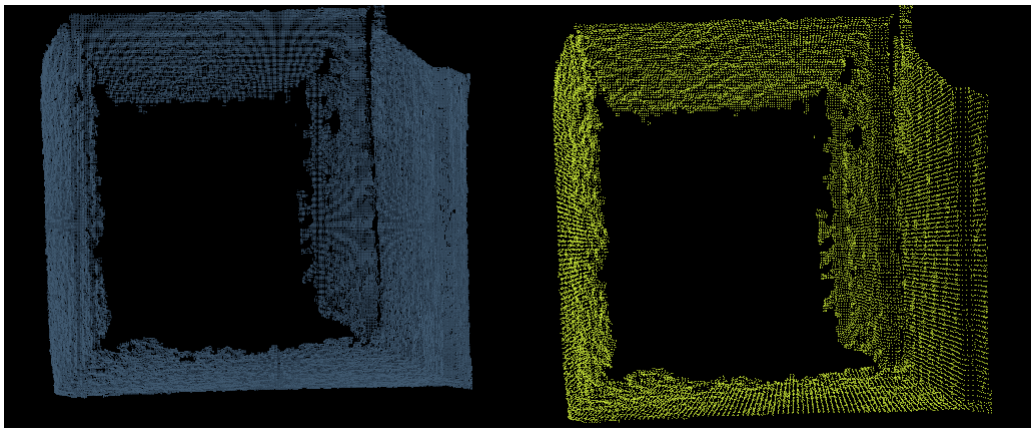

Figura 13: Nube original frente a filtrada

La nube de puntos original posee 307200 puntos (640 x 480) y tras el filtrado con el tamaño del cubo que hemos elegido el número de puntos se reduce entorno a un 12 %.

5.2.2. Segmentación

Dado que la parte que interesa en este proyecto es el suelo, en este paso se va a proceder a segmentar la nube de puntos ya filtrada en sus distintos planos: suelo, techo y paredes.

Para tal propósito se utilizará el algoritmo RANSAC (RANdom SAmple Consensus). Este método ajusta, de forma iterativa, los parámetros que definen a un plano que abarque el mayor número de puntos y evalúa las distancias entre puntos para discriminar los que superen un cierto límite, minimizando así los errores.

Una vez se han obtenido los coeficientes de cada uno de los planos ($ax+by+cz+d = 0$), el programa los analiza para determinar cuál es cuál: suelo, techo, pared izquierda y pared derecha. Para tal análisis geométrico hay que tener en cuenta que la posición de la kinect es paralela al suelo en su eje Z y perpendicular en su eje Y. Por tanto, si el valor absoluto del coeficiente a está comprendido entre 0,9 y 1 se tratará de un plano vertical, pudiendo ser la pared derecha o izquierda. Para determinar de cual se trata, habrá que prestar atención al valor de su coeficiente d. Si es positivo está situado a la derecha mientras que si es negativo será la pared izquierda. Por contra, si es el coeficiente b el que se encuentra entre 0,9 y 1 el plano es horizontal y entonces habrá que examinar su coeficiente d para más información. En este caso si d es positivo indica techo y en caso contrario suelo.

Tabla 1: Coeficientes de los planos generados

	a	b	c	d
Pared dch.	0.990	0.995	0.124	1.134
Pared izq.	0.995	-0.030	0.093	-1.204
Techo	-0.047	0.999	0.025	0.920
Suelo	-0.044	-0.999	-0.01	-1.954

Finalmente, se obtienen cuatro nubes de puntos representando cada una a una de las cuatro superficies. Esto descarta de nuevo a un gran número de puntos que no entran en ninguna de las nubes anteriores.

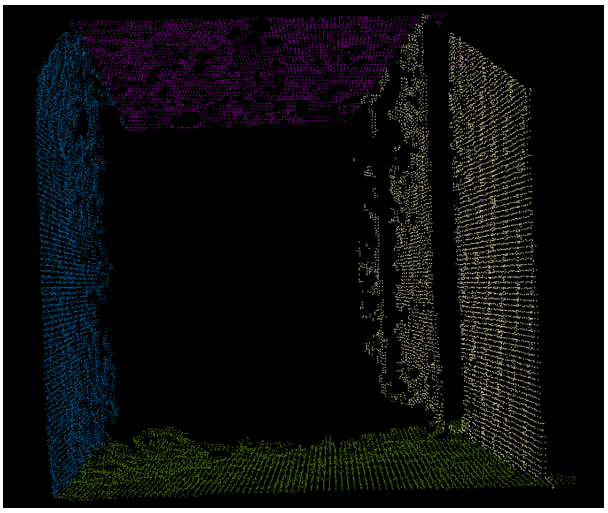

Figura 14: Nubes de puntos segmentadas

El diagrama de flujo de esta sección se ver en la figura 15:

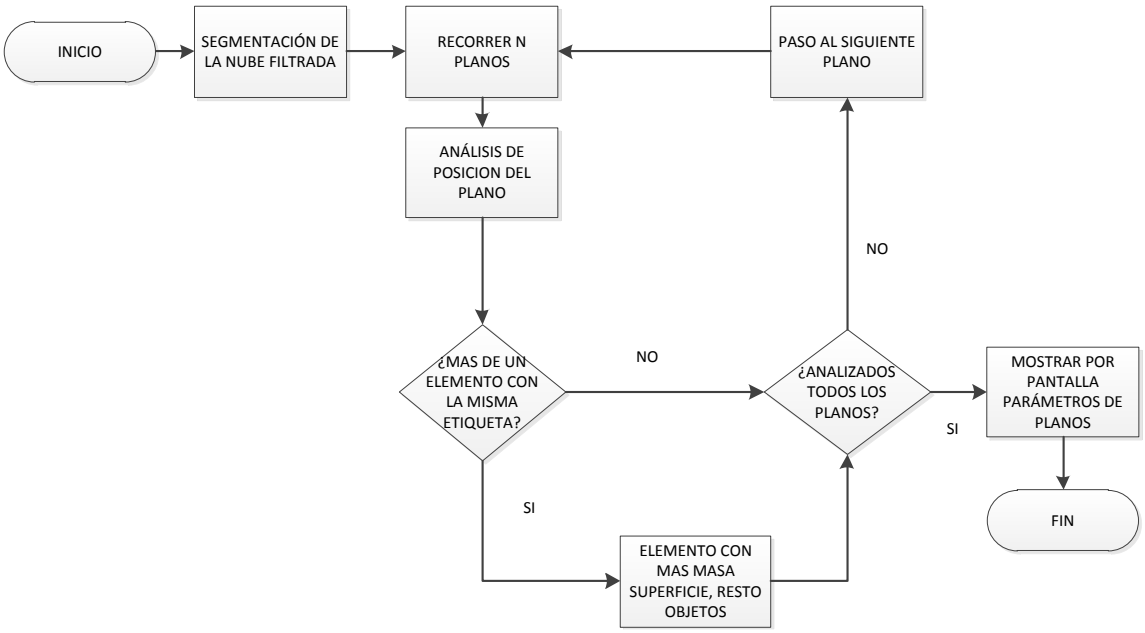

Figura 15: Diagrama de flujo del proceso de segmentación

La gran ventaja de conocer la ecuación que describe el plano del suelo es que ahora se puede deducir qué puntos de la nube inicial pertenecen a dicho plano y por tanto al suelo. Esto solo se prodrá realizar para los puntos que entren dentro del rango de medida de los sensores laser, ya que son los únicos de los que se tiene acceso a su posición en 3D (x,y,z) y estos datos son los que usaremos para comprobar si cumplen la ecuación del plano del suelo.

En realidad el requisito no será que lo cumplan a la perfección, se introducirá cierta tolerancia en el criterio. Se considerarán puntos que pueden pertenecer al suelo a aquellos que se encuentren en un rango de 10 cm de distancia al plano que describe el suelo. De ese modo, ya tendremos en un primer momento una serie de puntos -correpondientes a píxeles en la imagen en 2D- que podrían ser considerados suelo navegable. Para que lo sean definitivamente, tendrán que cumplir otro requisito más adelante, cuando se realice el clustering.

5.2.3. Cálculo de la región de interés

La región de interés, conocida como ROI (Region Of Interest) por sus iniciales en inglés, es la parte de la imagen -conjunto de píxeles- que pueden ser suelo navegable. Conocer la ROI es útil para ahorrar tiempo y trabajo al algoritmo, pues no será necesario considerar los píxeles que se encuentren fuera de esta región a la hora de deducir si pertenecen al suelo navegable o no.

Para calcularlo se utilizará la información obtenida en la segmentación. Conociendo la ecuación del plano del suelo y de las dos paredes, se calcula la intersección entre ellos. Se obtienen dos rectas que se proyetarán sobre la imagen en 2D, como se ve a continuación:

Figura 16: Rectas intersección

Todos los píxeles encerrados entre las dos rectas será los pertenecientes a la ROI, que serán utilizados más adelante.

5.2.4. Análisis de parámetros de color

Una vez que se conocen los primeros píxeles (o puntos) que pertenecen al suelo, gracias a la información 3D de los sensores laser, el resto de puntos hay que extraerlos usando la información sobre el color que nos aporta la cámara VGA que posee la kinect.

Hay dos procesos en este apartado:

- El primero es un clustering (en L*ab) para dividir según su color los píxeles que se encontraban en un rango de 10 cm al plano del suleo, calculados anteriormente al final de la segmentación. El número de grupos del clustering, k, es una de las variables de entrada del algoritmo y se puede modificar a placer como haremos más adelante en la sección de pruebas. La finalidad de esta división en grupos según su tonalidad es la de quedarnos solo con el cluster de mayor tamaño y etiquetar esos píxeles como suelo transitable.

  Con esto se consigue descartar un conjunto de píxeles que podían ser obstáculos y no parte del suelo transitable, puesto que el único requisito que habían pasado previamente era solo el de ser píxeles cercanos al suelo. Al quedarnos con el grupo mayor de tonalidad similar, tenemos muchas posibilidades de que todos esos píxeles sean suelo transitable que tengan un color similar. Esto parte también de la idea de que la mayor parte del suelo será transitable y habrá un mayor número de píxeles transitables que de obstáculos en las imagenes con las que trabajemos.

  Cabe señalar que en un primer momento el algoritmo se implementó sin el clustering y el resultado era claramente peor. Resulta clave para el proceso, y más teniendo en cuenta como se desarrolla más adelante, el poder eliminar en un primer momento cualquier píxel perteneciente a obstáculos del conjunto de píxeles que consideraremos suelo transitable en un primer momento.

- En segundo lugar, y una vez conocido el primer conjunto de píxeles etiquetados como suelo transitable, llega el momento de expandir el etiquetado hacia el resto de píxeles pertenecientes a la ROI de la forma que se explica a continuación.

  Para cada uno de los píxeles etiquetados como suelo se chequea a sus cuatro vecinos, es decir, los píxeles colindantes tanto horizontal como verticalmente. Sin embargo esto solo se llevará a cabo siempre y cuando los vecinos pertenezcan a la ROI.

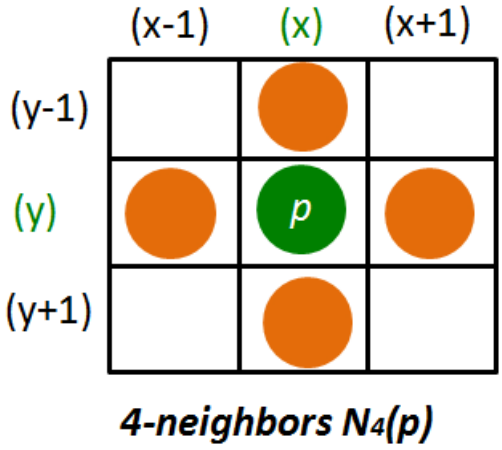

Figura 17: vecinos a 4

El chequeo consiste en calcular la Delta E entre el pixel considerado suelo y su vecino. Si el valor de la Delta E es inferior a un cierto umbral (th, parametro de entrada variable) al haber una diferencia de color que consideramos lo suficientemente pequeña, ese pixel vecino se etiquetará también como suelo. Por tanto se procederá a examinar también a los vecinos de ese nuevo pixel etiquetado. De este modo, el algoritmo continuará expandiendo la zona etiquetada de pixel a pixel vecino mientras las transiciones de color sean suaves.

$$DeltaE = \sqrt{(L_1^* - L_2^*)^2 + (a_1^* - a_2^*)^2 + (b_1^* - b_2^*)^2} \leq th \tag{14}$$

En el algoritmo anterior se tomaba un cierto color (con su valor medio y su desviación) como referencia del color que debía tener un pixel para ser considerado suelo navegable. Los parámetros de color de cada pixel de la ROI se comparaban con él para determinar si era lo suficientemente cercano y por tanto se podía considerar suelo transitable o no. Eso no permitía captar bien los tramos en los que las variaciones de color se iban produciendo paulatinamente hasta llegar a un color muy distinto del de partida. Esto es lo que ocurre, por ejemplo, a lo largo de la zona central de los pasillos en las imágenes que venimos usando de muestra, donde el color del suelo se va gradualmente tornando más blanco. Esto es fruto de la iluminación artificial que se da en interiores.

Un algoritmo como este que toma como referencia las transiciones de color que se producen progresivamente, pixel a pixel, muestra un comportamiento mucho más adecuado a la hora analizar los parámetros de color en interiores, mejorando notablemente los resultados en el etiquetado. El hecho de usar la Delta E para evaluar las variaciones de color es también un avance que influye en la mejora del resultado, pues cuantifica la distancia entre colores de una forma mucho más uniforme y más cercana a la percepción del ojo humano.

5.2.5. Generación de mapa

Como último paso queda generar un mapa visto desde arriba, lo que se conoce como vista de pájaro. Para ello es necesario conocer previamente las distancias

reales en 3D de los píxeles etiquetados como suelo.

Partiendo del modelo pinhole y conociendo los parámetros de calibración de la lente y la distancia focal, se puede conocer la distancia real haciendo uso de las ecuaciones (1), (2) y (3) (sección 3.1). Partiendo de los datos (u,v) del pixel se desea calcular su posición (x,y,z) en 3D, con lo que tenemos un sistema indeterminado.

Para obtener un dato más que nos falta para hacer el sistema determinado, asumimos que la distancia entre la cámara y el plano del suelo es Y, según los ejes del plano que se pueden ver en la figura 18:

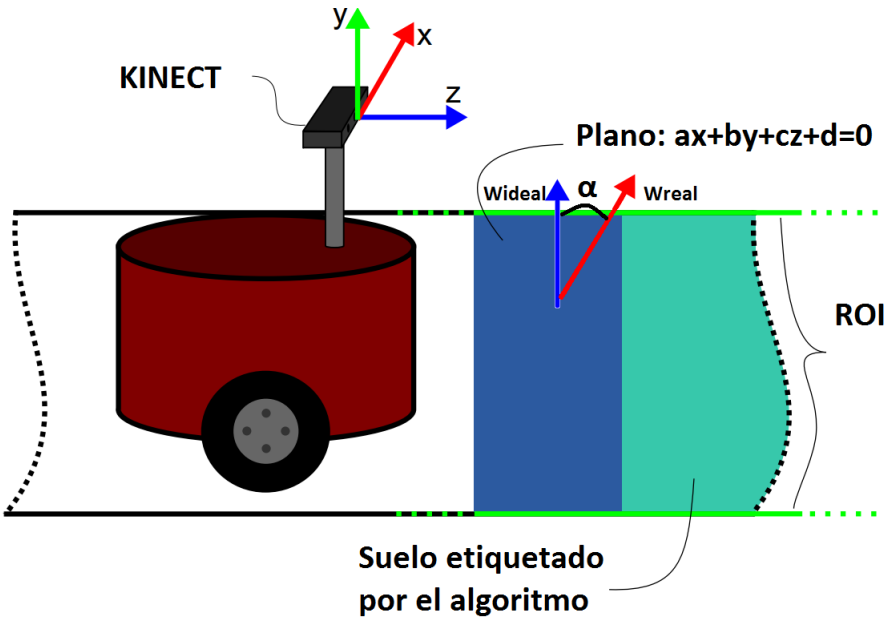

Figura 18: Representación espacial kinect

Este paso sería suficiente si los ejes de la kinect fuesen realmente paralelos al suelo y su eje Y fuese, por tanto, perpendicular al plano del suelo. Pero esto no ocurre, el eje Y está ligeramente rotado y no es perpendicular. Se encuentra rotado respecto al eje Z que es paralelo al plano del suelo. Para solucionar este inconveniente, es necesario calcular el ángulo que forma el eje Y respecto del vector normal al plano del suelo para posteriormente rotar los píxeles respecto del eje Z en el sentido de las agujas del reloj.

Sabemos que el vector normal al plano del suelo real y el ideal que buscamos serán:

$$w = \begin{vmatrix} a \\ b \\ c \end{vmatrix} \quad w_{ideal} = \begin{vmatrix} 0 \\ -1 \\ 0 \end{vmatrix} \tag{15}$$

Se puede calcular el ángulo que forman ambos vectores de la siguiente manera:

$$cos\theta = \frac{w\ w_{ideal}}{|w||w_{ideal}|} \tag{16}$$

Y posteriormente se pueden rotar los píxeles usando la siguiente matriz de rotación:

$$R_z(-\theta) = \begin{vmatrix} cos(-\theta) & -sin(-\theta) & 0 \\ sin(-\theta) & cos(-\theta) & 0 \\ 0 & 0 & 1 \end{vmatrix} \tag{17}$$

De este modo conseguiremos que los píxeles del plano del suelo ya si sean perpendiculares al eje Y de la cámara kinect y podamos hacer la suposición anterior para obtener un sistema determinado y poder calcular las posiciones en 3D en el mundo real de cada uno de los píxeles del suelo transitable que aparecen en la imagen.

Por mostrar un ejemplo, el resultado es el de la figura 19:

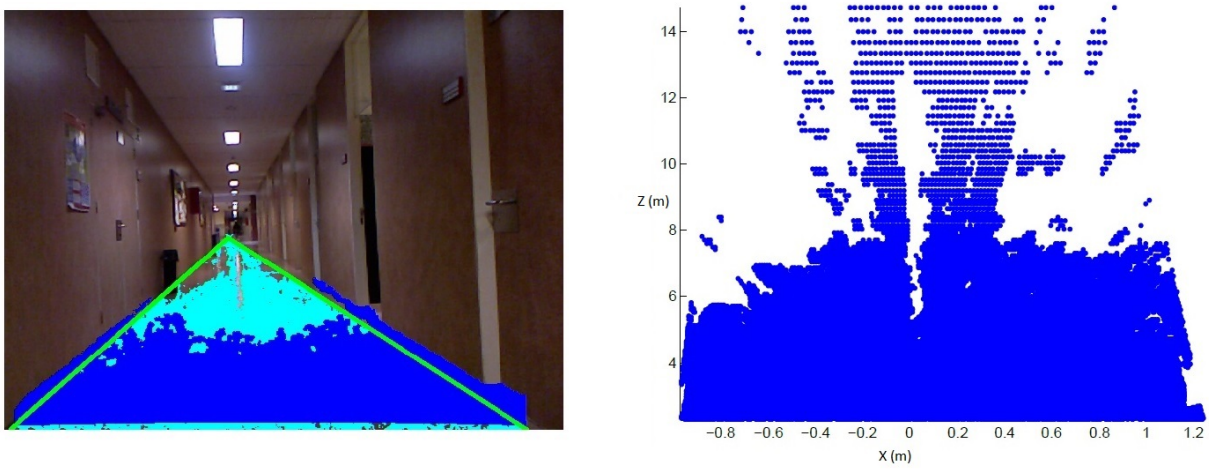

Figura 19: Ejemplo generación de mapa

6. Resultados experimentales

A continuación se mostrarán los resultados de las pruebas realizadas. En primer lugar se expondrán las pruebas llevadas a cabo en el algoritmo anterior, para conocer de donde partimos, y posteriormente se incluirán las pruebas y conclusiones sobre el algoritmo propuesto en este proyecto.

Para poder analizar los resultados haremos uso de tres parámetros que nos permitirán cuantificar el desempeño del algoritmo de forma objetiva: *precision*, *recall* y *F-score*. *Precision* es el índice de píxeles etiquetados como suelo correctamente, *recall* representa la sensibilidad del algoritmo para etiquetar correctamente los píxeles del suelo, y *F-score* es una media armónica que combina ambos parámetros anteriores para obtener un resultado global y por tanto es el mejor parámetro para comparar que etiquetado es el más eficaz. Su formula es la siguiente:

$$Precision = \frac{t_p}{t_p + f_p} \tag{18}$$

$$Recall = \frac{t_p}{t_p + f_n} \tag{19}$$

$$F = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{20}$$

Donde t_p significa *true positives*, f_p *false positives* y f_n *false negatives*.

6.1. Algoritmo anteior

Se han llevado a cabo una serie de pruebas con el algoritmo anterior en tres espacios de color (RGB, L*ab, y HSV) en los tres pasillos distintos que se pueden ver en las figuras 20 21 22. En estos tres entornos se han añadido obstáculos y se ha determinado el numero de píxeles etiquetados correcta e incorrectamente para cada uno de los diez valores distintos dados al único parámetro de entrada del programa: la *mahalanobis distance*. Es el equivalente al threshold de la Delta E de nuestro algoritmo propuesto en el sentido de que marca el umbral para el que una diferencia de color mayor ya no será considerada suelo navegable. Sin embargo, hay que tener presente que los método ultilizado para la comparación son totalmente distintos tal y como se explicó anteriormente (sección 5.2.4).

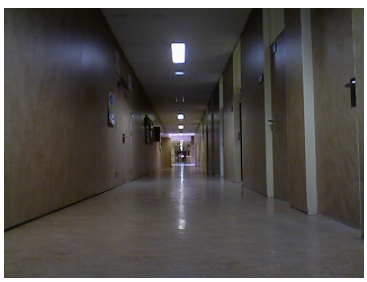

Figura 20: Pasillo 1

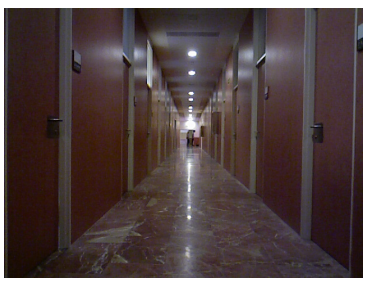

Figura 21: Pasillo 2

Figura 22: Pasillo 2

De este modo, podemos mostrar a continuación cómo varían *precision* frente a *recall* y *F-score* frente a *mahalanobis distance* que es lo que realmente nos ayudará a determinar cuál es la configuración más adecuada de este algoritmo en cada uno de los tres entornos.

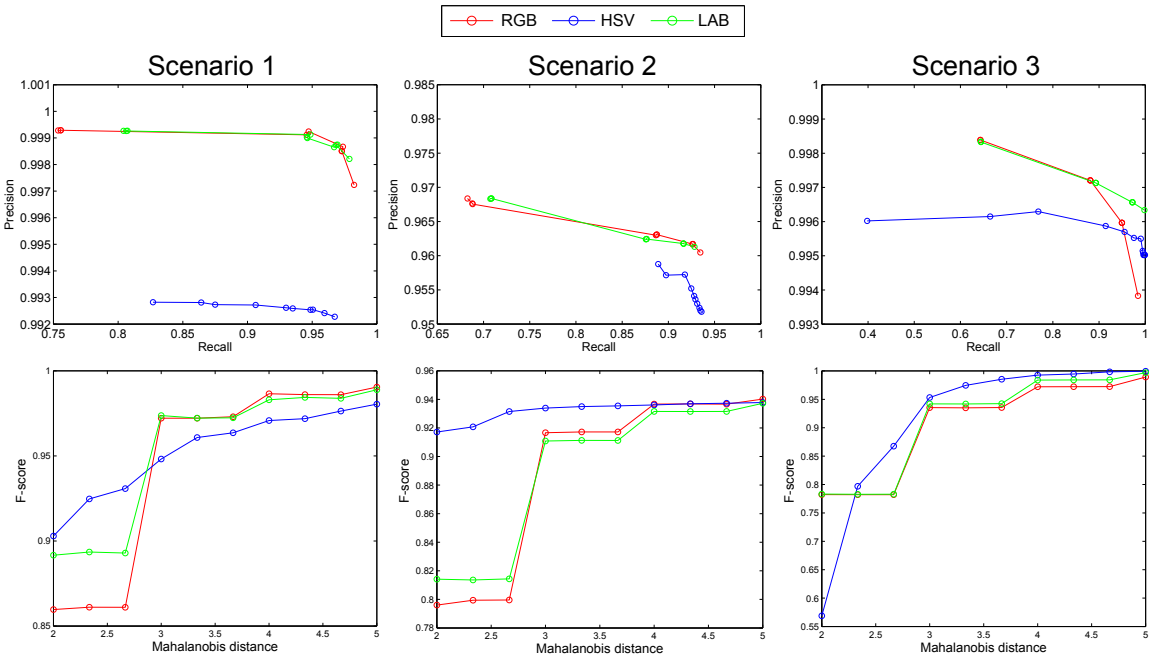

Figura 23: Resultados en los tres escenarios con obstáculos. En HSV la *mahalanobis distance* está multiplicada por 10

Precision es muy similar en RGB y en Lab, en ambos casos superando a HSV. Esto quiere decir que en HSV se etiquetan más obstáculos como suelo navegable. Sin embargo, en HSV el valor de *recall* es superior. En cuanto a *F-score*, para valores más restrictivos de la distancia de mahalanobis se comporta mejor en HSV, aunque con RGB o Lab se consigue un desempeño más equilibrado al comportarse mejor con umbrales menos restrictivos.

6.2. Algoritmo propuesto

En este apartado se mostrarán los resultados de las distintas pruebas llevadas a cabo con el algoritmo propuesto. Para cada uno de los tres pasillos que se muestran en las figuras 24, 25 y 26 se han realizado una serie de pruebas modificando los dos parámetros de entrada que posee el algoritmo para encontrar su configuración más adecuada.

6.2.1. Parámetros de entrada

Los dos parámetros de entrada que modificaremos son:

- k: Número de agrupaciones que se relizan en el clustering. Cuanto mayor sea su valor, menor será el número de puntos iniciales de los que partiremos para

Figura 24: Pasillo A

Figura 25: Pasillo B

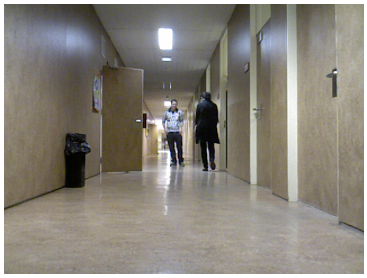

Figura 26: Pasillo C

comparar los píxeles vecinos.

Si su valor es 1, no habrá en realidad un proceso de clustering, pues solo habrá un grupo que incluirá a todos los puntos. Estos píxeles son los que aparecen en color verde y que darán origen por comparación con sus vecinos a los pintados en azul. Si en el pasillo A (con un valor de th = 7) tomamos k = 1 se aprecia como tenemos un gran número de píxeles en color verde:

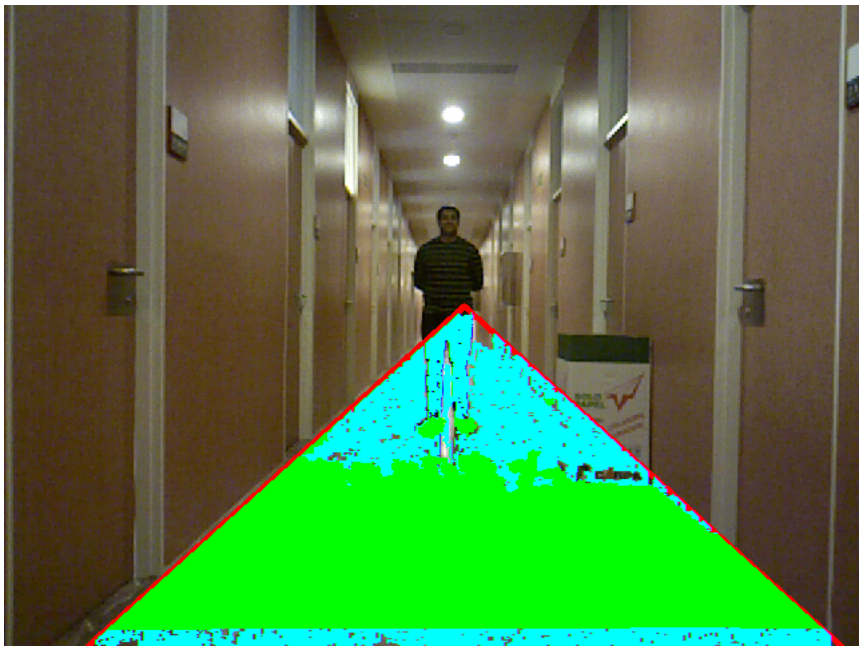

Figura 27: Pasillo A con k=1

Al no estar realizando el clustering, explicado en la sección 5.2.4, se están tomando como píxeles de partida, para la posterior comparación de color, a algunos píxeles pertenecientes a obstáculos, como son las piernas de la persona que aparece en la imagen. Por tanto, es inevitable que por comparación de color se acabe etiquetando a gran parte de estos obstáculos (píxeles azules). Para evitar esto, conviene aumentar el valor de k hasta que

se consiga no etiquetar obstáculos como parte de los píxeles de partida (color verde). Tomando k=4 se consigue lo anteriormente expuesto, como se puede ver en la imagen 28:

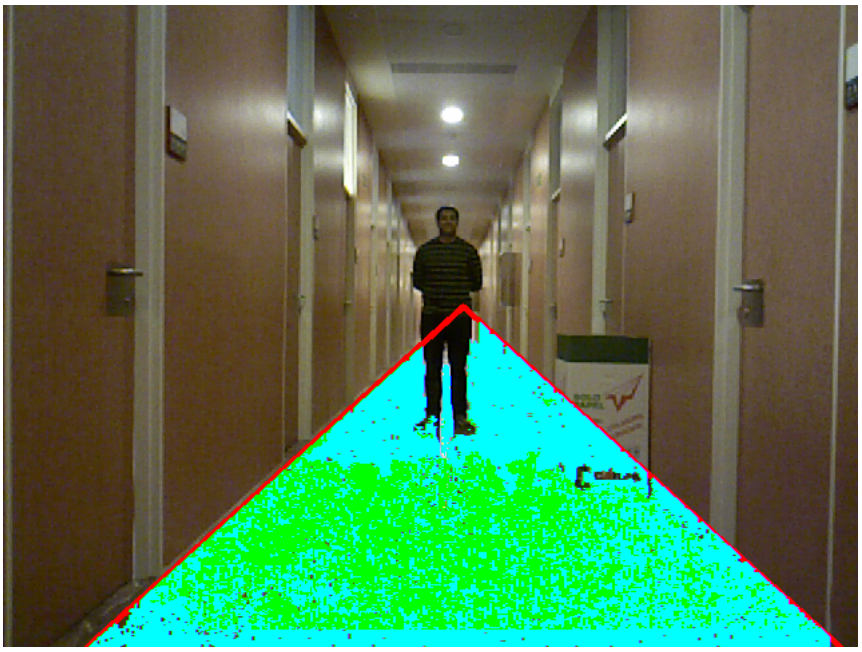

Figura 28: Pasillo A con k=4

Este sería el valor óptimo de k para este entorno, ya que si seguimos aumentando la k ya no conseguiremos mejorar los resultados. Lo único que ocurrirá será que dejará de etiquetar incluso algunas zonas de suelo que antes si etiquetaba, y el tiempo de ejecución del algoritmo aumentará notablemente también. Esto se aprecia en la figura 29 para un k = 7.

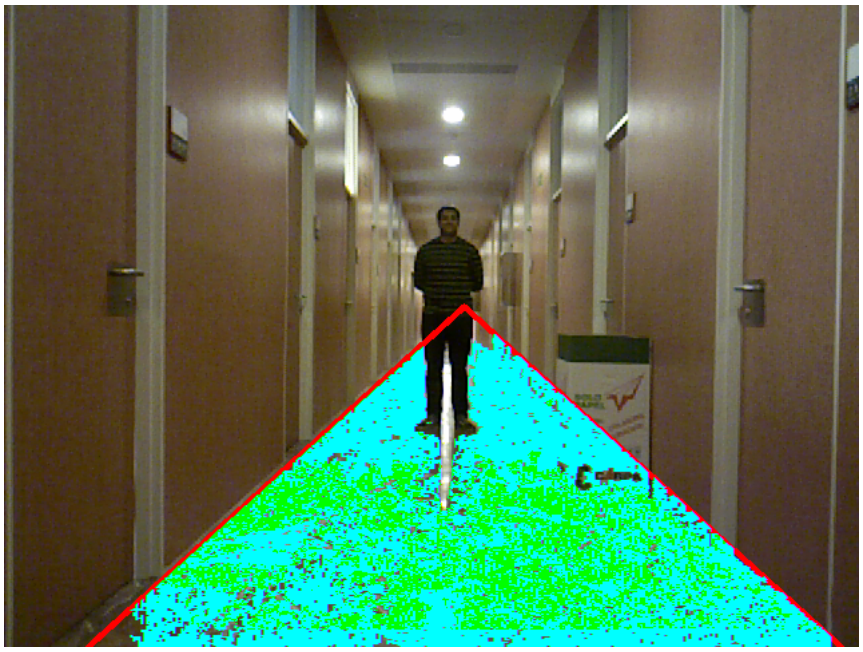

Figura 29: Pasillo A con k=7

Para poder volver a etiquetar esa zona central, que dejó de considerarse suelo al aumentar demasiado el valor de k, será necesario jugar con el valor de th aumentandolo.

Por tanto, podemos concluir que una vez alcanzado el valor de k óptimo que evita el etiquetado inicial de los obstáculos no conviene aumetarlo más.

- th: Es el umbral (*threshold* en inglés) que tendrá la Delta E. Para un valor superior a ese umbral se considerará que la distancia entre colores es demasiado grande, y por tanto, ese pixel vecino no pertenece al suelo navegable.

  Una vez encontrado el valor de k óptimo en el pasillo C (k=2) vamos a comprobar como influye el threshold que pongamos en la Delta E. Si el valor es igual a 2, solo cumplirán la condición píxeles que tengan practicamente el mismo color que su vecino. Por tanto, no se puede esperar que haya muchos más píxeles etiquetados que los que lo están de inicio (los verdes).

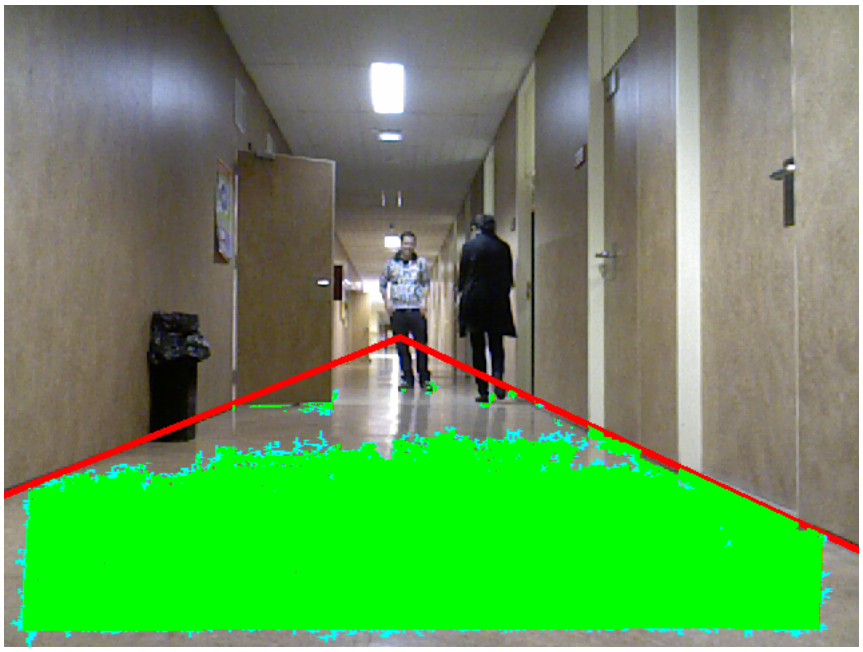

Figura 30: Pasillo C con th=2

Se aprecia que apenas hay píxeles en azul, que son los que han cumplido el requisito de la divergencia de color. Aumentando el th hasta un valor de 7 mejora mucho el etiquetado.

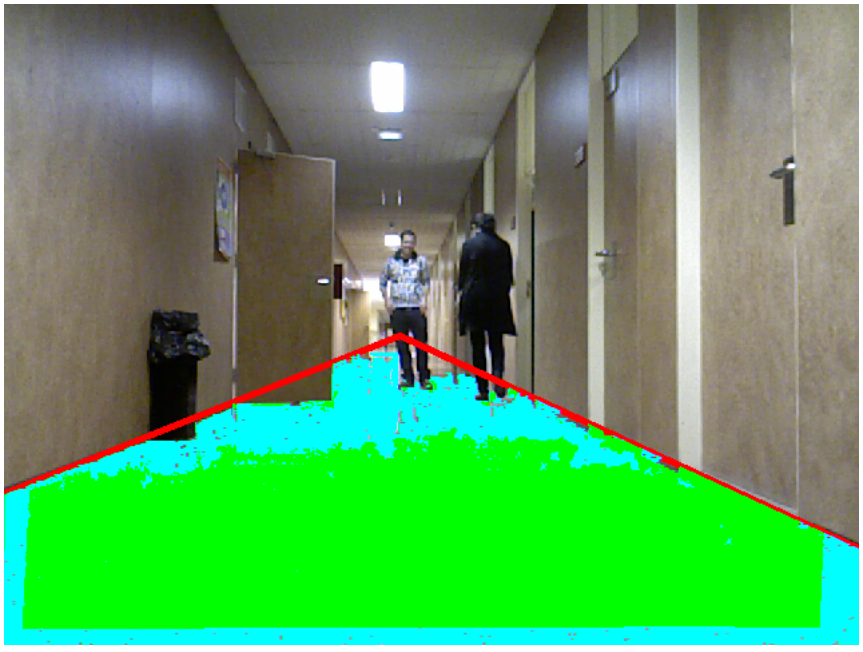

Figura 31: Pasillo C con th=7

Sin embargo, existe también un valor óptimo de th que consiga etiquetar lo máximo posible del suelo navegable sin llegar a etiquetar de forma indiscriminada los obstáculos también. Como ocurre en la figura 32 con un th=15.

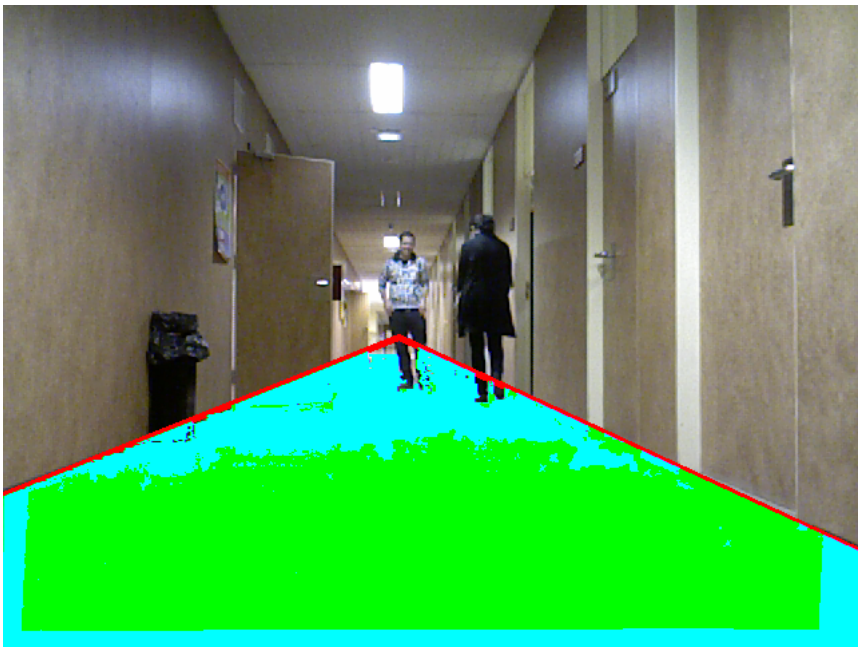

Figura 32: Pasillo C con th=2

En este último caso se admite una diferencia de color tan grande que es imposible evitar que la papelera, la puerta o las piernas sean consideradas suelo.

6.2.2. Pasillo A

Pasamos ahora a mostrar el resultado de las pruebas experimentales relizadas con la nube de puntos captada por la cámara kinect en el primero de los tres pasillos.

Tras realizar una serie de pruebas tomando un valor fijo de k y variando la th y viceversa hemos estimado los píxeles etiquetados de forma correcta e incorrecta para poder computar el valor de *precision* y de *recall* en cada prueba. El resultado se muestra en la figura 33

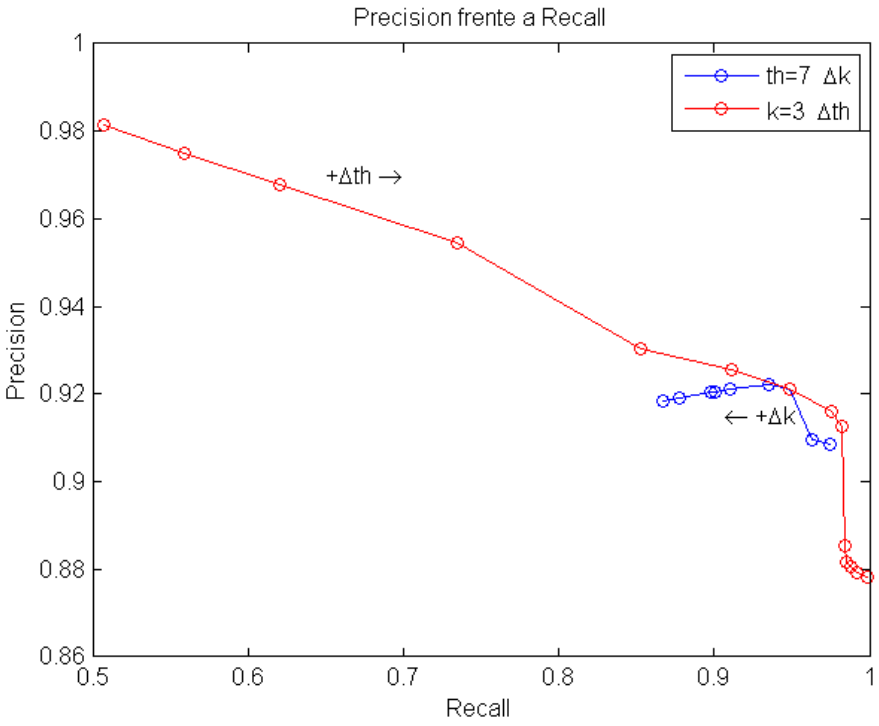

Figura 33: Precision frente a Recall en pasillo A

Como era de esperar, al mejorar *recall* empeora *precision*. Esto se debe a que al relajar las condiciones de etiquetado, la proporción de píxeles del suelo navegable que son etiquetados aumenta, pues aumenta el número de píxeles etiquetados en general. Es decir, aumentan los *true positives*, lo cual hace que mejore recall. Sin embargo, esto ocurre pagando el precio de aumentar también los *false positives* al etiquetar como suelo a una mayor parte de los obstáculos, lo que hace que disminuya la precisión del etiquetado.

Para poder determinar cuál es el valor óptimo de los parámetros de entrada para lograr el mejor desempeño posible de nuestro algoritmo en este primer entorno, se han realizado un conjunto de pruebas con distintos valores de k y de th para determinar la *F-score* en cada caso. De este modo, se muestra a continuación en la figura 34 la superficie en 3D que describe *F-score* al ser representada frente a k y th.

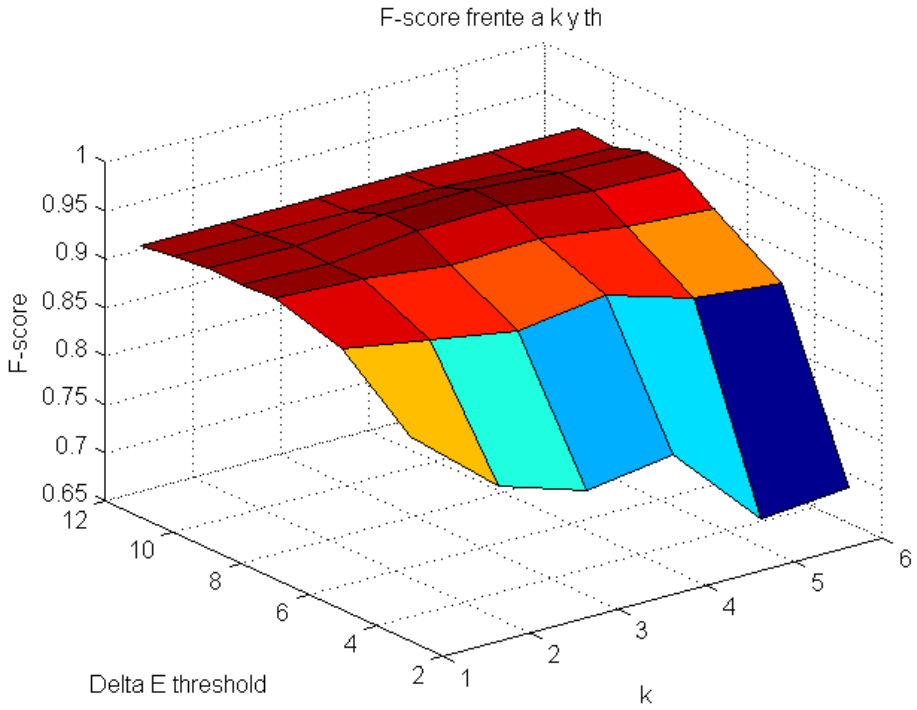

Figura 34: F-score frente a k y th

El valor más cercano a uno de *F-score* se alcanza para el caso k=4 y th=9. En este entorno esa sería la configuración óptima para un buen etiquetado del suelo navegable. En este caso el etiquetado queda tal como muestra la figura 35.

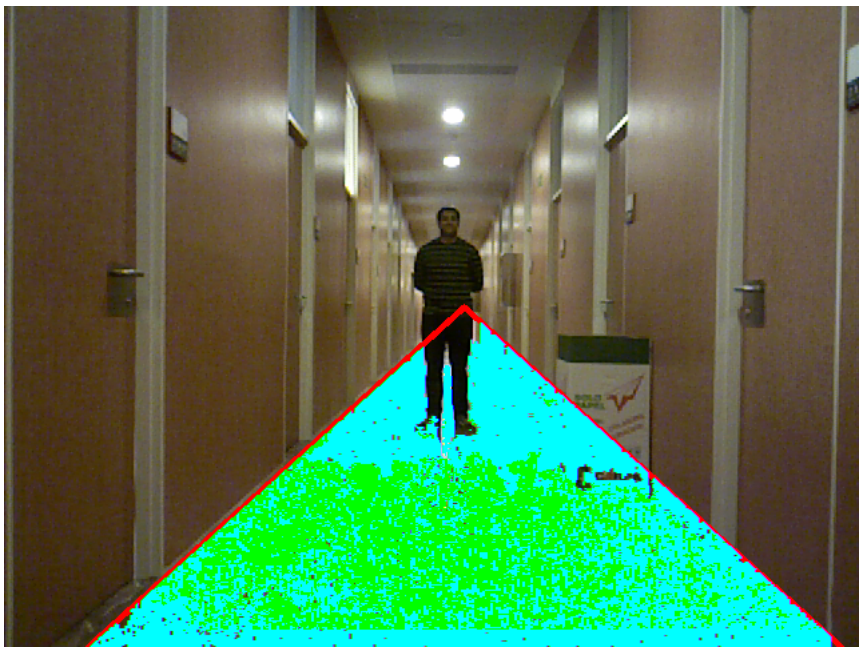

Figura 35: Etiquetado óptimo pasillo A

Sin embargo, cabe destacar que según las circunstancias podría convenir reducir el número de clusters y tomar un valor de k inferior. El valor de k es uno de los

principales responsables del tiempo de ejecución del algoritmo y por tanto en ciertas circunstancias podría interesar reducir algo su valor, aunque se reduzca ligeramente *F-score*, si así se consigue una mayor velocidad en la ejecución.

6.2.3. Pasillo B

Este entorno tiene la característica de no poseer obstáculos dentro de la ROI. Por tanto, cuanto más se relajen las condiciones de etiquetado y más píxeles se etiqueten mejor será el resultado. Esto quiere decir que al no haber riesgo de *false positives*, *precision* será siempre igual a uno y *recall* y *F-score* mejorarán cuanto más aumente th y más disminuya k. Esto se puede apreciar en las figuras 36 37.

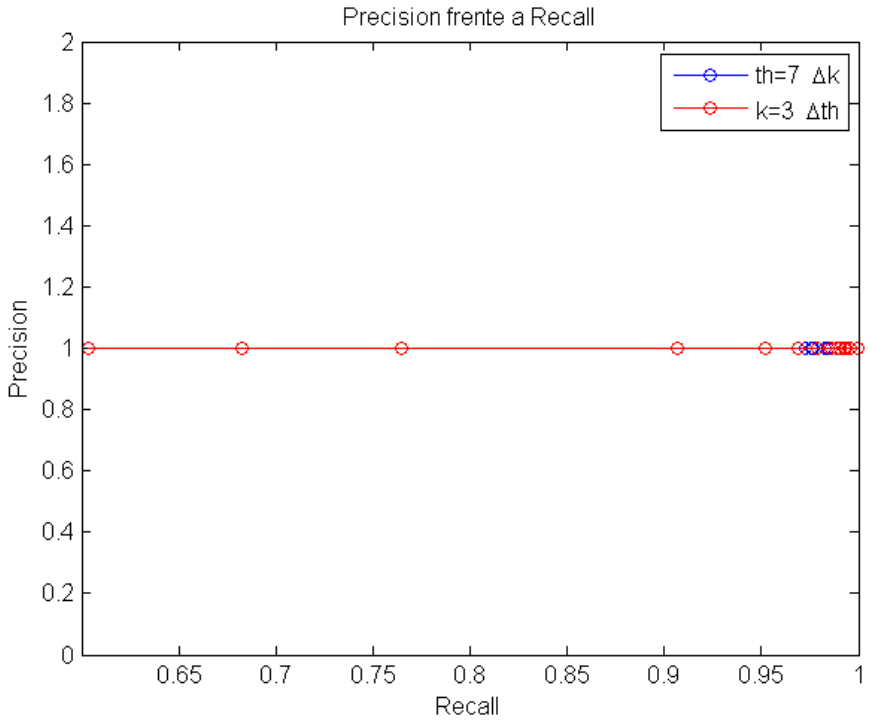

Figura 36: Precision frente a Recall en pasillo B

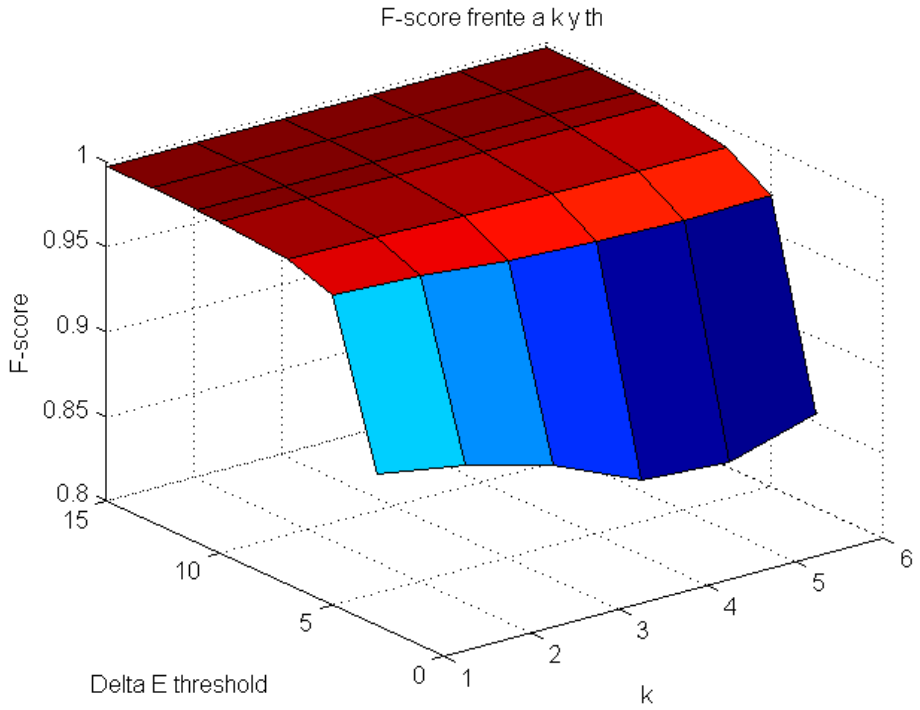

Figura 37: F-score frente a k y th

En este caso especial que no posee obstáculos de ningún tipo, la mejor configuración sería la más permisiva. Si bien hay que tener en cuenta que esto solo ocurrirá en estas circunstancias particulares, para el resto de casos estos parámetros de entrada estarían muy lejos de ser los mejores.

Para un caso así, tomando por ejemplo k=1 (no realiza clustering) y th=15 el etiquetado es el que se muestra en la figura 38.

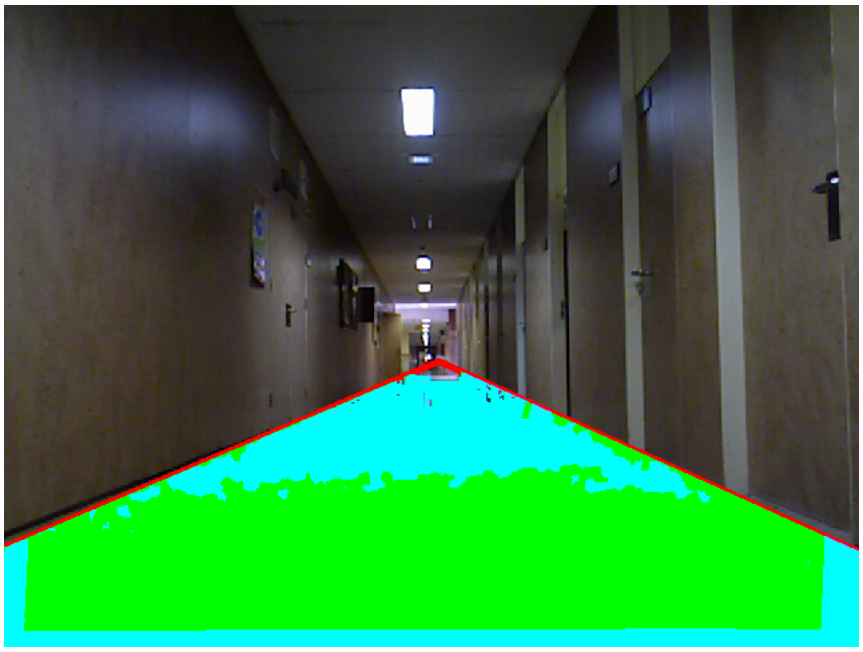

Figura 38: Pasillo B con k=1 y th=15

6.2.4. Pasillo C

Estamos de nuevo ante un pasillo con diversos obstáculos, y por tanto el comportamiento del algoritmo será cercano al que tuvo con el pasillo A.

En las figuras 39 y 40 se muestran las dos gráficas que venimos utilizando.

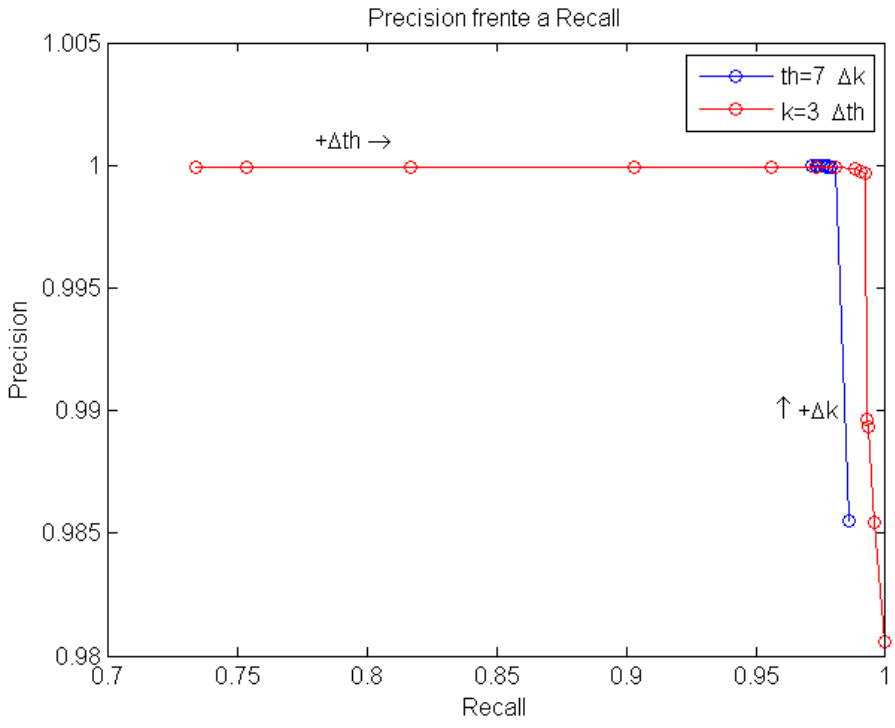

Figura 39: Precision frente a Recall en pasillo C

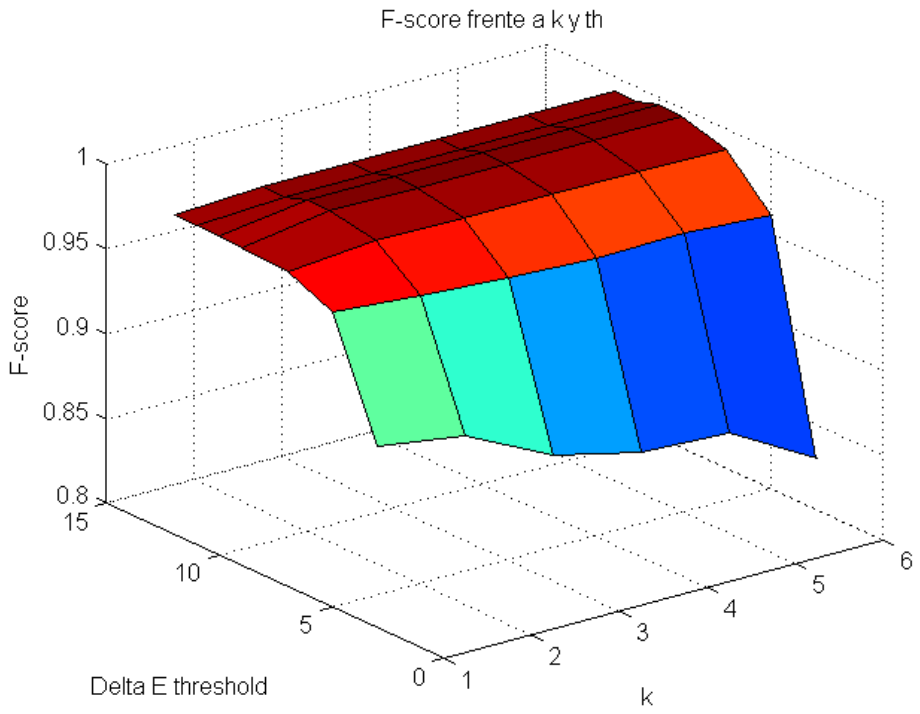

Figura 40: F-score frente a k y th

Igual que ocurría en el pasillo A, vemos como al principio para valores muy bajos de th la *F-score* se encuentra bastante alejada del 1. A medida que se aumenta la th mejora mucho el etiquetado, como nos indica la gran pendiente que posee la superficie en los primeros valores de th. El mejor resultado se obtiene con k=2 y th=10, donde el pico alcanza un *F-score*=0.9959. Se muestra esta situación en la figura 41

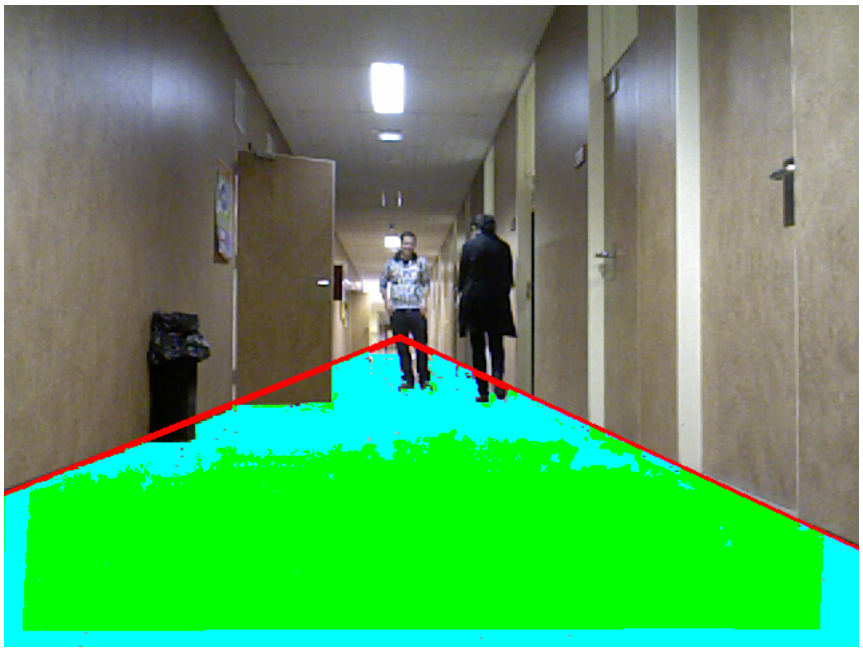

Figura 41: Etiquetado óptimo pasillo C

6.3. Análisis de tiempos

El propósito de este apartado es evaluar uno de los puntos claves de cualquier algoritmo que pretenda ser ejecutado en tiempo real de forma continua, como es nuestro caso, para cada frame que capte la cámara kinect.

Hay que señalar que los tiempos de ejecución dependen del procesador en el que se ejecute el algoritmo. En este caso las pruebas se han realizado valiendose de un procesador Intel i7-2670QM con una velocidad de reloj de 2,20GHz en cada uno de sus núcleos.

En primer lugar vamos a valorar la incidencia del threshold que fijamos en el tiempo de ejecución de nuestro algoritmo.

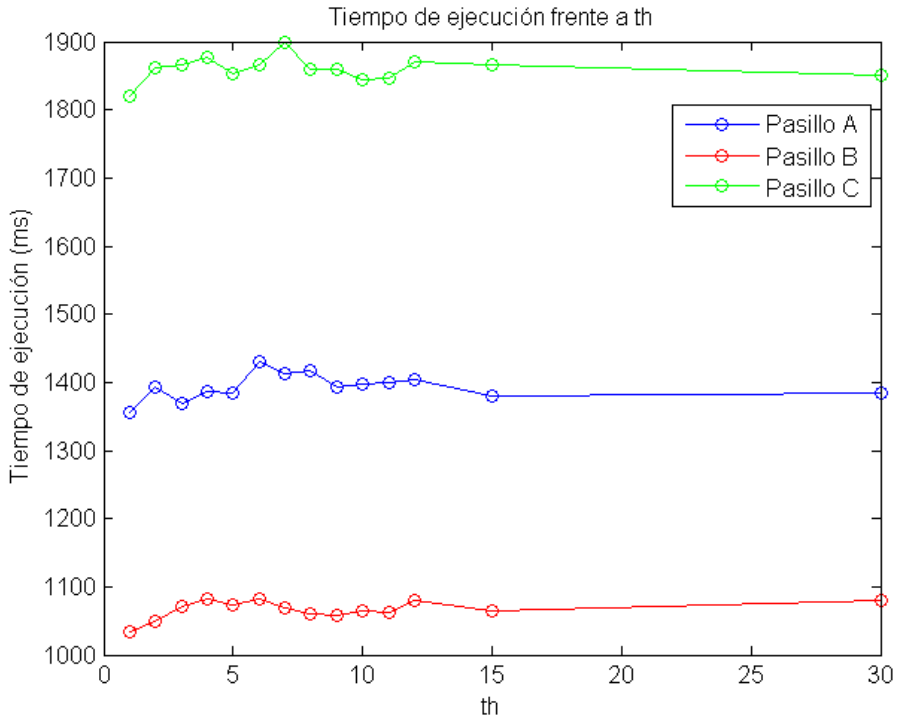

Figura 42: tiempo frente a th

Como se aprecia en la gráfica de la figura 42, el valor de th no influye realmente en el tiempo que tarda en ejecutarse el programa. El hecho de tener que etiquetar más o menos píxeles a la hora de hacer la comparación de color parece ser que no tiene incidencia en el tiempo total. Sin embargo, tal como vemos en la figura 43, el número de agrupaciones según tonalidad que tenga que realizar durante el proceso de clustering es lo que realmente marca la diferencia.

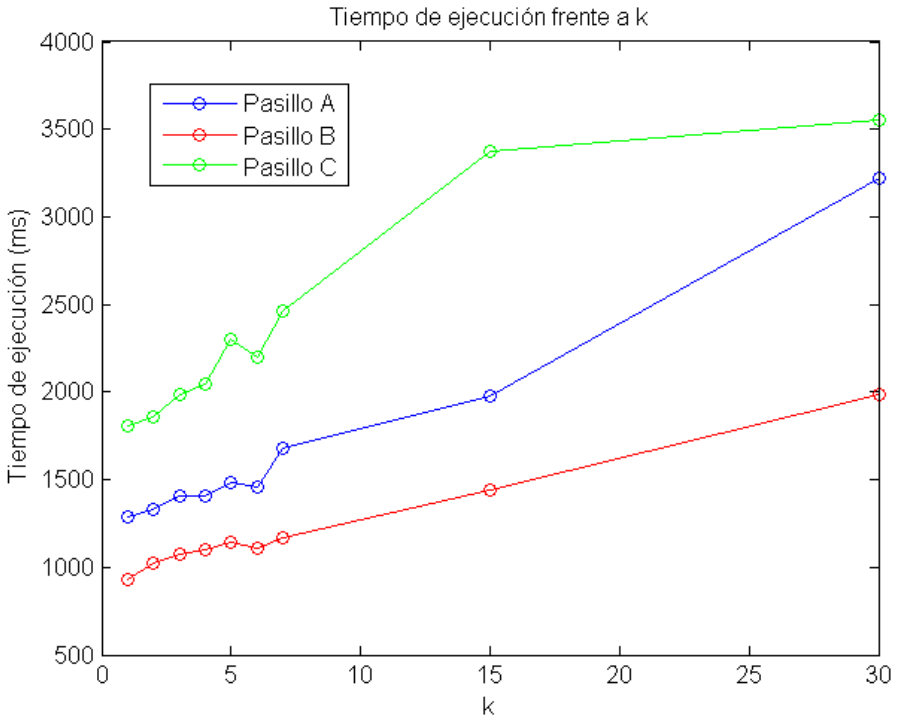

Figura 43: tiempo frente a k

También se aprecia que el tipo de entorno influye decisivamente. A mayor complejidad del entorno a analizar, mayor tiempo de procesado requerirá.

7. Conclusiones y trabajo futuro

Aunque la mejora del algoritmo resulta evidente respecto al anterior, su aplicación a entornos de interior sigue siendo muy mejorable.

En primer lugar tenemos las limitaciones propias del hardware utilizado. La tecnología basada en infrarrojos que usa la cámara kinect impide su funcionamiento en exteriores debido a las interferencias con la luz solar. También el reducido rango de actuación de los sensores laser son una limitación intrínseca de la cámara.

La limitación principal del algoritmo es que está diseñado para uso exclusivo en pasillos. Entornos donde haya siempre dos paredes, techo y suelo. Y además el pasillo debe ser de anchura no variable, pues si no es así el cálculo de la ROI no se corresponderá con la realidad. Por tanto, la mejora del algoritmo se podría encaminar hacia una ampliación de los distintos entornos en los que se pueda caracterizar correctamente la región de interés atendiendo a sus características geométricas. Así como introducir otros parámetros adicionales, como la textura, a la hora de comparar entre si los píxeles para determinar si cumplen las condiciones para ser considerados parte del suelo navegable.

Finalmente, cabe señalar que el tiempo de ejecución sigue siendo un inconveniente

importante a la hora del uso del algoritmo en tiempo real. Convendría reducir los tiempos buscando maneras de lograr el mismo resultado sin realizar el clustering, realizando un proceso alternativo para evitar etiquetar en un primer momento los píxeles de los obstáculos pero que consuma mucho menos tiempo.

8. Presupuesto

En el presente anexo se calculará el coste del valor del proyecto presentado a lo largo del documento, detallando cada coste por separado.

8.1. Costes de personal

Los costes de personal se han separado en dos tipos diferentes: tipo de personal y apartado del proyecto. La primera división obedece a la diferencia salarial entre el ingeniero y el director del proyecto, mientras que la segunda división detalla las horas empleadas en cada sección del proyecto, dada la magnitud de éste.

Para la división del proyecto en apartados se ha seguido la siguiente estructura:

- Planteamiento. Incluye la familiarización con las librerías necesarias, el estudio del algoritmo inicial y el diseño general del proyecto.
- Desarrollo. Incluye varios apartados, entre los que se encuentran el desarrollo del algoritmo y la investigación requerida para las distintas partes que lo componen.
- Pruebas. Incluye todas las pruebas realizadas para comprobar el funcionamiento del algoritmo.
- Redacción de la memoria. Incluye tanto la redacción como la corrección de la misma.

		Director de proyecto	Ingeniero
3*Planteamiento	Familiarización (h)	0	30
	Estudio del algoritmo (h)	10	20
	Diseño (h)	0	10
2*Desarrollo	Desarrollo del algoritmo (h)	5	300
	Investigación (h)	5	120
2*Pruebas	Pruebas en pasillo inicial (h)	10	80
	Pruebas en distintos pasillos (h)	1	5
2*Memoria	Redacción (h)	0	60
	Corrección (h)	15	15
4*Total	Total (h)	46	640
	Salario/hora	50	30
	Salario total ()	2300	19200
	Total ()		21500

Tabla 2: Costes desglosados de personal

8.2. Costes materiales

Los costes materiales incluye todo lo relacionado con los costes de hardware y software y recursos de oficina. Dentro del hardware se incluye un ordenador portátil de gama alta en el que poder realizar la búsqueda de información y todas las tareas relacionadas con el desarrollo del algoritmo, además de la cámara Kinect con la que tomar la información y realizar las pruebas. Ambos dispositivos tienen un plazo de amortización de tres años, y se han tenido en cuenta los 4 meses de duración del proyecto. En cuanto al software, se han utilizado alternativas libres y gratuitas tanto en el sistema operativo como en los programas y librerías, por lo que el coste es nulo. Finalmente, respecto a los recursos de oficina únicamente se tiene en cuenta el coste de acceso de Internet.

			Coste de proyecto
2*Hardware	Ordenador	900 (total)	100
	Kinect	100 (total)	11.11
Software	Software	0	0
Oficina	Internet	30 (/mes)	120
Total ()			231.11

Tabla 3: Costes desglosados de material

8.3. Total

Sumando los apartados anteriores y añadiendo tanto costes indirectos (veinte por ciento) como IVA (veintiuno por ciento), el precio total del proyecto asciende a *treinta y un mil quinientos cincuenta y tres euros y cincuenta y siete céntimos*

	Coste()
Personal	21500
Materiales	231.11
Costes indirectos (20 %)	4346.22
Subtotal	26077.33
IVA (21 %)	5476.24
Total ()	31553.57

Tabla 4: Costes desglosados totales

Leganés, 14 de Septiembre de 2014

El ingeniero

Referencias

[1] Pardeiro Blanco, Jose. En *Modelado automático tridimensional del entorno y extracción de características*

[2] Kinect camera color stream: `http://msdn.microsoft.com/en-us/library/jj131027.aspx` (1 de septiembre de 2014)

[3] Unix (2001, 19 de diciembre). En *Wikipedia, la enciclopedia libre*. Disponible en: `http://es.wikipedia.org/wiki/Unix` (1 de septiembre de 2014).

[4] Wang Yan-ping, Ma Hui-quan. En *Design of autonomous mobile robot navigation system*. Disponible en: `http://ieeexplore.ieee.org.strauss.uc3m.es:8080/stamp/stamp.jsp?tp=&arnumber=6013120` (1 de septiembre de 2014).

[5] F. J. Jiménez, J.C. Moreno, R. González, F. Rodríguez Díaz. En *Sistema de visión de apoyo a la navegación de un robot móvil en invernaderos*. Disponible en: `http://www.ceautomatica.es/old/actividades/jornadas/XXIX/pdf/279.pdf` (1 de septiembre de 2014).

[6] Ma Ling, Wang Jianming, Zhang Bo, Wang Shegbei. En *Automatic Floor Segmentation for indoor robot navigation*. Disponible en: `http://ieeexplore.ieee.org.strauss.uc3m.es:8080/stamp/stamp.jsp?tp=&arnumber=5555399` (1 de septiembre de 2014).