TRABAJO FIN DE GRADO:

MODELADO AUTOMÁTICO TRIDIMENSIONAL DEL ENTORNO Y EXTRACCIÓN DE CARACTERÍSTICAS

Autor: Jose Pardeiro Blanco
Tutor: Javier Victorio Gómez González

Índice

Resumen **6**

Abstract **7**

1. Introducción **8**
1.1. Motivación . . . 8
1.2. Objetivos . . . 8
1.3. Estructura . . . 9

2. Estado del arte **10**
2.1. Dispositivos que interactúan con el usuario . . . 10
2.2. Navegación de robots móviles . . . 11
2.3. Visión artificial . . . 12

3. Fundamentos teóricos **14**
3.1. Óptica . . . 14
3.2. Espacio de color RGB . . . 16
3.3. Estudio de planos . . . 17
3.4. Estudio de rectas . . . 19
3.4.1. Ecuación de la recta que pasa por dos puntos . . . 19
3.4.2. Distancia de un punto a una recta . . . 20
3.5. Análisis estadístico multivariante . . . 21
3.5.1. Cluster k-means . . . 21
3.5.2. Distancia de Mahalanobis . . . 23
3.6. Distribución gaussiana . . . 24

4. Tecnologías asociadas **26**
4.1. Kinect . . . 26
4.2. OpenNI . . . 27
4.3. C++ . . . 28
4.4. Herramientas de desarrollo . . . 28
4.4.1. Colección de compiladores GNU . . . 29
4.4.2. CMake . . . 30
4.5. Licencia GPL . . . 30
4.6. Bibliotecas . . . 31
4.6.1. Point Cloud Library . . . 32
4.6.2. OpenCV . . . 34

5. Algoritmo propuesto para segmentación y etiquetado de zonas navegables **36**
5.1. Introducción . . . 36
5.2. Partes del algoritmo . . . 38
5.3. Filtrado de la nube de puntos . . . 39
5.4. Segmentación de la nube de puntos . . . 40
5.4.1. Píxeles pertenecientes al plano del suelo . . . 45
5.4.2. Extracción de parámetros de color . . . 48

5.5. Análisis de parámetros de color 49
5.5.1. Cálculo de matrices de covarianza 54
5.5.2. Aprendizaje de gaussianas 55
5.6. Región de interés 56
5.7. Etiquetado del suelo 60
5.8. Generación de mapa 63

6. Funcionamiento del algoritmo con la cámara Kinect 67
6.1. Problemas que surgieron en el funcionamiento dinámico 67
6.2. Pruebas en funcionamiento dinámico 67

7. Conclusiones y trabajo futuro 72

A. Presupuesto 73
A.1. Costes de personal 73
A.2. Costes materiales 74
A.3. Total 74

Índice de figuras

1. Cámara Kinect . . . 11
2. Ejemplo de modelo pinhole. Imagen cortesía de Wikipedia, la enciclopedia libre . . . 14
3. Ejemplo de tablero A3. Imagen cortesía de Nicolas Burrus . . . 15
4. Ejemplo de tablero A3 calibrado. Imagen cortesía de Nicolas Burrus 16
5. Cubo de color RGB . . . 17
6. Ejemplo de plano . . . 17
7. Ejemplo de representación gráfica de plano . . . 18
8. Paso 1 k-means. Imagen cortesía de Wikipedia, la enciclopedia libre 21
9. Paso 2 k-means. Imagen cortesía de Wikipedia, la enciclopedia libre 22
10. Paso 3 k-means. Imagen cortesía de Wikipedia, la enciclopedia libre 22
11. Paso 4 k-means. Imagen cortesía de Wikipedia, la enciclopedia libre 22
12. Ejemplo de distancia de Mahalanobis . . . 23
13. Ejemplo de distribución gaussiana . . . 24
14. Ejemplo de distribución gaussiana en dos dimensiones . . . 25
15. Lanzamiento de rayos infrarrojos de Kinect . . . 26
16. Nube de puntos de un pasillo . . . 27
17. Logo de OpenNI . . . 27
18. Logo de GCC . . . 29
19. Logo de CMake . . . 30
20. Logo de GPL versión 3 . . . 30
21. Logo de Point Cloud Library . . . 32
22. Logo de OpenCV . . . 34
23. Captura RGB . . . 36
24. Rango de Kinect . . . 37
25. Resultado final ideal . . . 37
26. Nube de puntos original . . . 38
27. Diagrama de flujo general del algoritmo . . . 39
28. Filtrado de la nube de puntos . . . 40
29. Nube original frente a filtrada . . . 40
30. Ejemplo de planos verticales . . . 41
31. Ejemplo de planos horizontales . . . 42
32. Segmentación con filtrado . . . 43
33. Nubes de puntos segmentadas . . . 44
34. Conjunto de planos segmentados . . . 44
35. Diagrama de flujo del proceso de segmentación . . . 45
36. Puntos pertenecientes al suelo con distancia menor a 1 centímetro . 46
37. Puntos pertenecientes al suelo con distancia menor a 5 centímetros 46
38. Puntos pertenecientes al suelo con distancia menor a 10 centímetros 47
39. Diagrama de flujo del proceso de píxeles pertenecientes a un plano . 48
40. Resultados con k=3 . . . 51
41. Resultados con k=5 . . . 52
42. Resultados con k=8 . . . 53
43. Diagrama de flujo del proceso de clustering . . . 54
44. Matriz de covarianzas . . . 54

45. Diagrama de flujo del proceso de aprendizaje 56
46. Resultado del cálculo de rectas de intersección 57
47. Intersección entre el suelo y la pared izquierda 57
48. Intersección entre el suelo y la pared derecha 58
49. Representación de rectas 59
50. Píxeles guardados 60
51. Representación de píxeles analizados con distancia 5 61
52. Representación de píxeles analizados con distancia 2.7 62
53. Representación de píxeles analizados con distancia 1 62
54. Diagrama de flujo del proceso de distancia de Mahalanobis 63
55. Área de triángulos 64
56. Curva de ajuste 65
57. Mapa vista de pájaro sin filtrar 65
58. Mapa vista de pájaro 66
59. Pasillo con distancia de Mahalanobis 2.7 68
60. Pasillo con distancia de Mahalanobis 2 69
61. Mapa vista de pájaro 69
62. Pasillo con distancia de Mahalanobis 2 70
63. Pasillo con distancia de Mahalanobis 2.7 71

Índice de tablas

1. Extracción de parámetros de color 49
2. Agrupaciones y reducción a parámetros gaussianos con k=3 50
3. Agrupaciones y reducción a parámetros gaussianos con k=5 52
4. Agrupaciones y reducción a parámetros gaussianos con k=8 53
5. Píxeles analizados 61
6. Costes desglosados de personal 73
7. Costes desglosados de material 74
8. Costes desglosados totales 74

Resumen

La visión 3D ha sido uno de los campos que más se han visto extendidos en los últimos años. Los últimos avances tecnológicos han propiciado una mayor facilidad y fiabilidad a la hora de obtener la información del entorno.

Hasta ahora, problemas como detectar el suelo o el entorno requerían de un *hardware* muy costoso, además de aparecer numerosas dificultades como errores de sincronización entre el hardware y la unidad de proceso. Actualmente, dichos problemas se encuentran paliados, en parte, gracias a la aparición de cámaras de bajo coste, como por ejemplo el dispositivo Microsoft Kinect, que poseen la capacidad necesaria para desempeñar esas funciones. Por otro lado, han aparecido librerías de desarrollo enfocadas especialmente para el campo de la visión 3D.

En este proyecto se propone un algoritmo capaz de procesar la información recibida de una cámara Kinect con el fin de detectar suelos lisos y transitables. Además, el algoritmo es capaz de superar las limitaciones intrínsecas de Kinect, aumentando el rango de funcionamiento mediante técnicas de aprendizaje. Para ello se realizarán pasos intermedios, que incluyen el análisis, aprendizaje e interpretación de los parámetros de color. Para esta tarea se utilizará tanto el lenguaje de programación C++ como librerías externas tales como OpenCV o PCL, además del apoyo de la herramienta matemática Matlab.

Abstract

Artificial 3D Vision is one of the research fields which has been extended the most in recent years thanks to current technological advances that have led to a greater ease and reliability when obtaining information from the environment.

Up to now, detecting problems like soil or environment required a *hardware* very expensive, besides appearing numerous problems as synchronization errors between the hardware and the processing unit. Currently, these problems are alleviated in part by the emergence of low-cost cameras, such as Microsoft Kinect, which have the capacity to perform these functions. On the other hand, have appeared specially focused development libraries for 3D computer vision.

In this project we propose an algorithm capable of processing the information received from a Kinect camera to detect smooth and walkable zones. We also implement an algorithm able to remove the intrinsic limitations of Kinect, increasing the maximum operating distance. This will involve intermediate steps, including the analysis, learning and interpretation algorithms of color parameters. For this task we use both the C++ programming language and external libraries such as OpenCV or PCL, and support from the Matlab mathematical tool.

1. Introducción

1.1. Motivación

En 2005, un grupo de estudiantes de la Universidad de Stanford, California, participaron en el concurso DARPA Grand Challenge. Dicho concurso se organiza por DARPA, acrónimo de *Defense Advanced Research Projects Agency*, Agencia de Investigación de Proyectos Avanzados de Defensa en castellano. Se trata de una división del Departamento de Defensa de Estados Unidos destinada al desarrollo de nuevas tecnologías con fines militares. En este concurso los participantes tenían que presentar un vehículo capaz de moverse de forma autónoma, sin componente humano, desde un punto de los Estados Unidos a otro.

El grupo ganador hizo público el sistema utilizado[1],el cual está basado en sensores de distancia. Para poder ganar la competición, el equipo buscó que su vehículo alcanzase la mayor velocidad posible. Sin embargo, esto suponía que los datos de los sensores no podían ser procesados en un tiempo suficiente. Por tanto se le implementó otro más (de alto rango de distancia) y lo usaron para añadir una nueva variable, el color mediante el uso de una cámara RGB.

Los sensores de distancia analizaban el terreno determinaban las zonas por las que el vehículo podía circular correctamente. Se extraía el color de dichas zonas gracias a la cámara RGB, pero ese color no era constante en todo el terreno por distintos motivos, como por ejemplo, los brillos o las sombras. Para ello añadieron al algoritmo funciones encargadas de procesar dicha información, conociendo así la similitud entre los colores. Dicha similitud, además, permitía asumir que si un terreno detectado como transitable por los sensores tenía cierto color, todo el terreno cercano con el mismo color también sería transitable.

De esto han pasado varios años, y mientras que en 2005 en coste del hardware necesario para este fin era de más de 20.000 euros, actualmente tenemos a nuestro alcance herramientas mucho más baratas de características similares. Además, entre todas esas herramientas disponibles, se encuentran algunas que poseen gran potencia y muy bajo coste, como es la cámara Kinect, y herramientas de desarrollo orientadas a la visión 3D libres y totalmente gratuitas.

1.2. Objetivos

Antes de plantear los objetivos hay que tener en cuenta que se trata de un trabajo de investigación, por lo que si bien hay unos objetivos definidos al principio del proyecto, estos podían cambiar sobre la marcha en función de las dificultades inesperadas que pudieran ir surgiendo.

El objetivo principal era el de crear un algoritmo de libre distribución capaz de detectar superficies transitables por un robot, basándonos en el desarrollo de la Universidad de Stanford, tomando como base la publicación que realizaron.

Hablando de forma más exhaustiva, los objetivos perseguidos son:

- Uso de un hardware de bajo coste, con el fin de que el algoritmo sea fácilmente reproducible.
- Uso de software con licencia *open-source*, para potenciar el bajo coste del algoritmo y facilitar su distribución y reproducción por terceras personas.
- Usando las herramientas anteriores, programar un algoritmo capaz de detectar superficies transitables a través de los sensores de la cámara.
- Ser capaces de interpretar la información de los sensores para ampliar el rango de medida.
- Integrar sistemas de aprendizaje para agilizar el proceso de detección de suelos.
- Generar un mapa de la zona transcribiendo la información en píxel a distancias reales.

1.3. Estructura

En el presente documento se realizará una breve introducción a la tecnología necesaria y a las bases teóricas con el análisis del estado del arte y las tecnologías asociadas. En estas tecnologías se incluye una descripción a nivel técnico de la cámara Kinect, una pequeña explicación del lenguaje de programación C++, de las librerías utilizadas y del driver libre necesario para su funcionamiento. Tras eso, se explicará como se ha llegado a la solución requerida y cuales han sido las opciones contempladas en cada uno de los pasos antes de elegir la final. Por último, se realizará una demostración del funcionamiento completo del sistema.

2. Estado del arte

En este capítulo se introducirá al lector en la tecnología usada para poder llevar a cabo este proyecto y el estado en el que se encuentra el campo en el que trabajaremos. Se comenzará con la historia hasta llegar a lo que es la Kinect a día de hoy y se continuará con el estado de los campos de la navegación de robots móviles y la visión artificial.

2.1. Dispositivos que interactúan con el usuario

Para poder llegar a Kinect, hay que hacer un estudio de las situaciones previas que llevaron a Microsoft a desarrollar esta cámara. El origen de todo data de Octubre de 2003, con el lanzamiento de Eye Toy por parte de Sony para su videoconsola PlayStation 2.

Eye Toy se trataba, simplemente, de una cámara web de PC desarrollada por la empresa Logitech y utilizada para capturar movimientos e interactuar con los videojuegos. Su resolución era de 320x240 píxels, y resultó el principio de la interacción entre un dispositivo y el usuario en el mercado de masas. La cámara poseía un anillo capaz de rotar sobre la lente, y dos LED's en el frontal a modo de indicadores. Se conectaba mediante puerto USB 1.1, y tenía ciertas limitaciones, como la necesidad de una iluminación adecuada para funcionar correctamente. Toda esta tecnología no era nueva ni pionera, pero si que fue una introducción para el gran público de tecnología novedosa. Dicha tecnología se encontraba algo limitada en comparación con la disponible para grandes empresas, pero resultaba ser suficiente para usos comedidos en entornos controlados.

Años después salió a la venta la llamada PlayStation Eye, que no era más que una evolución de Eye Toy diseñada para la sucesora de PlayStation 2. Disponía de una resolución de 640x480 píxels en el caso de que se grabase a 60Hz, y de la mitad si se trabajaba a 120Hz. Un hardware continuista, que mejora la tecnología del Eye Toy original usando tecnología más actual.

Años después, Microsoft quiso dar un paso más allá con su Kinect (figura 1). Fue lanzado a finales de 2010 en todo el mundo y desde un primer momento ha cosechado un enrome éxito fuera del sector de los videojuegos. Kinect era revolucionario por ofrecer al gran público un aparato tecnológicamente avanzado a un precio muy bajo.

Figura 1: Cámara Kinect

Las posibilidades que Kinect brindaba eran enormes y el apoyo desde el primer día resultó abrumador. Pese a que primeramente los drivers tuvieron que ser programados mediante ingeniería inversa debido a la negativa de Microsoft a hacerlos públicos, finalmente se vieron obligados a cambiar de opinión y liberarlos.

Actualmente, Kinect se utiliza en diversas aplicaciones. La Universidad de California[2] evalúa todas las posibilidades de Kinect aplicada a la robótica, especialmente en navegación, realizando un análisis de las posibilidades y limitaciones de Kinect. Por su parte, la Universidad de Beijing[3] trabaja en un sistema controlado mediante gestos utilizando Kinect como base. Microsoft además está diseñando su tecnología en torno a Kinect, incluso implementando soporte nativo en su nuevo sistema operativo, Windows 8.

2.2. Navegación de robots móviles

En los últimos años, parte de los avances tecnológicos se han enfocado en la navegación autónoma de robots. Para que un robot se pueda mover por un entorno debe de ser capaz de detectar cualquier tipo de obstáculo y evitarlo de forma correcta.

Como base para el desarrollo de la navegación se usan tres técnicas diferentes: *Self-localisation*, *Path planning* y *Map-building and interpretation*. La primera técnica consiste en la reconstrucción del escenario mediante el uso de distintos sensores. Una vez conocida la forma del escenario y la colocación de los distintos elementos, el robot es capaz de conocer su posición dentro de la sala y puede moverse en consecuencia. La segunda técnica, Path planning, es una extensión de la primera. Requiere la posición del robot y del lugar objetivo con el mismo origen de coordenadas. Finalmente, Map-building consiste en la descripción de un escenario usando el marco de referencia de un robot.

La navegación basada en visión utiliza unos sensores ópticos dedicados a extraer toda la información necesaria de la localización en la que se encuentra el robot. Para lograr la interpretación de los datos obtenidos y obtener una navegación satisfactoria, existen técnicas como la representación del entorno, generar modelos o algoritmos de localización. Dentro de la navegación basada en visión existen dos

campos, la navegación en interiores y la navegación en exteriores.

La navegación en interiores puede resultar sencilla si se guía al robot hacia el objetivo. Este guiado se puede hacer de diferentes formas, entre las que se incluyen raíles magnéticos, pintar líneas en el suelo o utilización de códigos de barras.

Como trabajo actual, la Universidad de Changzhou[4], en China, diseñó un sistema de navegación por interiores basada en ultrasonidos. La emisión de ultrasonidos, junto con un conjunto de sensores, permite generar un mapa del terreno. Dicho mapa facilita las coordenadas de posicionamiento y la ruta más óptima para alcanzar el objetivo.

2.3. Visión artificial

La visión artificial es otro de los campos que más avances han recibido en los últimos años. La visión artificial estudia la capacidad para procesar e interpretar la información que se obtiene mediante sensores ópticos.

Actualmente, la visión artificial consta de tres pasos para trabajar correctamente. El primer paso constituye la adquisición de datos mediante sensores. El segundo engloba todas las funciones requeridas para procesar toda la información obtenida en bruto mediante los sensores y traducirla en información utilizable. Finalmente, el tercer paso utiliza la información obtenida mediante el procesado y la utiliza para la toma de decisiones.

En el primer paso, la adquisición de información se realiza en muchos casos con cámaras 2D convencionales, aunque existen multitud de alternativas entre las que se engloban rayos infrarrojos o cámaras 3D. Dicha información es transmitida a la unidad de proceso mediante puertos USB estándar, puertos FireWire o puertos Ethernet.

En el segundo paso, el correspondiente al procesado de información tomada en cada frame, existen multitud de métodos. Entre dichos métodos se encuentra la conversión a una escala de grises para facilitar el procesado, segmentación, reconocimiento de elementos, detección de bordes, procesado de aprendizaje mediante redes neuronales o filtrado. El resultado final tras aplicar los métodos existentes es la información utilizada para la toma de decisiones.

Finalmente, el tercer paso toma la información saliente del paso de procesado para tomar la decisión. Aplicándolo al campo de la navegación, la decisión se corresponderá a la ruta más óptima que evite cualquier peligro para el robot.

Aunque es un campo que tecnológicamente ha avanzado enormemente en cuanto a calidad y fiabilidad, los costes necesarios para implantar un sistema de navegación que utilice visión artificial son muy altos. Por tanto, parte de la investigación de estos campos consiste en implementar dicha tecnología en equipos de bajo coste.

En referencia a esas investigaciones, Universidades de todo el mundo han publicado sus soluciones ante escenarios muy concretos. Recientemente, la Universidad de Almería[5] publicaron un sistema que automatizaba el riego en invernaderos. En dicha tarea usaban un robot que integraba todos los elementos necesarios para el riego, y un sistema de navegación. Para ello implementan dos soluciones, por un lado el uso de una cámara web para recibir 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.

Por otro lado, y en este caso más centrado en la segmentación, existe un proyecto creado por Michael E. Farmer[6] que trata la segmentación de una forma novedosa. Concretamente, además de las variables geométricas, añade otras como la textura, el color o la escala de grises. Gracias a este aumento en el número de variables es posible obtener una segmentación de mayor precisión.

También existen numerosos proyectos basados en la navegación de interiores. Entre ellos, destaca el ofrecido por la Universidad Tianjin[7], en China. En este proyecto tratan de evitar el gran flujo de datos y procesos que provoca el uso de una o varias cámaras en un robot. Por ello, la cámara se encuentra colocada en el techo, y será capaz de segmentar el suelo y convertirlo en mapas de navegación que pueda utilizar el robot para moverse.

Este proyecto buscará contribuir a este estado del arte proporcionando una solución a uno de los principales problemas actuales, el precio. Actualmente muchos de los equipos utilizados para navegación autónoma tienen un coste excesivamente alto, resultando poco rentables en muchas situaciones. Por ello se utilizará un hardware moderno, actual y potente, con la peculiaridad de su bajo coste. Usando dicho hardware se proporcionará un algoritmo que interprete la información captada por la cámara y pueda ser utilizado satisfactoriamente en un sistema de navegación por interiores.

3. Fundamentos teóricos

3.1. Óptica

Un efecto del campo de la óptica que ha sido fundamental para el desarrollo de este proyecto ha sido el pinhole. El pinhole tiene como base el desarrollo de la cámara oscura. Una cámara oscura no era más que una caja en la que se realizaba un agujero que dejaba pasar la luz, y ésta se reflejaba en una superficie fotosensible. Este reflejo se ve invertido y con un tamaño proporcional pero diferente al original, variando según la distancia.

Hablando de forma más específica, la información 3D del mundo se proyecta en el plano de una imagen. Esto ocurre a través de una lente, que se encuentra a una distancia del plano de la imagen. A dicha distancia se le denomina distancia focal. Este efecto se muestra en la figura 2:

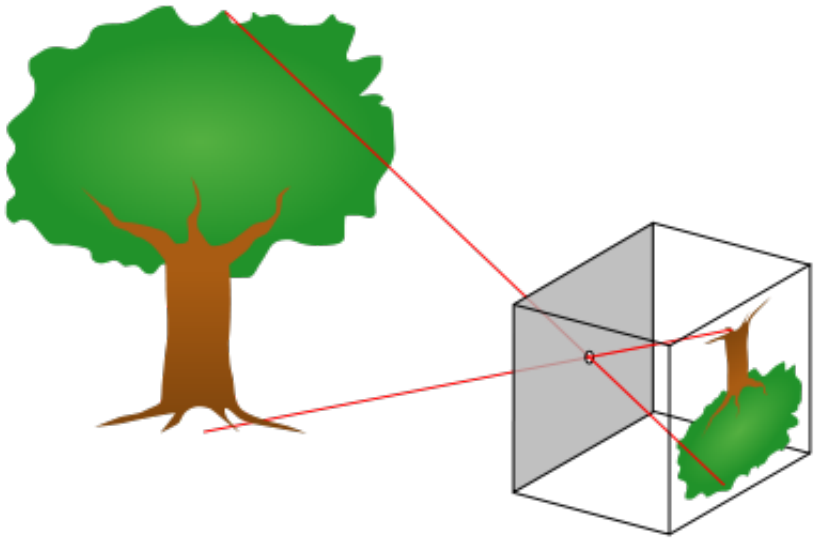

Figura 2: Ejemplo de modelo pinhole. Imagen cortesía de Wikipedia, la enciclopedia libre

El modelo pinhole, pese a poseer una alta exactitud en medidas cercanas, ha demostrado una pérdida de exactitud a medida que el objeto se encuentra más alejado en píxeles.

Existen evoluciones y derivaciones de este modelo al aplicarlo en cámaras, siendo *Range imaging* una de las más comunes. Range imaging es el nombre que reciben un conjunto de técnicas utilizadas para generar una imagen en dos dimensiones mostrando la distancia de los puntos con respecto a uno específico, generalmente asociado al tipo de sensor. La imagen resultante, *range image*, cuyo valor de píxel corresponde con la distancia. Si la cámara está correctamente calibrada, el valor del píxel se puede dar directamente en unidades físicas, como pueda ser el metro.

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 en la figura 3:

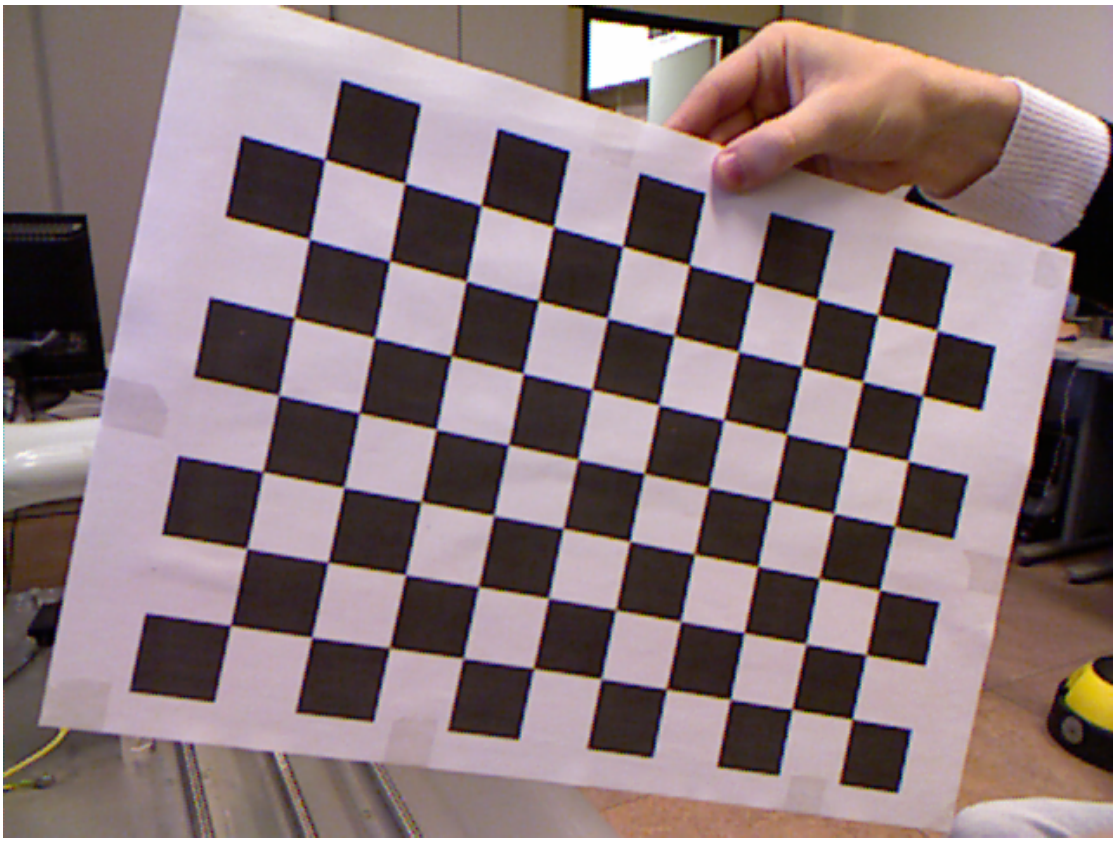

Figura 3: Ejemplo de tablero A3. Imagen cortesía de Nicolas Burrus

Usando ese tablero, de dimensiones conocidas, y colocándolo a distancias también conocidas, se puede calibrar la cámara y obtener los valores de profundidad, como se ve en la figura 4:

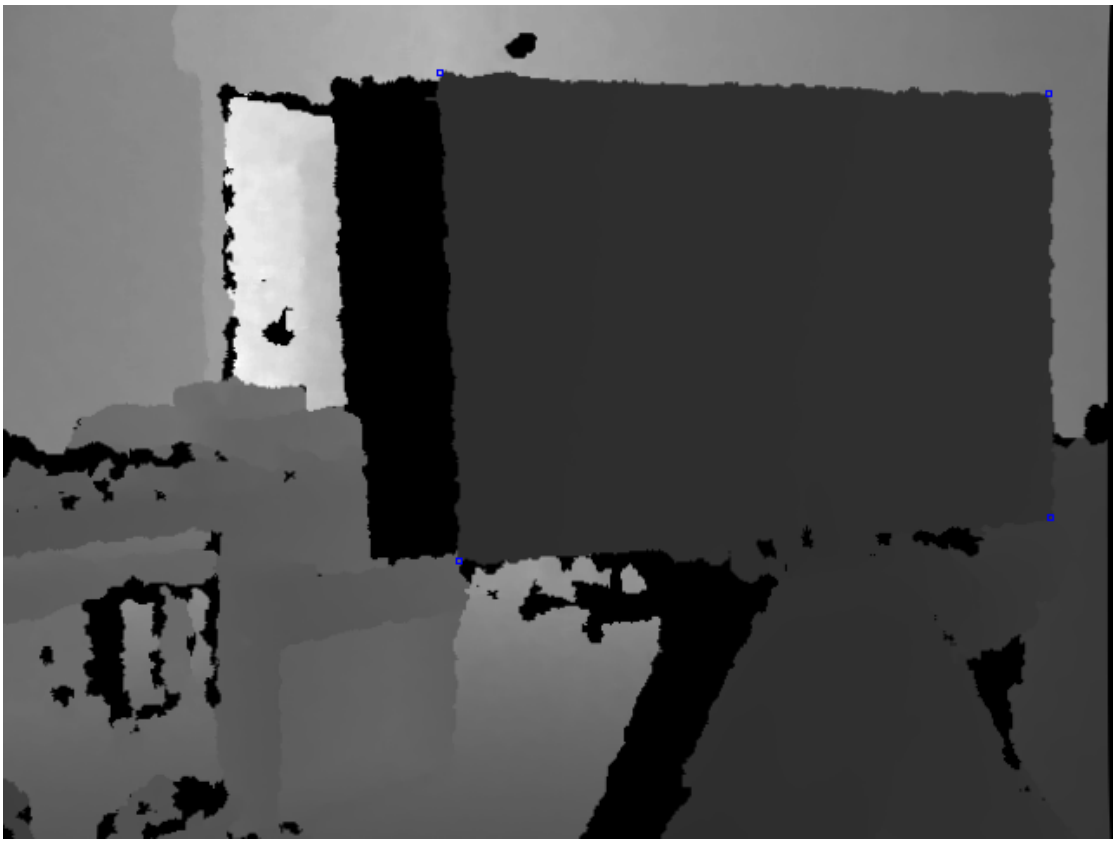

Figura 4: Ejemplo de tablero A3 calibrado. Imagen cortesía de Nicolas Burrus

3.2. Espacio de color RGB

En un espacio de color RGB, se utiliza la intensidad de los tres colores primarios, rojo, azul y verde, para generar otros colores en base a su combinación. Se trata de un modelo basado en la síntesis aditiva. Este modelo de color no define por sí mismo lo que significa rojo, verde y azul, por lo que en distintas pantallas el resultado puede ser muy diferente.

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 un efecto nulo y 255 máxima intensidad. Se suele representar en forma de cubo, como en la figura 5:

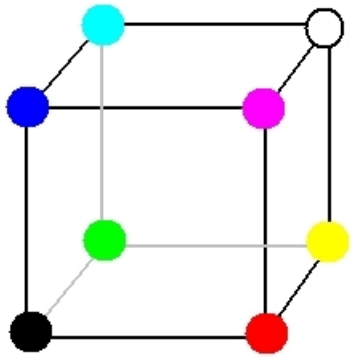

Figura 5: Cubo de color RGB

3.3. Estudio de planos

En el campo de la geometría, consideramos un plano (figura 6) como el ente ideal que posee dos dimensiones y contiene infinitos puntos y rectas. Un plano solo se puede ver descrito en relación a otros elementos geométricos similares.

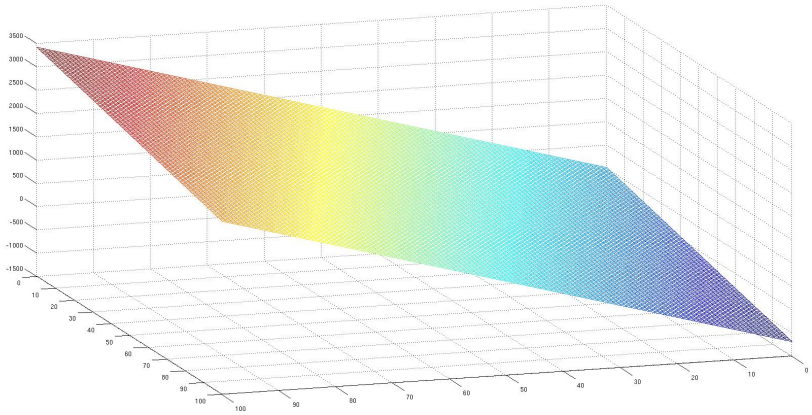

Figura 6: Ejemplo de plano

Un plano queda definido por cualquiera de los siguientes elementos: tres puntos no alineados, una recta y un punto externo a ella, dos rectas que se cortan o dos rectas paralelas. Como convención, los planos se nombran con letras del alfabeto griego. Gráficamente se suele representar como una figura con límites irregulares, para representar que es solo una parte de una superficie infinita, como en la figura 7.

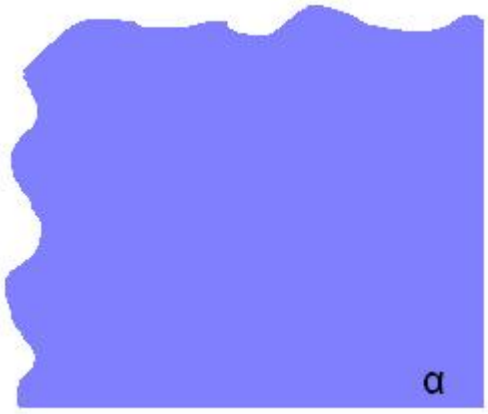

Figura 7: Ejemplo de representación gráfica de plano

Los planos en un espacio euclidiano tridimensional $\Re^3$ poseen varias características:

- Dos planos o son paralelos o se cortan en una línea.
- Una línea o es paralela al plano, o la corta en un punto o está contenida por el mismo plano.
- Dos líneas perpendiculares a un plano son paralelas entre sí.
- Dos planos perpendiculares a una misma línea son paralelos entre sí.
- Entre un plano Π cualquiera y una línea no perpendicular al mismo existe solo un plano que contenga la línea y es perpendicular a Π.
- Entre un plano Π cualquiera y una línea perpendicular al mismo existe un número infinito de planos tal que contienen a la línea y son perpendiculares a Π.

La ecuación de un plano queda definida por dos vectores y un punto por el que pasa:

$$\begin{aligned} &\text{Punto } \mathbf{P} = (x_1, y_1, z_1) \\ &\text{Vector } \mathbf{u} = (a_1, b_1, c_1) \\ &\text{Vector } \mathbf{v} = (a_2, b_2, c_2) \end{aligned}$$

Partiendo de esto podemos llegar a la ecuación reducida del plano, haciendo un producto vectorial:

$$\begin{vmatrix} (X-P) \\ u \\ v \end{vmatrix} = \begin{vmatrix} x-P_x & y-P_y & z-P_z \\ u_x & u_y & u_z \\ v_x & v_y & v_z \end{vmatrix} = 0 \Rightarrow Ax + By + Cz + D = 0 \qquad (4)$$

Siendo (A, B, C) un vector perpendicular al plano, y D la distancia respecto del origen.

Para calcular la distancia de un punto a un plano, primeramente suponemos un plano definido por $\Pi : ax + by + cz + d = 0$ y un punto cualquiera $p_1 = (x_1, y_1, z_1)$ no necesariamente contenido en el plano. La distancia entre ellos se define por:

$$D = \frac{|ax_1 + by_1 + cz_1 + d|}{\sqrt{a^2 + b^2 + c^2}} \tag{5}$$

3.4. Estudio de rectas

En términos geométricos, la recta es un componente que se extiende en una misma dirección, unidimensional y compuesta por infinitos puntos. Es una unidad geométrica fundamental, junto al plano, previamente comentado, y el punto.

La recta posee las siguientes características: se prolonga al infinito por los dos sentidos. La distancia más corta entre dos puntos está en una línea. Y, finalmente, la recta está compuesta por un conjunto de puntos situados a lo largo de la intersección de dos planos.

3.4.1. Ecuación de la recta que pasa por dos puntos

Para calcular la ecuación de una recta que pasa por dos puntos bidimensionales (x_1, y_1), (x_2, y_2), tienen que cumplirse las siguientes ecuaciones, donde m es la pendiente de la recta y b es la ordenada en el origen:

$$y_1 = mx_1 + b$$
$$y_2 = mx_2 + b$$

Como se puede ver, ambas comparten dos términos, m y b, por lo que se resta las dos ecuaciones y obtenemos:

$$y_1 - y_2 = m(x_1 - x_2)$$

Por lo que la pendiente buscada es:

$$m = \frac{y_1 - y_2}{x_1 - x_2}$$

Y si se sustituye en la primera ecuación, obtenemos que b es:

$$b = y_1 - mx_1 = y_1 - \frac{y_1 - y_2}{x_1 - x_2} x_1$$

Dejando finalmente la ecuación de la recta que pasa por dos puntos como:

$$y = \frac{y_1 - y_2}{x_1 - x_2} x + y_1 - \frac{y_1 - y_2}{x_1 - x_2} x_1 \tag{6}$$

3.4.2. Distancia de un punto a una recta

Para calcular la distancia de un punto a una recta, primeramente se supone una recta definida por $r : ax + by + d = 0$ y un punto cualquiera $p_1 = (x_1, y_1)$ no necesariamente contenido en la recta. La distancia entre ellos se define por:

$$D = \frac{|ax_1 + by_1 + d|}{\sqrt{a^2 + b^2}} \tag{7}$$

En el caso de que se trate de una recta intersección de dos planos en tres dimensiones, el procedimiento varía ligeramente.

Para poder definir la recta, es necesario conocer tanto el vector director de la recta, que se puede obtener realizando el producto vectorial de los planos, como un punto que pase por ellos. Es decir, si se tienen dos planos definidos como:

$$\pi_1 : a_1x + b_1y + c_1z + d_1 = 0$$
$$\pi_2 : a_2x + b_2y + c_2z + d_2 = 0$$

Se calcularía su vector director como:

$$\begin{vmatrix} \overrightarrow{i} & \overrightarrow{j} & \overrightarrow{k} \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{vmatrix} \tag{8}$$

Dando como resultado un vector director tal que:

$$\overline{u_r} = ((b_1c_2 - b_2c_1)\overrightarrow{i}, (c_1a_2 - a_1c_2)\overrightarrow{j}, (a_1b_2 - a_2b_1)\overrightarrow{k}) \tag{9}$$

Mientras, para obtener un punto de dicha recta es necesario asignar aleatoriamente un valor a una de las incógnitas, al tratarse de un sistema indeterminado. Dicho valor suele ser 0 por simplicidad, obteniendo lo siguiente:

$$\pi_1 : a_1x + b_1y + d_1 = 0$$
$$\pi_2 : a_2x + b_2y + d_2 = 0$$

Con ello si es posible resolver el valor del sistema y calcular las coordenadas de un punto que pase por esas rectas, definido como $P = (x_0, y_0, z_0)$. La recta se define como:

$$r \equiv \begin{cases} x(t) = x_0 + it \\ y(t) = y_0 + jt \\ z(t) = z_0 + kt \end{cases}$$

En este caso, la distancia de un punto P a esta recta, se puede calcular como:

$$d = \frac{|\overline{u_r} x \overline{AP}|}{|\overline{u_r}|} \tag{10}$$

Siendo $\overline{AP}$:

$$\overline{AP} = \overline{P} - \overline{u_r}$$

3.5. Análisis estadístico multivariante

El análisis multivariante se trata de un conjunto de métodos estadísticos que se emplea para determinar la contribución de diversos factores a un único evento. Los factores a estudiar son las variables independientes, dando como resultado una variable dependiente de las de estudio.

Éste tipo de análisis posee muchas ventajas sobre los modelos de regresión tradicionales. Por ejemplo, es posible trabajar con múltiples variables de entradas aunque no sean linealmente independientes. También se puede trabajar con matrices con más variables que observaciones, o con matrices incompletas solo si las vacantes están aleatoriamente distribuidas y no superan el 10 %[10].
Dentro del análisis multivariante existen diversas técnicas de análisis. La separación de grupos o *clustering* es una técnica muy útil cuyo objetivo es la agrupación de elementos similares entre sí. Dicha agrupación se realiza sobre unas categorías que no necesariamente han de estar establecidas.

3.5.1. Cluster k-means

Dentro de los distintos tipos de clustering, hay uno que sobresale por encima de los demás por su sencillez. Éste análisis se denomina *k-means*, o *k-medias*. El proceso a seguir es sencillo a nivel conceptual.

Inicialmente se dispone de un número n de puntos, como el mostrado en la figura 8:

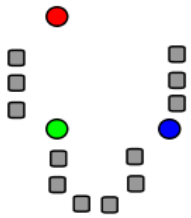

Figura 8: Paso 1 k-means. Imagen cortesía de Wikipedia, la enciclopedia libre

Tras eso, es necesario determinar el número de grupos (k) que se desean (para el ejemplo serán 3). Una vez decidido, se determina el valor central de cada uno de esos grupos y se calcula la distancia de cada dato a cada uno de los centros, agrupando cada punto junto al centro del que está más cerca, como se ve en la figura 9:

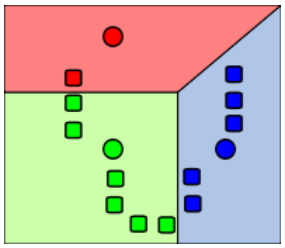

Figura 9: Paso 2 k-means. Imagen cortesía de Wikipedia, la enciclopedia libre

Posteriormente, se calculan nuevos centros que estarán en el punto medio de todos los puntos de cada agrupación, obteniendo el resultado de la figura 10:

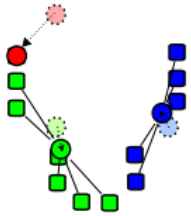

Figura 10: Paso 3 k-means. Imagen cortesía de Wikipedia, la enciclopedia libre

Finalmente se repite el proceso de las distancias para realizar nuevas agrupaciones, como en la figura 11:

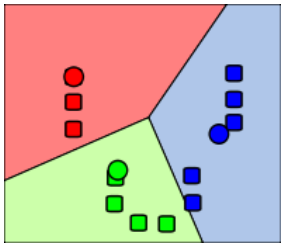

Figura 11: Paso 4 k-means. Imagen cortesía de Wikipedia, la enciclopedia libre

Todo este proceso se realiza en bucle hasta que de una iteración a otra ningún punto cambie de agrupación.

Matemáticamente, si tenemos un número n de observaciones de un vector de d dimensiones, el cluster divide en k partes, compuestas por n datos.:

$$\underset{s}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{x_j \in S_i} \|x_j - \mu_i\|^2 \tag{11}$$

El paso de asignación, o división de puntos en cluster con respecto a un centroide, se puede definir como:

$$S_i^{(t)} = \{x_p : \|x_p - m_i^{(t)}\| \leqslant \|x_p - m_j^{(t)}\| \ \forall \ 1 \leqslant j \leqslant k\} \tag{12}$$

Y el paso de actualización, o cálculo de nuevos centroides, se puede definir como:

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j \tag{13}$$

3.5.2. Distancia de Mahalanobis

En estadística, la distancia de Mahalanobis es una medida de distancia introducida por el matemático indio Prasanta Chandra Mahalanobis[8] como una mejora de la distancia euclídea. Gracias a ella se puede conocer la similitud entre variables multidimensionales.

La distancia euclídea se define para distribuciones probabilísticas como:

$$\sqrt{\Sigma_{i=1}^{n}(p_i - q_i)^2} \tag{14}$$

La distancia euclídea posee varias limitaciones, principalmente al no tener en cuenta la correlación entre las variables, como tampoco tiene la dispersión de la distribución, ya que trabaja expresamente con medias. Esto hace que la distancia euclídea sea propensa a errores en casos en los que la dispersión de la distribución es grande. Como solución a estos problemas nació la distancia de Mahalanobis. Se usará la figura 12 para ejemplificar las diferencias:

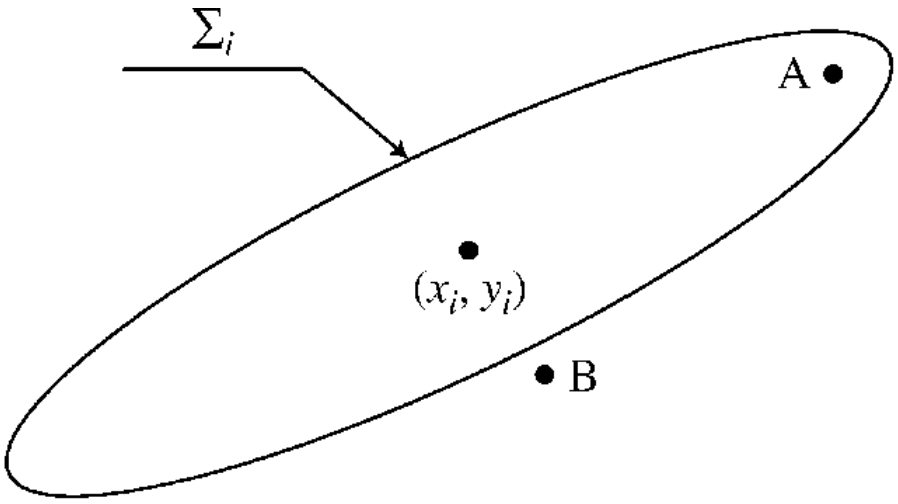

Figura 12: Ejemplo de distancia de Mahalanobis

Si en este ejemplo se calcula la distancia de A y B a la distribución gaussiana mediante la distancia euclídea, el resultado es que B está más cerca que A, erróneo ya que A es parte de la gaussiana. En cambio, si se procede al mismo problema mediante la distancia de Mahalanobis, el resultado será que A es más próximo que B.

La distancia de Mahalanobis está definida como la distancia entre dos variables

aleatorias que comparten una distribución de probabilidad $\overline{x}$ y $\overline{y}$ con una matriz de covarianza Σ:

$$d_m(\overline{x},\overline{y}) = \sqrt{(\overline{x}-\overline{y})^T\Sigma^{-1}(\overline{x}-\overline{y})} \tag{15}$$

3.6. Distribución gaussiana

En el campo de la estadística, se denomina función gaussiana a una de las distribuciones de probabilidad de variable continua que aparece con más frecuencia. La gráfica de su densidad tiene una forma acampanada y es simétrica respecto a su media, μ, por lo que también recibe el nombre de *Campana de Gauss*.

La función gaussiana unidimensional es una función que se define por la expresión:

$$f(x) = ae^{-}\frac{(x-b)^2}{2c^2} \tag{16}$$

Siendo a, b y c constantes.

Las funciones gaussianas utilizadas en estadística se corresponden a la densidad de una variable aleatoria con distribución normal de media $\mu = b$ y varianza $\sigma^2 = c^2$. Esto solo se da en el caso en que se de:

$$a = \frac{1}{c\sqrt{2\pi}} \tag{17}$$

En estadística, la media, μ, se define como la media aritmética de todos los datos:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} X_i \tag{18}$$

Mientras que la varianza, σ^2 queda definida de la siguiente forma:

$$\sigma^2 = \lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)^2 \tag{19}$$

En la figura 13 se puede observar una representación de una campana de Gauss.

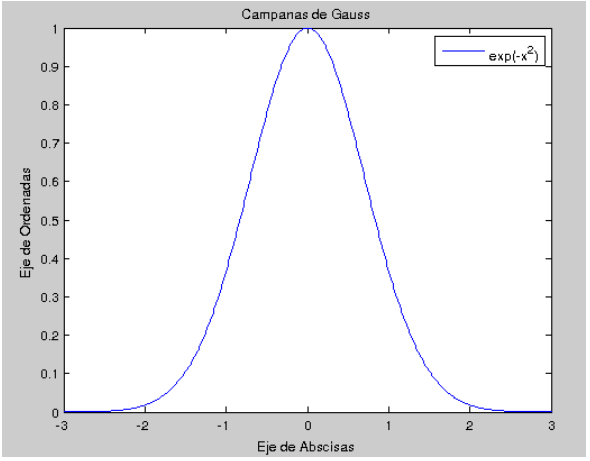

Figura 13: Ejemplo de distribución gaussiana

En el caso multivariable, el conjunto de medias es:

$$\mu = \begin{vmatrix} \mu_0 \\ \mu_1 \\ ... \\ \mu_n \end{vmatrix} \tag{20}$$

Y la matriz de covarianzas se define como:

$$\mathbf{\Sigma} = \begin{vmatrix} \sigma_{11}^2 & \sigma_{12}^2 & ... & \sigma_{1n}^2 \\ \sigma_{21}^2 & \sigma_{22}^2 & ... & \sigma_{2n}^2 \\ \sigma_{n1}^2 & \sigma_{n2}^2 & ... & \sigma_{nn}^2 \end{vmatrix} \tag{21}$$

Siendo la diagonal principal la varianza y el resto de elementos, los elementos de covarianza, calculados como:

$$\sigma^2{}_{i,j} = \frac{\Sigma_{i=1}^{n}(X_i - \overline{x})(Y_i - \overline{y})}{n-1} \tag{22}$$

En la figura 14 se muestra un ejemplo de gaussiana bidimensional:

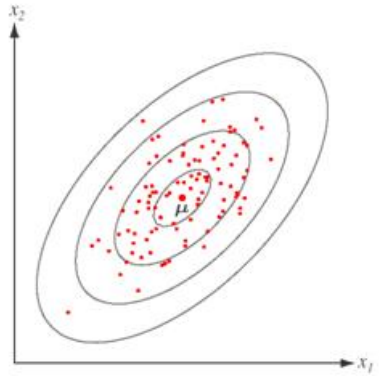

Figura 14: Ejemplo de distribución gaussiana en dos dimensiones

Las propiedades de la distribución normal son las siguientes:

- Es simétrica respecto a su media, μ.
- La moda y la mediana son iguales a μ.
- Los puntos de inflexión de la curva son: $x = \mu - \sigma$ y $x = \mu + \sigma$.
- En el intervalo $[\mu - \sigma, \mu + \sigma]$ se encuentra el 68.26 % de la distribución.
- En el intervalo $[\mu - 2\sigma, \mu + 2\sigma]$ se encuentra el 95.44 % de la distribución.
- En el intervalo $[\mu - 3\sigma, \mu + 3\sigma]$ se encuentra el 99.74 % de la distribución.

4. Tecnologías asociadas

En esta sección se realiza una introducción a todo el área tecnológico sobre el que se basa el trabajo desarrollado. Se comenzará con la cámara Kinect, siguiendo con el lenguaje C++ y las librerías adicionales que han sido utilizadas para poder llevar a cabo el proyecto. Seguidamente se hablará de las herramientas de desarrollo necesarias, para finalmente tratar el driver libre de Kinect.

4.1. Kinect

Kinect, físicamente, se trata de una barra horizontal, conectada a una pequeña base con una pieza motorizada, diseñada para colocar la cámara del modo correcto para funcionar. El dispositivo posee una cámara RGB, un sensor de profundidad y un micrófono multibanda, todo ello funcionando sobre software propietario. Todo ello provee una captura tridimensional de gestos, movimientos y voz.

El sensor de profundidad lanza un láser por todo el campo que puede abarcar, conociendo su posición exacta. Si el láser rebota con los elementos la trayectoria se desvía, y en función de la nueva colocación de los rayos es capaz de conocer la presencia de objetos y su colocación aproximada. Dicho láser actúa en el campo infrarrojo y es capaz de calcular de forma aproximada la distancia en el mundo real con respecto a sus ejes de los elementos. El láser infrarrojo, como punto negativo, solo puede actuar en unas condiciones de luz muy concretas que solo se dan en interiores y a unas distancias muy concretas. Demasiado cerca es muy problemático, y demasiado lejos no funciona. El rango del sensor es ajustable, y la propia Kinect es capaz de autorregularlo si las condiciones van cambiando, y acomodándose a los obstáculos que puedan aparecer. En la figura 15 se muestra un ejemplo de los rayos que manda la Kinect:

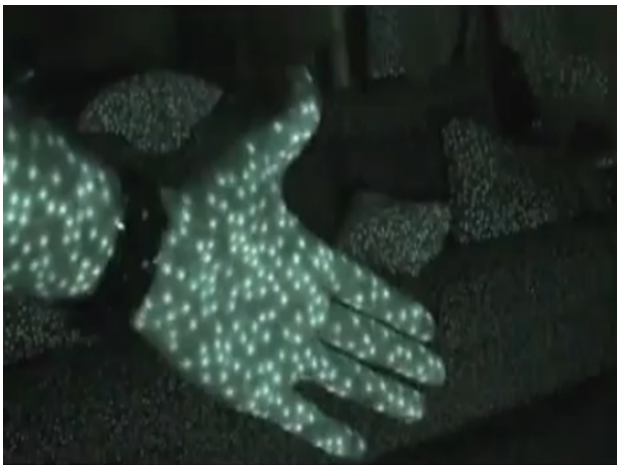

Figura 15: Lanzamiento de rayos infrarrojos de Kinect

Mientras que el resultado que se obtiene del sensor láser en un pasillo es la figura 16:

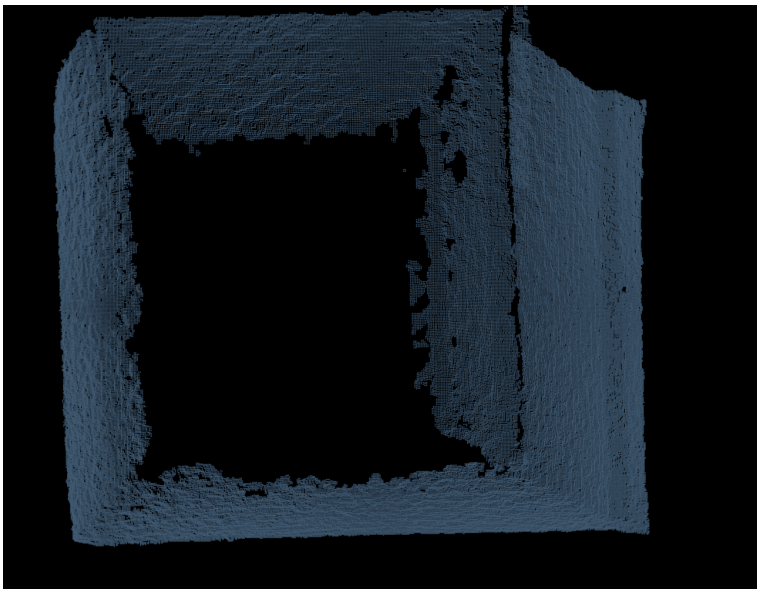

Figura 16: Nube de puntos de un pasillo

Aunque lo que sigue en este texto no son datos oficiales, son los resultados obtenidos tras un arduo trabajo mediante ingeniería inversa. Se ha descubierto que la salida de vídeo funciona a 30Hz, mientras que la cámara RGB se trata de una cámara VGA[9] de 8 bits con una resolución de 640x480 píxels que trabaja sobre un filtro *Bayer*. Este tipo de filtro convierte la información para trabajar con *arrays*. El sensor de profundidad es monocromo, con resolución VGA de 640x480 píxels también, y con 11 bits de profundidad, lo que nos da 2.048 niveles de sensibilidad.

La cámara tiene un rango de trabajo entre 1.2 y 3.5 metros de distancia desde la posición de la cámara, aunque se puede llegar a 6m. La cámara posee un ángulo de visión de 57º horizontalmente y 43º verticalmente.

Para funcionar, Kinect necesita alimentación externa a la que proporciona el puerto USB 2.0, por lo que necesita una toma de corriente para poder funcionar.

4.2. OpenNI

Figura 17: Logo de OpenNI

OpenNI (*Open Natural Interaction*, figura 17) se trata de una organización sin ánimo de lucro enfocada en la mejora de la interoperabilidad entre interfaces de

usuario naturales e interfaces de usuario orgánicas para una interacción suave que facilite el uso de distintos dispositivos.

La organización fue creada en noviembre de 2010, siendo uno de los miembros más importantes PrimeSense, la creadora de la tecnología utilizada por Kinect. En Diciembre de 2010, PrimeSense desarrolló unos drivers libres de Kinect para PC y anunciaron un acuerdo con Asus para sacar una cámara similar a Kinect para PC, de nombre ASUS Xtion.

El *OpenNI framework* provee una serie de APIs[11] libres con el fin de estandarizar las aplicaciones que buscan una interacción natural. Generalmente estas API son referidas directamente como OpenNI. Las API proveen soporte para reconocimiento de comandos por voz, gestos con la mano y movimientos con el cuerpo.

4.3. C++

C++ se trata de un lenguaje de programación muy extendido y de gran potencia y flexibilidad. Un lenguaje de programación no es más que un idioma artificial diseñado para expresar acciones que tras un proceso de compilación, en el caso de que sea un lenguaje compilado, puedan ser entendidas por una computadora.

C++ fue creado en los años ochenta por Bjarne Stroustrup[12], extendiendo el antiguo pero potente lenguaje C a la programación orientada a objetos. Actualmente existe un estándar de C++, llamado ISO C++, al que se han ido añadiendo los creadores de compiladores.

4.4. Herramientas de desarrollo

En esta sección se hará un repaso de las herramientas utilizadas para poder programar en el lenguaje C++, como son la colección de compiladores GCC o la herramienta de gestión de proyectos de software multiplataforma Cmake.

4.4.1. Colección de compiladores GNU

Figura 18: Logo de GCC

La colección de compiladores GNU (*GNU Compiler Collection*, figura 18), GCC, son unas herramientas libres creadas por el proyecto GNU de forma libre y liberadas bajo la licnecia GPL. Estos compiladores son básicos en los sistemas basados en UNIX[13], tanto libres como propietarios (como es el caso del Mac OS X de Apple).

GCC nació como parte del proyecto GNU, creando un compilador exclusivamente de C. Con el tiempo se fue extendiendo a más lenguajes, entre los que se encuentra C++ o Python. El proyecto GNU se inició en los años ochenta con Richard Stallman[14] como máximo responsable, con el fin de crear un sistema operativo totalmente libre.

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. También soporta multitud de plataformas diferentes además de la x86 clásica de los ordenadores. Por ejemplo, tiene soporte para ARM (tipo de arquitectura muy extendida entre dispositivos móviles como teléfonos o tabletas) o PowerPC (arquitectura que actualmente solo se encuentra en videoconsolas o estaciones para cálculos complejos). Además es el compilador integrado en multitud de famosos entornos de desarrollo multiplataforma, como Eclipse, Netbeans o Code::Blocks.

4.4.2. CMake

Figura 19: Logo de CMake

CMake (figura 19) es una herramienta que se encarga de la generación de código para ayudar a portar código 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*, que contendrá 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, compilador elegido y la ubicación de las librerías que van a ser enlazadas. Como extra es posible generar un *Makefile* específico para algunos entornos de desarrollo concretos, como Visual Studio o Eclipse.

CMake nació como la necesidad de otorgar facilidad para la construcción multiplataforma. Comenzó basándose en un sistema anterior, *packer*, y fue recibiendo un gran número de características y añadidos, principalmente de desarrolladores externos que iban adaptando la herramienta a sus necesidades, dando lugar a lo que conocemos hoy día como CMake.

Entre otras características, permite analizar dependencias para los lenguajes C, C++, Fortran y Java. También permite el uso de compilaciones paralelas o cruzadas, además de generar ficheros *Makefile* específicos para algunos entornos de desarrollo.

4.5. Licencia GPL

Figura 20: Logo de GPL versión 3

Una de las bases de este 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 va a ser publicado bajo la llamada licencia *GNU General Public License* (GPL, figura 20).

Esta licencia fue creada por la *Free Software Foundation*, encabezada por Richard Stallman, en 1989, como forma para proteger el software libre, con el fin de impedir la apropiación del código y el robo de libertades.

En términos legales, la licencia GPL actúa como un contrato debido a que el documento cede algunos derechos al usuario, siendo una licencia reconocida en los juzgados.

Actualmente la licencia se encuentra en su tercera revisión, publicada en Junio de 2007. En ella se contemplan aspectos como las formas en las que se podría robar la libertad de los usuarios, prohibir el uso de software libre en sistemas que utilicen DRM (un sistema de gestión de derechos digitales con muchos detractores, entre los que se encuentra la *Free Software Foundation*). También se resuelven vacíos legales y se ayuda a compatibilizarla con otras licencias. También se defiende al usuario en el caso de uso indebido de patentes y se facilita su adaptación a otros países para que pueda ser contemplada en los juzgados.

En proyectos grandes, algunos de esos puntos son fundamentales, especialmente la compatibilidad de licencias. Eso quiere decir que se puede licenciar como GPL código que contenga partes publicadas bajo otra licencia, siempre y cuando se encuentre entre la lista de licencias aceptadas.

4.6. 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.

Dentro de las librerías, nos encontramos con dos tipos fundamentales: estáticas y dinámicas. En sus orígenes, las librerías solo podían ser estáticas, estando integradas en el propio archivo ejecutable. Este paso se realizaba al compilar, ya que el compilador en lugar de crear un enlace simbólico a las librerías las incluía dentro del ejecutable. Concretamente, convertía todas las direcciones no resueltas

en direcciones fijas, cargando tanto el código como las bibliotecas en posiciones de memoria en el tiempo de ejecución. Dicho proceso de enlazado es lento y es necesario cada vez que algún módulo es modificado.

Por otro lado, y como avance reseñable en este campo, nos encontramos con las librerías dinámicas. En este caso las librerías se cargan en tiempo de ejecución, en lugar de hacerlo en tiempo de compilación. A la hora de compilar, el proceso de enlazado es muy rápido ya que solo crea enlaces simbólicos a la posición de la librería, en lugar de crear enlaces estáticos.

Las bibliotecas dinámicas proporcionan un mayor rendimiento en los programas, unas mejoras en el tiempo de compilación y una reducción en el tamaño de los ejecutables, por lo que suelen ser la opción preferente a la hora de programar.

4.6.1. Point Cloud Library

Figura 21: Logo de Point Cloud Library

Point Cloud Library (PCL, figura 21) es 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. 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.
Estos algoritmos se usan especialmente en el campo de la robótica, concretamente en la percepción. Esta librería es básica en este proyecto, por lo que es importante entrar en cierto grado de detalle al hablar de ella.

Está diseñado de forma modular, concretamente en los siguientes:

- *Filters*: Este módulo engloba los algoritmos de eliminación de ruido. Se basan en la distribución de puntos de la nube con respecto a sus vecinos.
- *Features*: Este conjunto de funciones abordan las representaciones de cierto punto 3D en el espacio, describiendo patrones geométricos basados en la información disponible alrededor del punto.
- *Keypoints*: Contiene las implementaciones de detección de puntos clave en una nube de puntos. Los puntos clave son puntos estables, distintivos y que

pueden ser identificados siguiendo criterios bien definidos. Lo habitual es que el número de puntos clave en una nube sea notablemente inferior al número de puntos totales, representando de forma descriptiva y compacta la nube original.

- *Registration*: Se trata del modelo que combina diversas fuentes de datos en un modelo global. La idea clave es identificar los puntos correspondientes entre las fuentes de datos y encontrar la transformación que minimice la distancia entre esos puntos. Este proceso se repite hasta que los errores de alineamiento desaparecen.

- *KdTree*: Es un conjunto de librerías que usando a su vez la librería FLANN (librería de C++ especializada en búsquedas de puntos próximos en espacios de grandes dimensiones) obtiene como resultado unas búsquedas eficientes y rápidas de elementos cercanos. Un *Kd-tree*, o árbol de k dimensiones, es una estructura de datos espacial que engloba un conjunto de puntos de k dimensiones permitiendo una búsqueda eficiente de sus puntos próximos.

- *Octree*: Esta librería proporciona métodos para generar estructuras de datos jerárquicas. Esto habilita un particionamiento espacial, simplificación y búsqueda de operaciones en los conjuntos de datos de puntos. Cada uno de los *Octree* tiene ocho nodos, que no necesariamente son derivados de él. El nodo principal describe un cubo delimitador que encapsula todos los puntos. Por cada nivel del árbol este espacio se subdivide en el factor de 2 que resulta tras incrementar la resolución del *voxel*. La correcta implementación de *Octree* facilita el trabajo con puntos próximos y provee multitud de algoritmos implementados que realizan este tipo de tareas.

- *Segmentation*: Se trata de un módulo que contiene algoritmos para segmentar nubes de puntos. Estos algoritmos son idóneos para nubes compuestas por varias zonas aisladas. En estos casos la nube se fracciona en distintas partes que son procesadas de forma independiente.

- *Sample Consensus*: Es una librería que implementa métodos *SAmple Consensus*, SAC, como por ejemplo RANSAC. Este tipo de métodos se encargan de estimar parámetros de un modelo matemático partiendo de un conjunto de observaciones. Por ello, en esta librería también se incluyen algoritmos para hallar modelos como planos, líneas, esferas o cilindros. La adecuación de planos ayuda entre otras cosas a detectar paredes, suelos, techo u objetos tales como puertas. Por ese motivo, es una herramienta fundamental en este proyecto.

- *Surface*: Esta librería reconstruye superficies desde escaneados tridimensionales. El alisado y el remuestreo pueden ser importantes dependiendo del grado de ruido de la muestra, o de si la muestra está formada por distintos escaneados que no están alineados correctamente. La complejidad de la estimación es ajustable, y se pueden elegir varios métodos, principalmente una triangulación entre los puntos o un proceso de mallado. Las diferencias

principales son que mientras el primero es rápido, su fiabilidad cae con respecto al segundo, que es notablemente más lento. Su elección se basa en la precisión que se busque.

- *Range Image*: Este módulo contiene dos clases encargadas de la representación y el procesado con *range images*, que no son más que las imágenes cuyos píxeles representan la distancia con respecto al origen del sensor. Conociendo los parámetros que utiliza la cámara para calibrarse, esta imagen se puede transformar en una nube de puntos.

- *IO*: Estas librerías contienen todo lo necesario para poder leer y escribir ficheros que guarden los datos de unas nubes de puntos, en formato *PCD*.

- *Visualization*: Son librerías diseñadas para la visualización de los datos tridimensionales de la nube de puntos. Las rutinas que utiliza dibujan imágenes en bidimensionales de forma que den la sensación de tener tres dimensiones. La librería incluye funciones de renderizado y configuración de propiedades, tipo colores, tamaño de los puntos, etc. También trae métodos para dibujar una estructura tridimensional básica en pantalla, visualización de histogramas, y, finalmente, visualización de *range images*. Se incluye una aplicación, llamada *pcd viewer*, que representa por pantalla nubes de puntos guardadas en el formato *PCD*, pudiendo modificar parámetros en tiempo real como la representación del color o el tamaño.

- *Common*: Contiene las estructuras de datos y métodos comunes, utilizados por la mayor parte de las librerías PCL. Entre otras cosas, incluye las estructuras de las posiciones, los colores o la representación de puntos. También incluye funciones para calcular distancias, medias o transformadas geométricas.

- *Search*: Este último módulo engloba todos los métodos utilizados para la búsqueda de puntos próximos, incluidos en otras secciones como *Octree* o *KdTree*.

4.6.2. OpenCV

Figura 22: Logo de OpenCV

OpenCV (figura 22) 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, ya sea con fines comerciales o de investigación, siempre que se cumplan las condiciones de la misma. Es una librería multiplataforma, con versión para Windows, MacOS X y GNU/Linux, y contiene

más de 500 funciones que abarcan todo el área de la visión.

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

5. Algoritmo propuesto para segmentación y etiquetado de zonas navegables

En esta sección se explicará en detalle el desarrollo del algoritmo y las distintas opciones barajadas. Por ello, primero se hará un breve resumen del objetivo del proyecto, y posteriormente se seccionará el programa en distintos trozos que serán analizados de forma individual, explicándolos en detalle y razonando el motivo de su elección. Para poder simplificar tanto la explicación como las pruebas, todas las pruebas se realizarán sobre nubes de puntos estáticas, donde todo el proceso se realiza una única vez. En el capítulo siguiente se entrará en el análisis del programa en forma dinámica.

La elección de opciones se realiza fundamentalmente teniendo en cuenta el tiempo de proceso. En algunos casos, se tiene en cuenta la función aislada, y en otros un conjunto de ellas al variar su velocidad en función de la anterior. Para poder medir el tiempo de forma fidedigna PCL incluye unas funciones, de nombre *tic* y *toc*, que nos devuelven el tiempo consumido, tiempo que será utilizado como base para las comparativas. Al ser un proceso matemático, el hardware sobre el que funcione es fundamental, requiriendo una gran capacidad de procesamiento por parte del microprocesador del ordenador para funcionar rápidamente. En este paso, la máquina sobre la que se realizan las pruebas es un microprocesador Intel i5 2500K con una frecuencia de reloj de 3.3GHz.

5.1. Introducción

Este proyecto trata de paliar una de las limitaciones de Kinect, expandiendo su rango de trabajo. Para explicarlo de forma sencilla, en la figura 23 se mostrará una imagen de lo que puede captar la cámara RGB de Kinect:

Figura 23: Captura RGB

Sin embargo, la información de profundidad abarcada por Kinect es limitada. Por tanto, si se superponen ambas informaciones, el resultado se puede ver en la figura 24:

Figura 24: Rango de Kinect

Como se puede apreciar, pese a que se detecta una parte del suelo, la cámara RGB tiene un rango que abarca mucho más terreno. Por tanto, la misión del proyecto es crear un algoritmo capaz de clasificar el suelo que se encuentra más allá del rango del sensor a través de los parámetros de color de la cámara RGB. El resultado final ideal sería el mostrado en la figura 25:

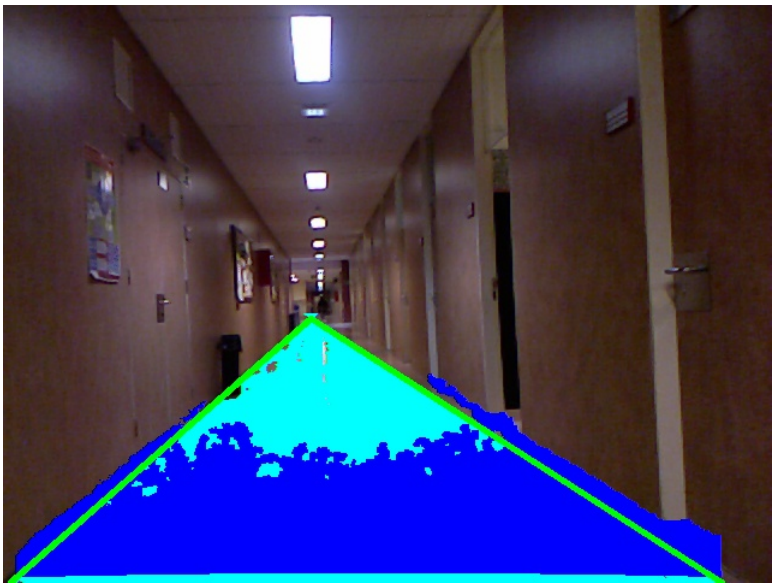

Figura 25: Resultado final ideal

En esta imagen se puede apreciar como de un color azul celeste se amplia el rango de la cámara de una forma fidedigna, excluyendo algunos elementos como

la papelera. Finalmente, a través de los píxeles del suelo se podrán extraer las distancias reales, pudiendo conocer así la posición de cada elemento.

Antes de comenzar la explicación de las distintas secciones, se mostrarán capturas de la nube de puntos sobre la que se va a trabajar. Esta nube se muestra por pantalla usando un software integrado en las librerías PCL, de nombre *pcd viewer*, como se puede ver en la figura 26:

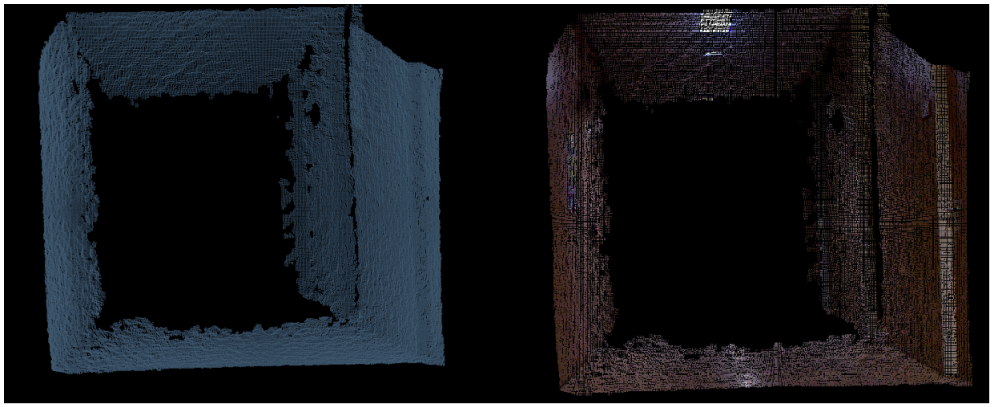

Figura 26: Nube de puntos original

5.2. Partes del algoritmo

Anteriormente se ha hablado del objetivo del proyecto, por lo que en esta sección se realizará un breve resumen de las distintas partes que componen dicho algoritmo, previo a la explicación detallada en las siguientes secciones.

Al comienzo del algoritmo se trabaja directamente con la nube de puntos, pues es la mejor base para extraer la información del entorno al ser una representación tridimensional. Debido a que la información que interesa en este caso es estrictamente la del suelo, el primer paso consiste en segmentar la nube para únicamente trabajar con él.

Una vez determinada la zona de la nube que se considera suelo, es preciso traspasar la información de la nube de puntos a píxeles, para así poder relacionarlo con la imagen RGB. Para ello se aprovecha la estructura de datos que manejan las librerías PCL y se hace una transformación adecuada.

Posteriormente, se procede a extraer los parámetros de color de los puntos de la nube pertenecientes al suelo. Debido a que es un flujo muy grande de datos, se procederá a agrupar los datos utilizando una técnica de clustering y posteriormente transformar esa información en parámetros de una gaussiana de tres dimensiones, para facilitar los cálculos posteriores.

Con toda esta información guardada se comienza a trabajar con la imagen extraída de la cámara RGB. Para ello se utiliza la transformación desde puntos de una nube a píxeles realizada anteriormente, y se extraen los parámetros de color

de la imagen RGB.

Para conocer cuales de esos píxeles poseen un color similar a los puntos de la nube etiquetados como suelo, se calcula la distancia entre dicho color y cada gaussiana. Si son parámetros cercanos se considera una extensión del suelo que no alcanza a detectar el sensor de Kinect, en caso contrario se discrimina y se considera obstáculo.

Debido a que es un algoritmo preparado para ser ejecutado en bucle, se han implementado técnicas de aprendizaje para evitar la saturación del sistema almacenando parámetros de gaussianas muy similares.

En la figura 27 se presenta el diagrama de flujo del programa, coincidiendo con la estructura general de la memoria.

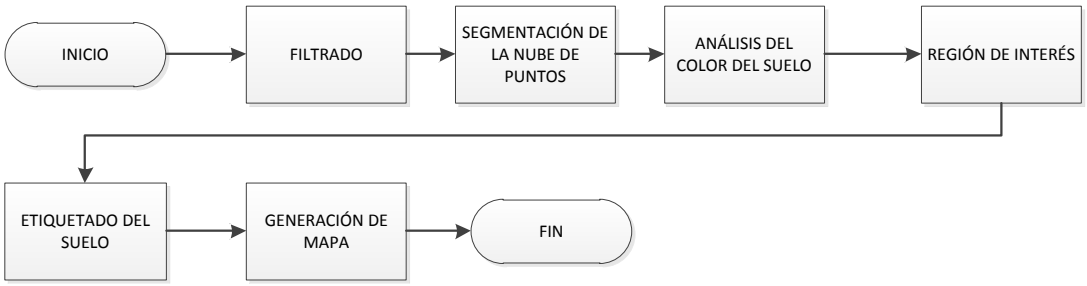

Figura 27: Diagrama de flujo general del algoritmo

5.3. Filtrado de la nube de puntos

Se comenzará el algoritmo con un proceso de filtrado. El filtrado consiste en la eliminación de un exceso de datos, tratando de suprimir la información superficial para únicamente mantener la característica. Así se consigue un aumento de velocidad en el procesado de información. Para ello, se hará uso de un filtro de tipo *voxel grid*.

El filtro *voxel grid* genera una rejilla de cubos de tamaño idéntico en la nube de puntos, detectando en que cubo se encuentra cada punto. Una vez detectado, simplifica los puntos de cada cubo en uno solo, que estará colocado en el centroide de los anteriores. Al finalizar devuelve una nube con un número menor de puntos.

La disminución de puntos depende íntegramente del tamaño del cubo, reduciéndose considerablemente a medida que el tamaño del cubo aumenta. Por contra, al disminuir el número de puntos se aumenta el riesgo a perder elementos característicos, por lo que de forma empírica fue necesario encontrar un punto medio entre ligereza y fidelidad con la imagen original.

Para poder realizar comprobaciones sobre el funcionamiento, se han añadido funciones de guardado de nubes en el disco duro, que serán eliminadas en la versión final del código.

Se comenzará con el algoritmo utilizando el filtrado previo a la segmentación. El resultado obtenido es el mostrado en la figura 28:

```
PointCloud before filtering: 307200 data points.
PointCloud after filtering: 36242 data points.
24 ms
```

Figura 28: Filtrado de la nube de puntos

De aquí se pueden extraer varios datos. Primeramente, la reducción de puntos es notable, quedando reducidos a prácticamente un 10 % de los puntos originales. En la figura 29 se muestra el resultado final:

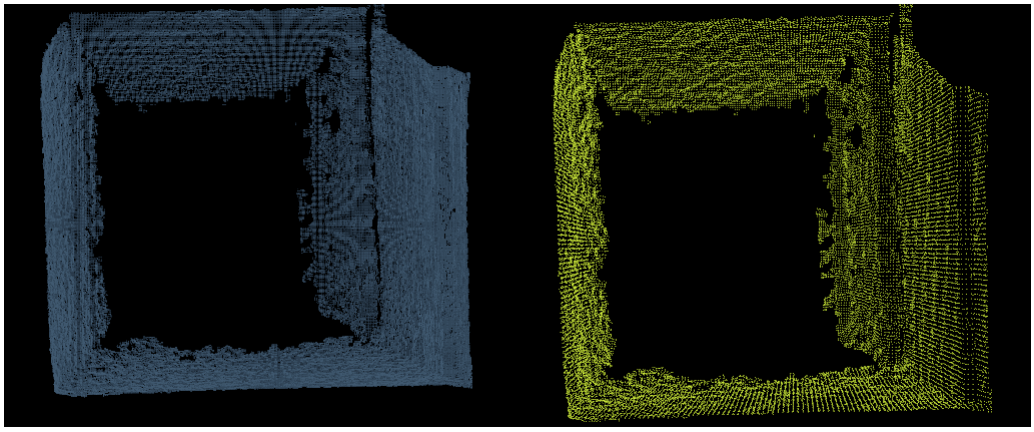

Figura 29: Nube original frente a filtrada

Se puede observar una notable bajada en el número de puntos, sin llegar a perder detalles relevantes. Este proceso dura 24ms. En la siguiente sección se evaluará el impacto de este filtrado en su funcionamiento.

5.4. Segmentación de la nube de puntos

El segundo paso del algoritmo se corresponde con la segmentación de la nube en los distintos planos que lo conforman. Para las necesidades del proyecto existe una función que ayuda a la segmentación, llamada *extract inliers*, ya que se puede usar para seccionar la nube en otras nubes más pequeñas.

Para realizar este ajuste de planos, se hará uso del algoritmo *RANSAC*, abreviación de *RANdom SAmple Consensus*. Es un método que, de forma iterativa, ajusta los parámetros que definen a un plano que abarque el mayor número de puntos. Para minimizar los errores, RANSAC también evalúa la distancia entre los puntos, discriminando los puntos que superen un límite de distancia.

A modo de recordatorio, un plano se puede definir como:

$$ax + by + cz + d = 0 \tag{23}$$

Tras obtener los coeficientes (a, b, c y d), el programa los analiza para determinar su posición. Geométricamente, 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. Si el valor absoluto del coeficiente a del plano se encuentra comprendido entre 0.9 y 1, la situación del plano se considera vertical. Por este motivo, e caso de que esto ocurra, las dos posibilidades que existen es que sea algo relacionado con la pared derecha, o con la pared izquierda. Por contra, si es el coeficiente b el que se encuentra entre 0.9 y 1, el plano es considerado horizontal, dejándonos como posibilidades que se encuentre relacionado con el suelo o con el techo.

Como se ve, aunque se pueda conocer si el plano es vertical u horizontal, es necesario determinar en que posición exacta se encuentra. Para ello, analizamos el coeficiente d del plano. Si se ha determinado que el plano es vertical y dicho coeficiente tiene un valor positivo, está situado a la derecha, mientras que si es negativo, lo hace a la izquierda. En los planos horizontales, un coeficiente d positivo indica techo, y uno negativo suelo.

Como prueba, se va a mostrar una serie de planos de ejemplo. Hay que tener en cuenta que al ser planos infinitos, ambos acaban cruzándose en una determinada recta del espacio.

- Si se tienen los planos siguientes:
  $\pi_1 : 0{,}95x - 0y + 0{,}2z + 1 = 0$
  $\pi_2 : 0{,}99x - 0y + 0{,}2z - 1{,}5 = 0$

  En la figura 30 se obtiene la representación vista desde los ejes XZ:

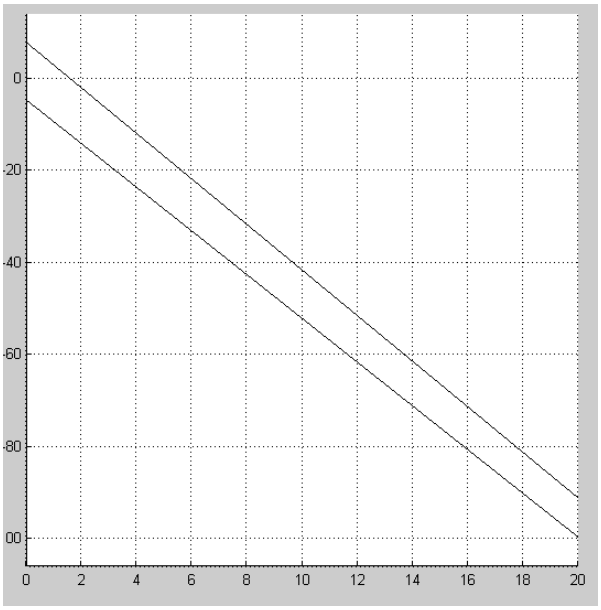

Figura 30: Ejemplo de planos verticales

- Mientras que con los siguientes planos:
  $\pi_1 : 0x - 0{,}99y - 0{,}1z + 1 = 0$
  $\pi_2 : 0x - 0{,}95y + 0{,}1z - 2 = 0$

  Se consigue la representación vista desde los ejes YZ en la figura 31:

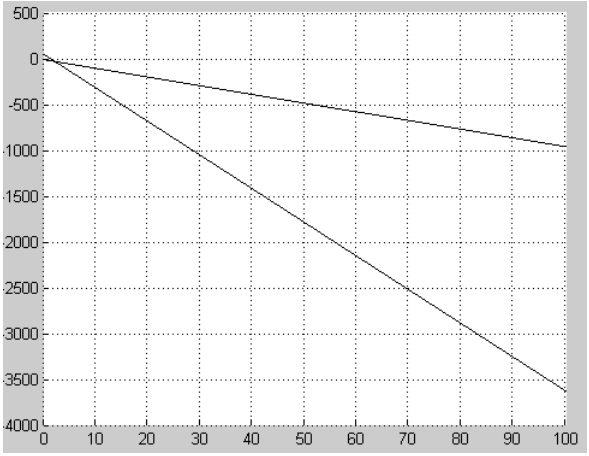

Figura 31: Ejemplo de planos horizontales

Finalmente, el software también detecta la presencia de objetos relevantes en cada una de las cuatro zonas, entendiendo relevantes como objetos de tamaño suficientemente grande como para ser modelados mediante planos. Se asume que los planos principales (paredes, suelo y techo) van a tener una superficie mucho mayor que el resto de objetos. En este caso, el límite escogido es que el plano detectado abarque al menos un 30 % de los puntos de la imagen que aún no se ha segmentado.

A partir de este punto es donde aparecen dos posibilidades. La nube de entrada, debido a las características y modo de funcionamiento de Kinect, se trata de una nube de un tamaño de 640 puntos a lo ancho y 480 puntos a lo largo, lo que da como resultado 307200 puntos en total.

Muchos de estos puntos pueden ser eliminados sin perder elementos característicos, dejando como resultado una nube con menos puntos pero que a cambio acelere el proceso de seccionado, y por tanto, del algoritmo en general.

Ejecutando el código, el resultado obtenido es el obtenido en la figura 32:

```
PointCloud before filtering: 307200 data points.
PointCloud after filtering: 36242 data points.
24 ms

PointCloud right wall: 8394 points.
Parameters of plane right wall:
a: 0.990276
b: -0.0624696
c: 0.124303
d: 1.1342

PointCloud left wall: 7688 points.
Parameters of plane left wall:
a: 0.995246
b: -0.0303002
c: 0.092564
d: -1.20326

PointCloud roof: 6360 points.
Parameters of roof:
a: -0.0469737
b: -0.998577
c: 0.0252612
d: 0.919507

PointCloud floor: 3694 points.
Parameters of floor:
a: -0.0441042
b: -0.999026
c: -0.00126959
d: -1.95364
427 ms
```

Figura 32: Segmentación con filtrado

Si se muestran todas las nubes segmentadas de forma simultánea en la figura 33, se podrá observar una nube muy similar a la filtrada:

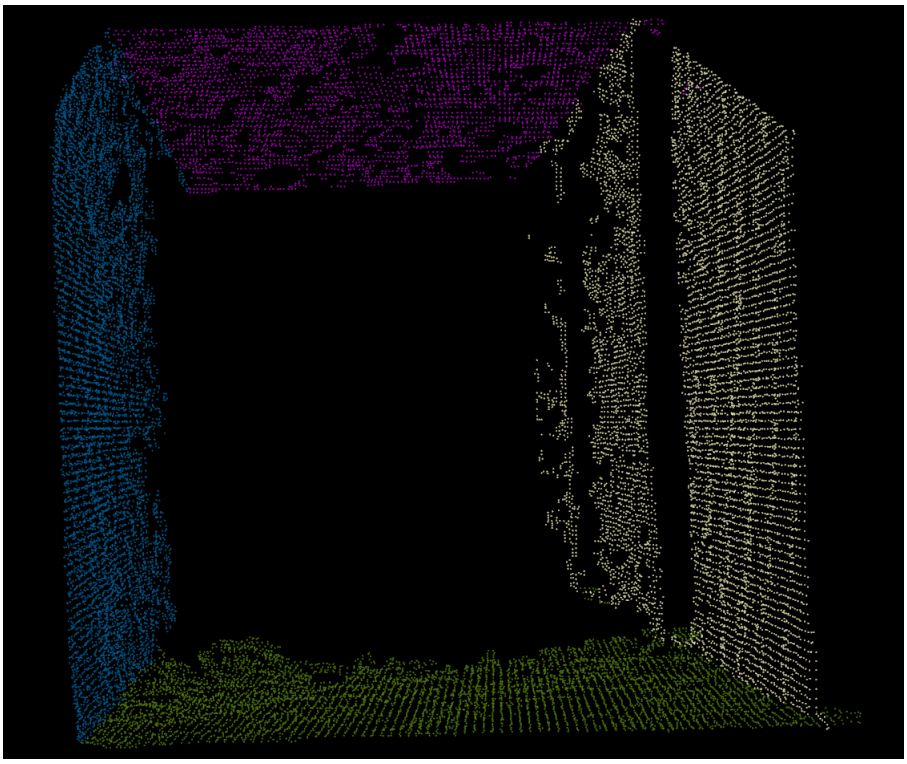

Figura 33: Nubes de puntos segmentadas

Para poder comprobar si los coeficientes de los planos son correctos, los representaremos utilizando un software matemático. Se ha intentado mantener los colores de la imagen anterior para ver más fácilmente la correspondencia de planos. El resultado es el mostrado en la figura 34:

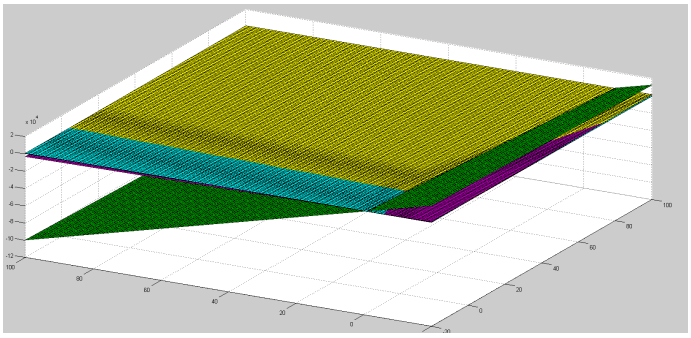

Figura 34: Conjunto de planos segmentados

Pese a que de forma bidimensional su visualización es compleja, su análisis de forma tridimensional nos da unos resultados satisfactorios.

Ahora se va a proceder a la segmentación sin filtrado para poder realizar una comparación. El resultado tras ejecutarlo nos devuelve 30 planos, y un tiempo de ejecución de 43512ms, una cifra muy superior a la anterior. Por ello, la mejor opción es filtrar previamente la nube. En la figura 35 se presentará el diagrama de flujo de esta sección, esquematizando todo el proceso explicado durante la misma.

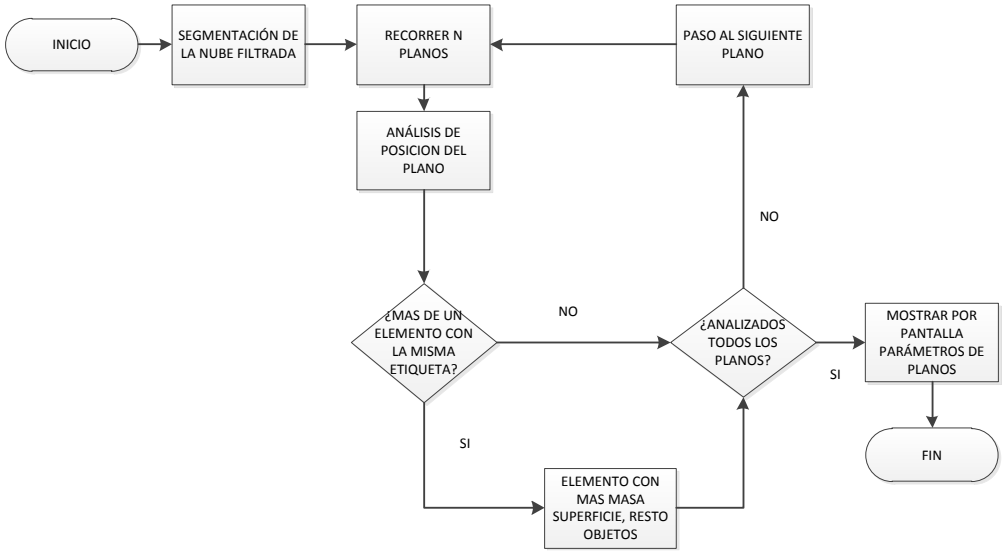

Figura 35: Diagrama de flujo del proceso de segmentación

5.4.1. Píxeles pertenecientes al plano del suelo

Una vez calculados los coeficientes de los planos de las superficies, es necesario conocer la ubicación en píxel del suelo. Para ello se parte de la nube original, que se encuentra ordenada, y de los coeficientes del plano que devuelve la segmentación de planos. Con todo eso, es posible conocer de una forma aproximada que píxeles pertenecen al suelo. La forma de lograrlo es utilizando la distancia punto a plano, estableciendo una distancia máxima a la que puede estar un punto para ser considerado suelo.

Para ello fueron realizadas varias pruebas estableciendo distintas distancias hasta encontrar, de forma empírica, el mejor resultado posible. Los datos que se extraen de la cámara tienen como unidad el metro, por lo que las distancias se proporcionarán en dicha unidad. La primera medida utilizada, ya que lo que se busca es precisión, es 1 centímetro de distancia. El resultado que obtenemos se puede ver en la figura 36:

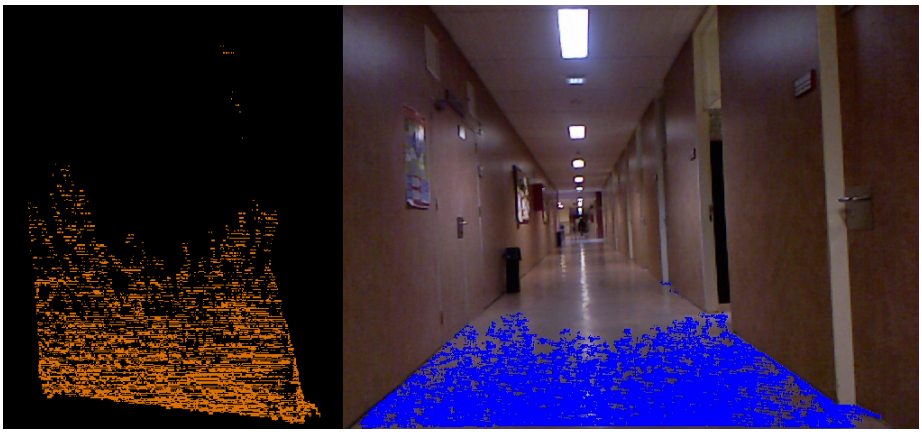

Figura 36: Puntos pertenecientes al suelo con distancia menor a 1 centímetro

Si se varía a 5 centímetros de distancia máxima, nos queda el resultado de la figura 37:

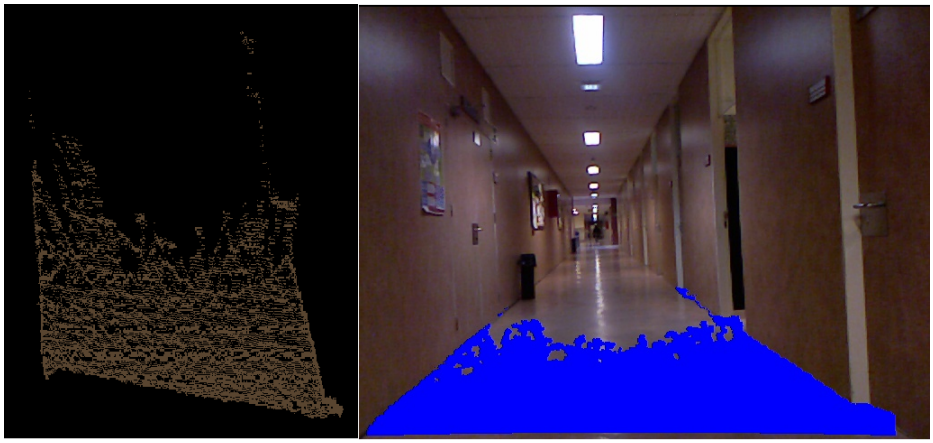

Figura 37: Puntos pertenecientes al suelo con distancia menor a 5 centímetros

Se puede apreciar que la nube que da como resultado mantiene una forma aproximada pero formada por más puntos, lo cual nos proporciona una precisión mayor. En este caso, una distancia menos restrictiva devuelve mejores resultados debido a que los coeficientes de un plano son una aproximación que intenta ser lo más ajustada posible, pero no abarcan de forma exacta todos los puntos que lo componen. Por este motivo, se utilizará un valor menos restrictivo aún para comprobar el resultado. En este caso, se ampliará el rango hasta los 10 centímetros, dando como resultado el mostrado en la figura 38:

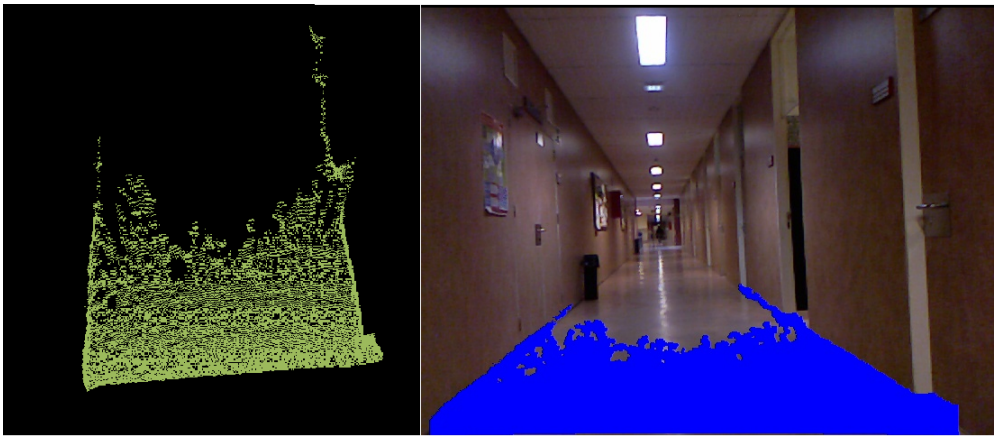

Figura 38: Puntos pertenecientes al suelo con distancia menor a 10 centímetros

En éste caso la nube posee aún más puntos, haciéndola más sólida. El mayor punto negativo sucede en los bordes, entendiendo como suelo un pequeño fragmento de pared, siendo aún así un error asumible.

Se puede apreciar el nivel de detalle sobre el que trabaja, incluyendo los pequeños recodos de las puertas, y de forma negativa, una pequeña parte de la pared. Aún así se debe lidiar con este problema ya que viene dado por la precisión de la cámara. En tiempo, este proceso es muy rápido, ya que se obtienen 52.812 índices almacenados, y se ha tardado 227 milisegundos en dicha tarea. Este tiempo se va a mantener prácticamente constante de forma independiente a la distancia elegida, ya que nunca se va a evitar pasar por todos los puntos y realizar un cálculo. Para aliviar este tiempo, se puede hacer que no pase por todos los puntos, pese a que se perderá precisión. En la figura 39 se muestra el diagrama de flujo correspondiente a esta sección.

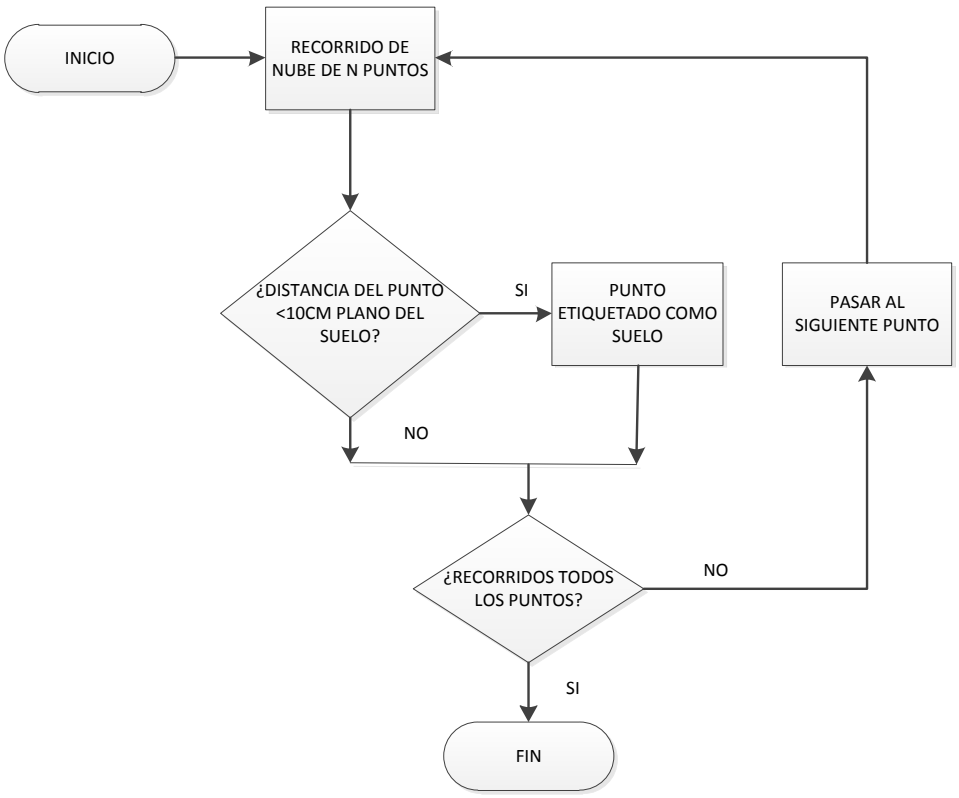

Figura 39: Diagrama de flujo del proceso de píxeles pertenecientes a un plano

5.4.2. Extracción de parámetros de color

Una vez etiquetados los píxeles que pertenecen al suelo, hay que comenzar a trabajar con los parámetros de color, tanto de la nube como de la cámara RGB. Para ello, se comienza analizando los parámetros de color de la nube, y para poder hacerlo, es necesario extraerlos de la nube antes.

Debido a la enorme cantidad de puntos que conforman la nube del suelo, no se mostrarán todos los datos de color. Aunque en el programa final no se muestre ninguno, a modo de ejemplo se mostrarán los colores de los 10 primeros puntos, además del tiempo empleado en almacenar todo. El resultado es el mostrado en la tabla 1:

	R	G	B
Iteración 1	13	10	23
Iteración 2	19	18	25
Iteración 3	56	53	53
Iteración 4	58	52	58
Iteración 5	53	47	56
Iteración 6	57	48	48
Iteración 7	57	51	53
Iteración 8	56	51	57
Iteración 9	57	49	45
Iteración 10	55	48	54

Tabla 1: Extracción de parámetros de color

Para todo ello, emplea 7 milisegundos en extraer el color. Al haber mostrado 10 puntos consecutivos, los parámetros de color son muy similares ya que pertenecen al mismo elemento.

5.5. Análisis de parámetros de color

Previamente se han guardado los datos de color de la nube de puntos, con el fin de analizarlos. En esta sección se tratará dicho análisis, y las opciones sobre las que se ha trabajado antes de llegar a la solución final.

Los estudiantes de la Universidad de Stanford, en su publicación[1], explican un método para conocer si unos puntos son similares a otros anteriores o se consideran diferentes. Para simplificar los cálculos, trabajan con gaussianas. Por tanto, nuestro algoritmo debe ser capaz de reducir toda la información RGB obtenida a funciones gaussianas. Para crear gaussianas que abarque información RGB de forma correcta se deben dividir los datos en agrupaciones de distintos colores mediante algún método de *clustering*.

El que mejor se ajusta a nuestras necesidades, recibe el nombre de *k-means* o *k-medias*. Aunque es un método explicado previamente en el estado del arte, se hará un breve recordatorio. En este tipo de algoritmo, se parte de un número de agrupaciones que se van a buscar, k, y el fin es etiquetar todos los datos en cada una de las agrupaciones.

El algoritmo *k-medias* es un algoritmo de agrupación muy extendido gracias a su sencillez, y por ello actualmente existen multitud de implementaciones. La más sencilla y eficiente a nivel de código, la explica Kardi Teknomo en su página web[15].

Para realizar este método, se comienza eligiendo los centroides. El sistema tendrá tantos centroides como agrupaciones se busquen. Inicialmente, se eligen como centroides datos aleatorios del sistema. Una vez elegidos, se calcula la distancia de cada punto a cada uno de los centroides, quedando como resultado una matriz de tantas filas como agrupaciones se busquen, y tantas columnas como da-

tos totales se tengan.

Una vez obtenida dicha matriz, el siguiente paso consiste en traducirla a una matriz binaria. en cada columna se estudiará cual es la distancia menor, lo que significa que el dato pertenece a esa agrupación, y se sustituye su posición por un 1, mientras que el resto de elementos de la columna serán 0.

Con la matriz binaria completa, se recalculan los centroides. Se analizará fila por fila, analizando las posiciones en las que aparezca un 1 y sumando sus datos. Cuando se acabe, se divide entre el número de datos que se han sumado en cada fila, obteniendo así tres centroides situados en el punto medio de todos los datos anteriores.

Con los nuevos centroides se repite el proceso anterior las veces necesarias hasta que se repita dos veces consecutivas la misma matriz binaria. En ese momento se considera que la agrupación está terminada y tenemos así etiquetados todos los puntos en sus respectivas agrupaciones.

Una vez se han obtenido las agrupaciones, es cuando se obtienen los parámetros de las gaussianas. De nuevo, en el estado del arte se encuentra información más detallada, por lo que aquí solo se remarcarán los datos necesarios. Una distribución gaussiana puede definirse utilizando su media y su varianza. Para calcular su media, μ, la fórmula aplicada es:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} X_i \tag{24}$$

Mientras que la varianza, σ^2 queda definida por:

$$\sigma^2 = \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} (x_i - \overline{x})^2 \tag{25}$$

Con ello podemos calcular los parámetros que buscamos. Además de dichos parámetros, se guardará también la masa (número de puntos), que será requerida en el futuro. El resultado del conjunto, tanto de agrupación como cálculo de parámetros gaussianos, usando una k=3, se puede ver en la tabla 2:

	R	G	B
Medias 1	122.773	109.447	122.984
Varianzas 1	249.702	278.566	262.49
Masa 1	309		
Medias 2	81.9109	62.0661	62.4672
Varianzas 2	30.2974	31.2947	77.3858
Masa 2	3221		
Medias 3	94.288	78.1837	85.6717
Varianzas 3	35.3101	30.9852	60.289
Masa 3	309		

Tabla 2: Agrupaciones y reducción a parámetros gaussianos con k=3

En este caso, necesita 27 iteraciones para poder etiquetar todos los puntos en las tres agrupaciones buscadas, empleando 78 milisegundos para realizar la agrupación, mientras que solo necesita 1 milisegundo para hacer el cálculo de los parámetros gaussianos. Pese a que los tiempos anteriores se van a mantener relativamente constantes, esta etapa variará mucho al calcular las iteraciones. Dependiendo de lo diversificado que se encuentren los valores puede costar más o menos esfuerzo encontrar la agrupación más óptima. Como prueba, se mostrará el resultado junto con una imagen del pasillo. Pese a que es complicado ajustar el resultado a la imagen, es válido para poder ver aproximadamente el resultado en la figura 40. Se puede apreciar como etiqueta de distinto modo el brillo y las sombras con respecto al suelo de color aproximadamente uniforme.

Figura 40: Resultados con k=3

Si se compara con los datos numéricos obtenidos, se pueden ver dos nubes de puntos muy abundantes en número, que se corresponden con los puntos amarillo y naranja, representando la zona grisácea y la neutral, y una tercera muy pequeña en la zona central, que corresponde con los brillos que aparecen en el suelo del pasillo. Esa gaussiana puede causar problemas, debido a su alta varianza, lo que implica una gran dispersión, lo cual otorga facilidad para que un punto se encuentre dentro de dicha gaussiana. Para poder obtener un número óptimo de parámetros gaussianos a guardar, se harán comprobaciones con distintos números para ver como evoluciona. A continuación se van a mostrar los resultados con diferentes valores para el número de agrupaciones, k. La diferencia, como se puede ver en las tablas 3 y 4 y las figuras 41 y 42, se da por los colores de los distintos puntos que conforman la nube.

	R	G	B
Medias 1	94.44995	78.4106	85.5494
Varianzas 1	26.0713	14.3051	30.0162
Masa 1	2126		
Medias 2	78.3617	57.7956	55.1664
Varianzas 2	26.9732	22.715	52.5128
Masa 2	1424		
Medias 3	85.3727	66.4589	69.8779
Varianzas 3	16.4307	15.5719	30.5124
Masa 3	2227		
Medias 4	106.464	92.326	105.536
Varianzas 4	48.6352	39.0799	58.0299
Masa 4	457		
Medias 5	137.579	125.357	137.913
Varianzas 5	190.342	225.943	218.704
Masa 5	126		

Tabla 3: Agrupaciones y reducción a parámetros gaussianos con k=5

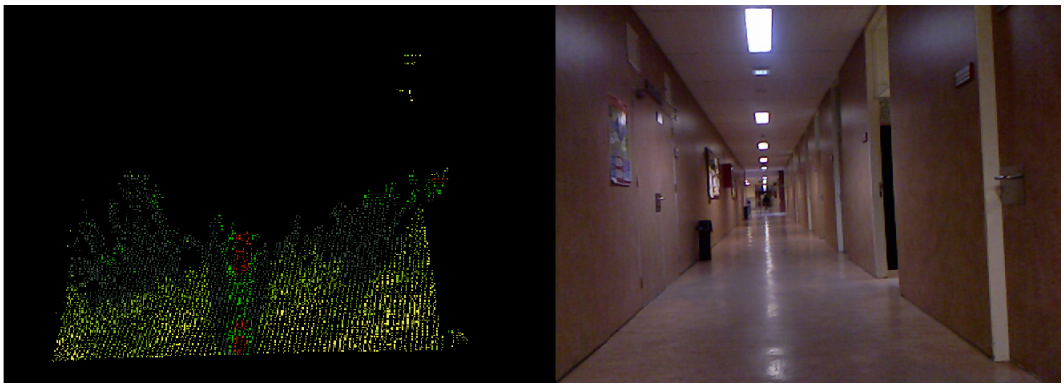

Figura 41: Resultados con k=5

	R	G	B
Medias 1	103.747	89.8153	102.863
Varianzas 1	22.5282	19.3415	30.0392
Masa 1	379		
Medias 2	84.578	65.1195	67.5311
Varianzas 2	14.7102	11.616	19.0021
Masa 2	1640		
Medias 3	96.6403	80.313	87.8058
Varianzas 3	19.7799	7.41025	21.6037
Masa 3	1390		
Medias 4	120.755	106.623	120.775
Varianzas 4	48.0796	53.4366	63.0423
Masa 4	151		
Medias 5	148.344	136.859	148.047
Varianzas 5	117.689	144.408	191.918
Masa 5	64		
Medias 6	79.1998	58.8115	57.6655
Varianzas 6	15.4957	11.2366	16.9046
Masa 6	1151		
Medias 7	89.1399	72.649	78.7806
Varianzas 7	14.8729	12.0069	18.6846
Masa 7	1322		
Medias 8	74.2167	52.9658	44.2624
Varianzas 8	54.5903	42.8652	66.0187
Masa 8	263		

Tabla 4: Agrupaciones y reducción a parámetros gaussianos con k=8

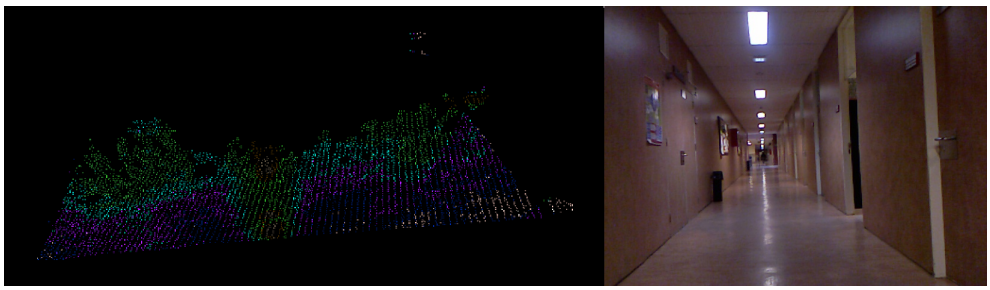

Figura 42: Resultados con k=8

Como se puede observar, a mayor número de agrupaciones, menor número de puntos en cada una de ellas, llegando a casos en los que una agrupación la componen un número de puntos insignificante en función de los demás. Esto a su vez genera casos peligrosos, creando gaussianas con una dispersión muy alta que puede inducir a error en pasos futuros. Además, el tiempo para calcular las agrupaciones se ve notablemente incrementado, ya que cada vez es más fácil cambiar de agrupación. Por eso, una buena elección del número de agrupaciones en cada situación es fundamental para un buen resultado. El software está programado

para que el número de agrupaciones sea un parámetro dentro de la función principal, y solo con cambiarlo todo el programa actúa en consecuencia. En la figura 43 se muestra el diagrama de flujo correspondiente a esta sección.

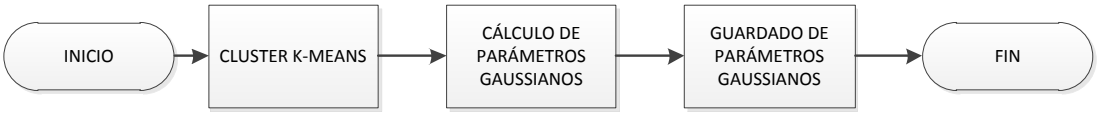

Figura 43: Diagrama de flujo del proceso de clustering

5.5.1. Cálculo de matrices de covarianza

Para todos los cálculos posteriores será necesario el uso de matrices de covarianza, por lo que en este paso se procederá a comentar la función diseñada para ello.

Como recordatorio, cada uno de los elementos de la matriz de covarianza se definen como:

$$\sigma_{i,j}^2 = \frac{\Sigma_{i=1}^{n}(X_i - \overline{x})(Y_i - \overline{y})}{n-1} \tag{26}$$

Según esto, la diagonal principal se encuentra calculada ya, puesto que se corresponde al valor de la varianza, por lo que solo será preciso calcular el resto de huecos de la matriz. Estos cálculos son necesarios para todas las matrices. El resultado es el mostrado en la figura 44:

```
Covariances 1:
 249.702 252.975 222.935
 24.1976 278.566 23.5696
 24.2714 26.8272 262.49

Covariances 2:
 30.2974 263.48 245.821
 25.2025 31.4333 32.4356
 26.7632 36.9185 77.3858

Covariances 3:
 35.3101 234.653 193.671
 22.4451 30.9852 27.9751
 21.0854 31.8416 60.289
2 ms
```

Figura 44: Matriz de covarianzas

Es un proceso muy rápido, ya que dura tan solo 2 milisegundos.

5.5.2. Aprendizaje de gaussianas

En este paso se realizará el aprendizaje de gaussianas. A nivel conceptual, se puede hablar de que el sistema *aprende* terrenos. Con el sensor de la cámara se logra un conocimiento del terreno casi total, pero es demasiado lento como para depender únicamente de ello, por lo que también se realiza el análisis del color. Pero al funcionar en tiempo real, el sistema no deja de recibir nubes de puntos nuevas, y clasificarlas. Debido a las pequeñas variaciones que existen en los colores que componen el terreno (el más mínimo brillo o la más pequeña sombra lo alteran), es prácticamente imposible que dos procesos de agrupaciones generen los mismos parámetros gaussianos de forma exacta.

Por ello, una vez calculadas las gaussianas nuevas y su matriz de covarianza, imprescindible para hacer los cálculos tanto de esta sección como de futuras, es posible conocer si una gaussiana es muy similar a otra. Utilizando la teoría expuesta por los estudiantes de Stanford[1], podremos saber si una gaussiana nueva (T) es muy similar a una aprendida (L) si cumple lo siguiente:

$$(\mu_L - \mu_T)^T(\Sigma_L + \Sigma_T)^{-1}(\mu_L - \mu_T) \leq d \tag{27}$$

Siendo d la distancia máxima a la que se pueden encontrar las gaussianas en función de la precisión. Dicha operación consiste en el empleo de una distancia de Mahalanobis tipificada. En caso de cumplirse dicha afirmación, se actualizarán los parámetros de la gaussiana aprendidos, siguiendo la siguiente ecuación:

$$\mu_L \leftarrow \frac{m_L\mu_L + m_T\mu_T}{m_L + m_T} \tag{28}$$

$$\Sigma_L \leftarrow \frac{m_L\Sigma_L + m_T\Sigma_T}{m_L + m_T} \tag{29}$$

$$m_L \leftarrow m_L + m_T \tag{30}$$

Como se puede apreciar, la actualización depende de la masa, es decir, el número de puntos, logrando así que cuanto mayor sea la masa, menos varíe la gaussiana aprendida.

En caso de no cumplir la igualdad, se realizaría otra operación. Para descartar gaussianas resultantes de casos aislados o situaciones muy concretas, como un exceso de luz en una zona pequeña, se evalúa su masa. Si es superior al 30 % de la masa de la gaussiana superior, se aprende una gaussiana nueva. El número máximo de gaussianas aprendidas se determina al principio del programa. En caso de que se haya llenado el vector, se compara la masa de la nueva gaussiana que se quiere aprender, y en caso de ser mayor que la gaussiana aprendida con menos puntos se eliminará dicha gaussiana para aprender la nueva. Así se evita mantener como gaussiana aprendida elementos que aparecen en momentos circunstanciales. Si lo realizamos obtenemos un proceso muy rápido, como se ve ha empleado un tiempo cercano a 0 milisegundos para realizar los cálculos. En este caso, como las gaussianas que han sido obtenidas son muy diferentes entre sí no ha actualizado ninguna y ha aprendido las tres como gaussianas nuevas. Este proceso se verá de

forma más clara cuando se experimente con un sistema dinámico. En la figura 45 se muestra el diagrama de flujo correspondiente a esta sección.

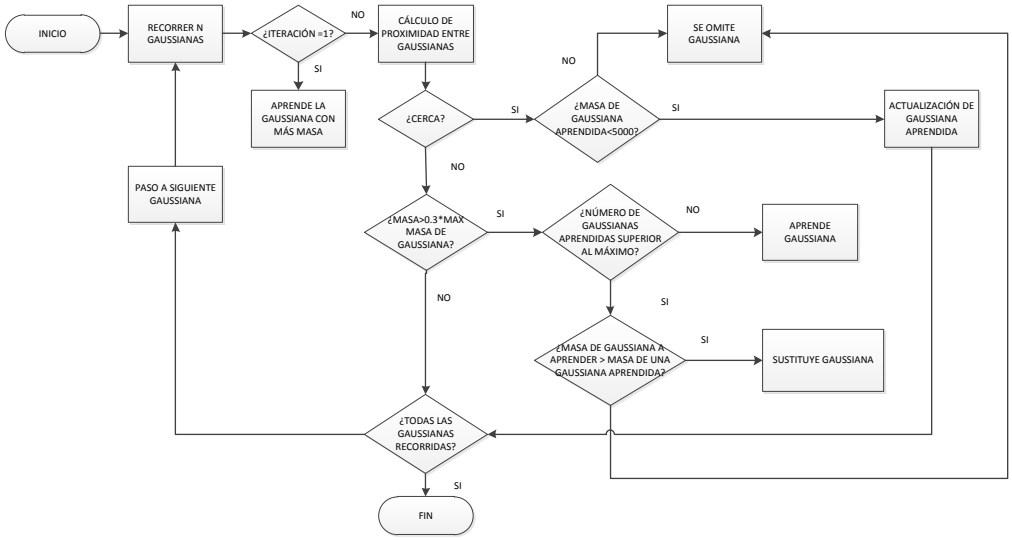

Figura 45: Diagrama de flujo del proceso de aprendizaje

5.6. Región de interés

En esta sección se van a analizar los pasos necesarios para poder hallar la región de interés de la nube, entendiendo como región de interés toda la zona que pertenece al suelo. El cálculo de una región de interés es necesario para poder separar los píxeles de la imagen RGB que pertenecen al suelo de los que forman parte de otra superficie.

A nivel conceptual, este paso es muy sencillo. En este punto tenemos los coeficientes que definen el plano que contiene al suelo, pero en tres dimensiones. Se comienza definiendo la recta intersección entre el plano del suelo y los dos de las paredes, dando como resultado dos rectas. Una vez se han calculado las rectas, se han de proyectar en la imagen RGB bidimensional.

Entrando más a fondo, es imprescindible trabajar con la nube original. Como se dijo anteriormente, la nube da los datos ordenados por píxel, mientras que tras el filtrado este orden se pierde. Por eso volvemos a encontrarnos con dos caminos. En ambos hay que comenzar calculando la recta intersección entre los planos, utilizando para ello el método matemático programado en el lenguaje C++. Al tener los puntos del plano desordenados, es necesario calcular la distancia de cada punto al plano del suelo y etiquetar cuales pertenecen a él.

Antes de comenzar, se mostrará el resultado de las rectas como intersecciones de planos, a modo de comprobación de su funcionamiento. El resultado del programa se muestra en la figura 46:

```
Equations of straigh 1:
x(t) = -1.85773 + -0.0916668t
y(t) = 1.85773 + 0.0294891t
z(t) = 33.5816 + 0.995252t

Equations of straigh 2:
x(t) = -0.93405 + -0.122548t
y(t) = 0.93405 + 0.0308545t
z(t) = -1.21381 + 0.991801t
0 ms
```

Figura 46: Resultado del cálculo de rectas de intersección

Como se puede observar, el tiempo que necesita para calcular dichas rectas son 0 milisegundos, por lo que es un proceso tremendamente veloz. Las rectas se representan por separado en las figuras 47 y 48 junto a los planos que la forman:

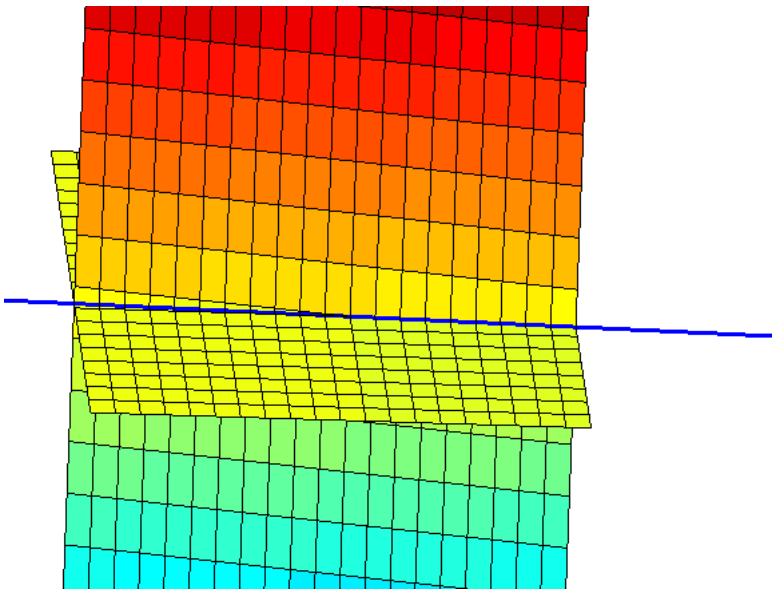

Figura 47: Intersección entre el suelo y la pared izquierda

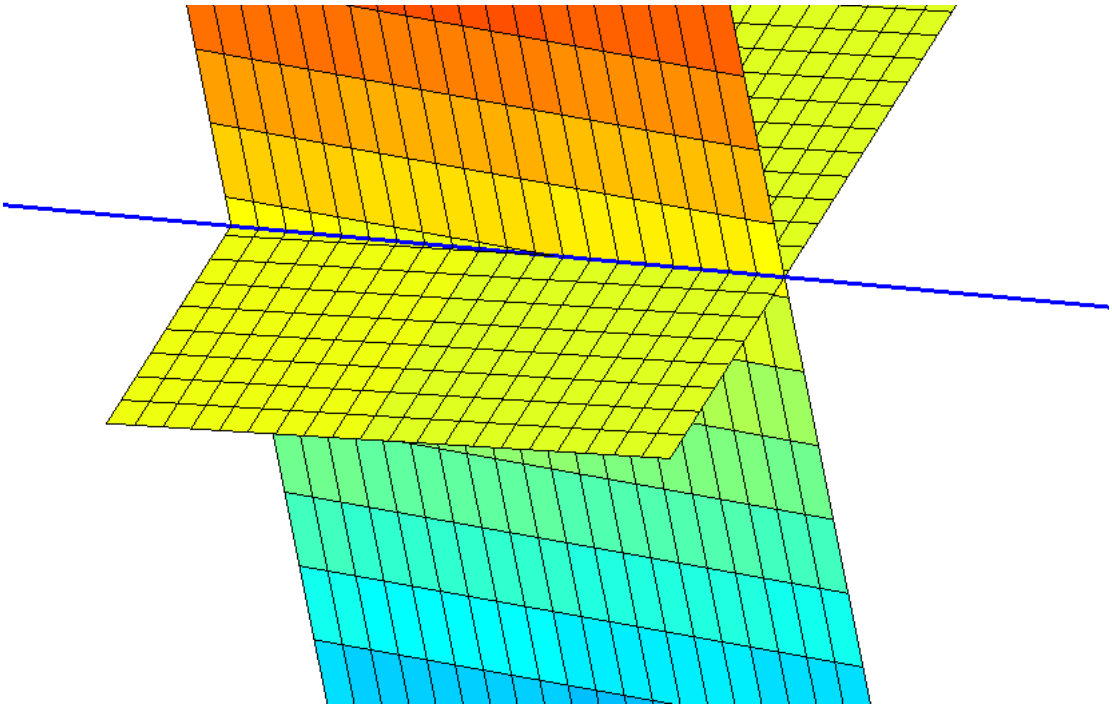

Figura 48: Intersección entre el suelo y la pared derecha

Como se puede apreciar, la recta se coloca exactamente en la intersección entre planos, quedando un modelo muy ajustado.

Una vez calculadas las rectas intersección de planos, se realiza la transformación a 2 dimensiones. Para poder comprobar si su funcionamiento es correcto, se dibujará sobre la imagen RGB en la figura 49 para ver si se corresponde:

Figura 49: Representación de rectas

Como se puede observar, el comportamiento es casi perfecto, ajustando de forma muy correcta la recta del suelo. Una vez calculadas las rectas, es posible conocer que píxeles de la imagen pertenecen al suelo. Este proceso no es más que una estimación, puesto que las rectas no abarcan todo el suelo, como por ejemplo recovecos que pueden existir, y porque pueden coger parte de la pared del fondo con la parte final del triángulo. Ese último problema, como se verá en próximas secciones, no afectará al programa dado que la pared no compartirá color con el suelo por regla general.

Para conocer los píxeles pertenecientes al suelo se realizarán varios procesos. El primero consiste en calcular la intersección de las rectas. Si dicha intersección se sucede dentro de la imagen, se cuentan los píxeles que se encuentran desde la parte de abajo de la imagen hasta dicha intersección. En caso se darse fuera, se cuentan todos. Como resultado, se guardan 64887 píxeles de la imagen en total, y tarda 24 milisegundos. Este proceso no se puede agilizar ya que es preciso recorrer todos los píxeles de la imagen si se busca que funcione con precisión. Reflejado en la imagen RGB se obtiene la figura 50:

Figura 50: Píxeles guardados

Se puede apreciar que trabaja de forma perfecta, rellenando completamente el espacio entre ambas rectas.

5.7. Etiquetado del suelo

En este momento es donde se engloba todo lo realizado anteriormente para poder ampliar el rango de la Kinect. Anteriormente, junto a la región de interés, se han etiquetados qué píxeles se encuentran dentro del área que delimitan las dos rectas intersección que forman los planos de las paredes con el suelo y se han extraído sus parámetros de color.

Para poder conocer si un punto de color se engloba dentro de una distribución gaussiana, es necesario calcular la distancia a la que se encuentra de la distribución, y para ello, en estadística multivariante lo más óptimo es usar la distancia de Mahalanobis. Esta distancia proporciona los parámetros más óptimos debido a que no tiene en cuenta solo el valor medio, sino la matriz de covarianza.

La fórmula para conocer la distancia de Mahalanobis es la siguiente:

$$M(x) = \sqrt{(x-\mu)^T \Sigma^{-}1(x-\mu)} \tag{31}$$

Siendo M la distancia a la que se encuentra. Esta distancia tenderá a 0 cuanto más cerca se encuentre de la distribución gaussiana, por lo que hay que ir probando empíricamente distintas distancias hasta encontrar una óptima. Para facilitar el proceso, se guardan previamente las distancias de Mahalanobis de todos los puntos,

para poder probar de un modo aproximado. Por ello, se da como punto de partida una distancia de 5 máxima, y se probará posteriormente con otro inferior, de 2.7, y finalmente una distancia de 1. Los resultados se muestran en la tabla 5 y las figuras 51, 52 y 53:

	Mahalanobis 5	Mahalanobis 2.7	Mahalanobis 1
Cluster 1	7841	6695	999
Cluster 2	34770	32420	5992
Cluster 3	21899	20645	3985

Tabla 5: Píxeles analizados

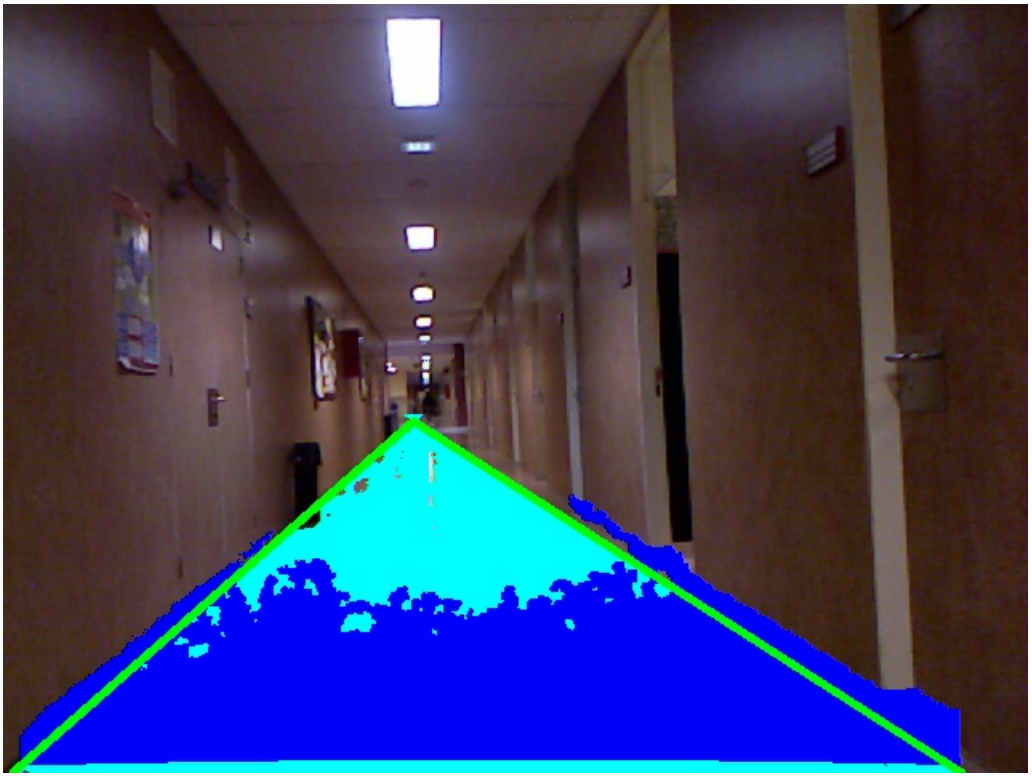

Figura 51: Representación de píxeles analizados con distancia 5

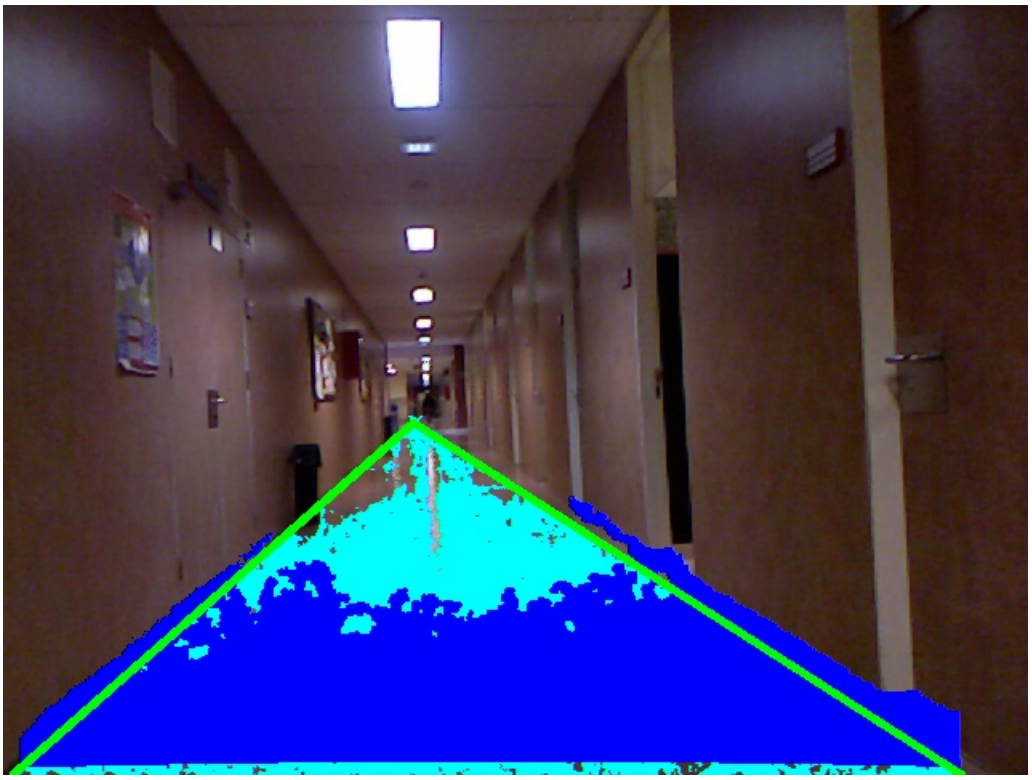

Figura 52: Representación de píxeles analizados con distancia 2.7

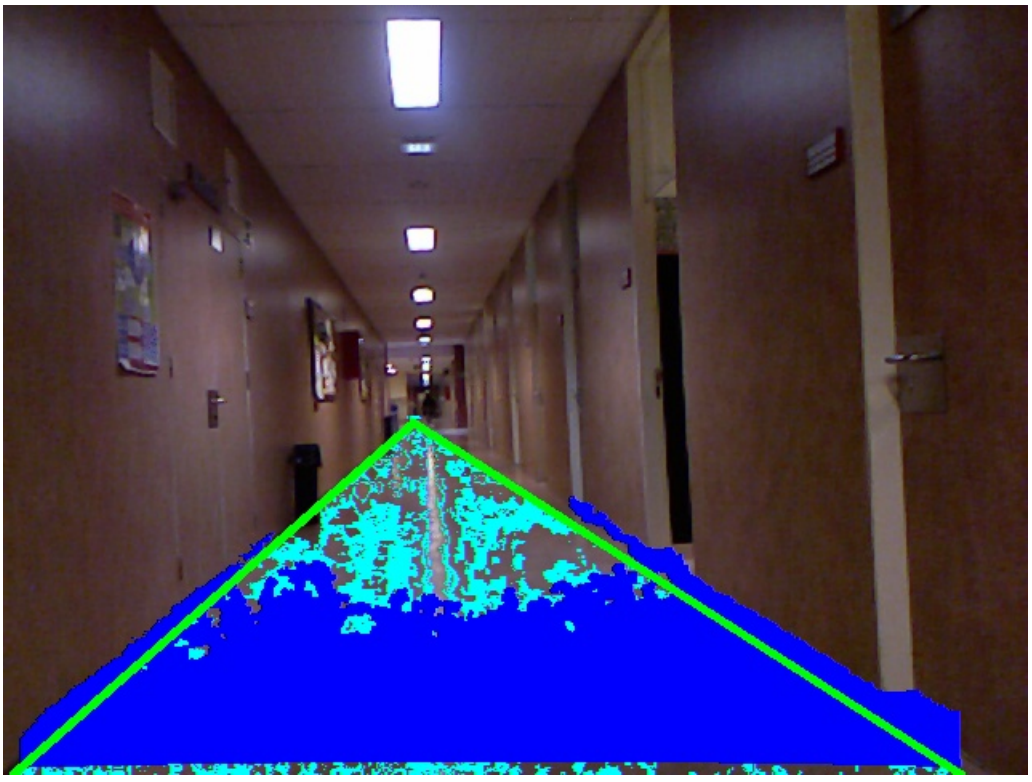

Figura 53: Representación de píxeles analizados con distancia 1

De una distancia de 5 a una de 2.7, el número de píxeles no ha bajado

considerablemente, y en la imagen RGB sigue siendo casi imperceptible el cambio. En cambio en el caso de distancia 1 la diferencia es más notable, dejando grandes porciones de suelo sin etiquetar, por lo que se utilizará 2.7 como distancia máxima. En la figura 54 se muestra el diagrama de flujo correspondiente a esta sección.

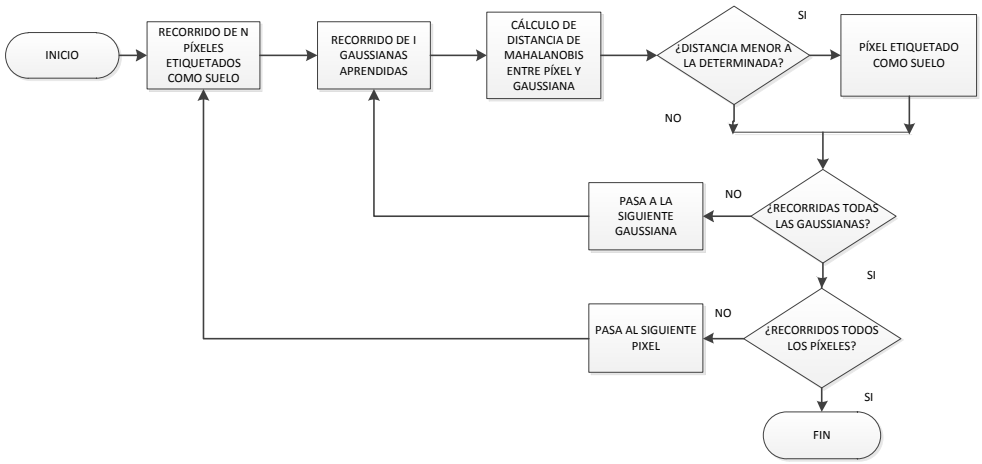

Figura 54: Diagrama de flujo del proceso de distancia de Mahalanobis

5.8. Generación de mapa

En este último paso se engloba todo lo que se ha realizado a lo largo del proyecto para poder, así, generar un mapa vista de pájaro del terreno, conociendo las distancias reales a las que se encuentra cada píxel catalogado como transitable.

Para ello se partió del modelo *pinhole*. Conociendo los parámetros de calibración de la lente y la distancia focal nos era posible conocer cuál era la distancia real de cada punto. El punto negativo que se encontró para este método fue la gran pérdida de precisión a medida que aumentaba la distancia. Si bien en distancias cortas su error era muy pequeño, en distancias largas aumentaba de forma considerable, errando las distancias en más de 20 metros en los puntos más alejados.

Por ese motivo se cambió el enfoque. Se estudió la distancia entre puntos teniendo en cuenta sus píxeles, y se pudo comprobar que si bien la relación entre píxeles y distancias no era lineal, sí que mantenían algún tipo de relación.

Esta relación viene dada por la relación entre distancias y área del triángulo que engloba el suelo. A medida que los puntos se alejan, el área de los triángulos era menor, según se puede ver en la figura 55:

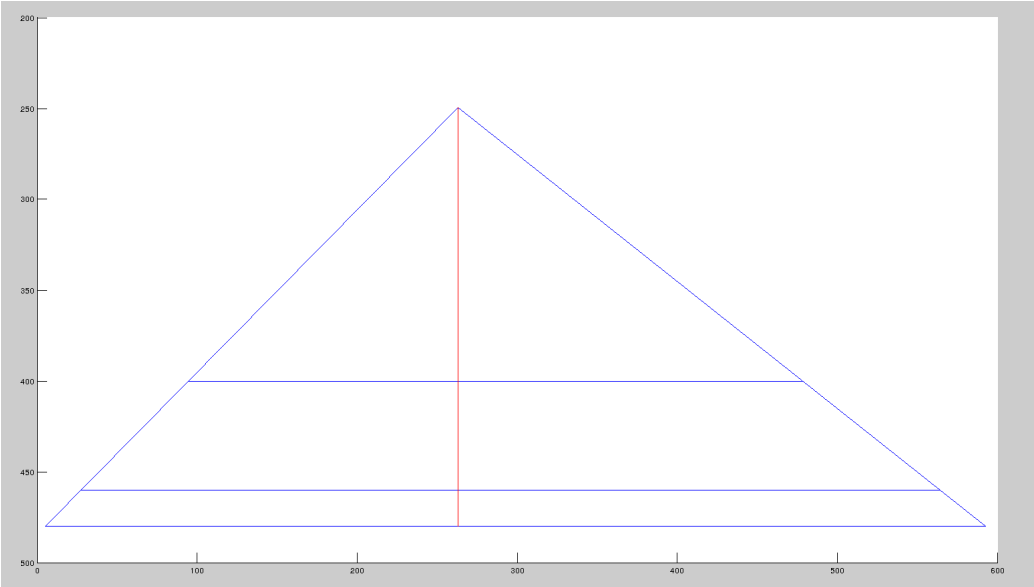

Figura 55: Área de triángulos

Como se puede ver, a medida que el punto se encuentra más lejos, el área disminuye, por lo que podríamos establecer la siguiente relación:

$$distancia \propto \frac{1}{area}$$

Matemáticamente, esta ecuación lleva a error, ya que el punto más lejano tendrá un área nula, haciendo que la relación tienda a infinito. Dicho error se tendrá que tener en cuenta en el futuro.

Para esta sección se utilizará el software matemático *Matlab*, que facilitará los pasos necesarios. Se comenzará salvando del programa en un fichero los puntos que pertenecen al suelo dentro del área de la Kinect y los píxeles que ocupan. Además de eso, se guardará en otro fichero todos los píxeles etiquetados como suelo.

Una vez con dichos ficheros, se realizará un proceso de ajuste de una curva que relacione la distancia en píxeles con la real. La curva que más se ajusta a este modelo es la curva racional de grado 1, como se puede observar en la figura 56:

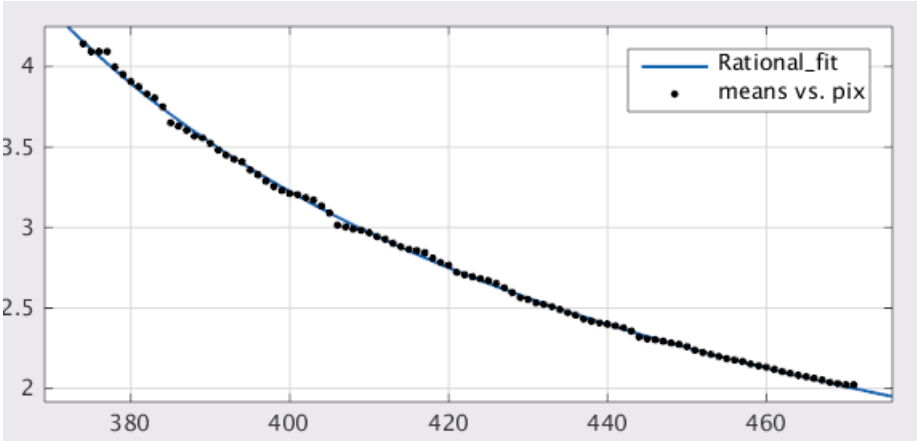

Figura 56: Curva de ajuste

Una vez ajustada la curva y obtenidos los parámetros de ajuste, se les aplica a todos los píxeles etiquetados como suelo dichos parámetros, dando como resultado lo mostrado en la figura 57:

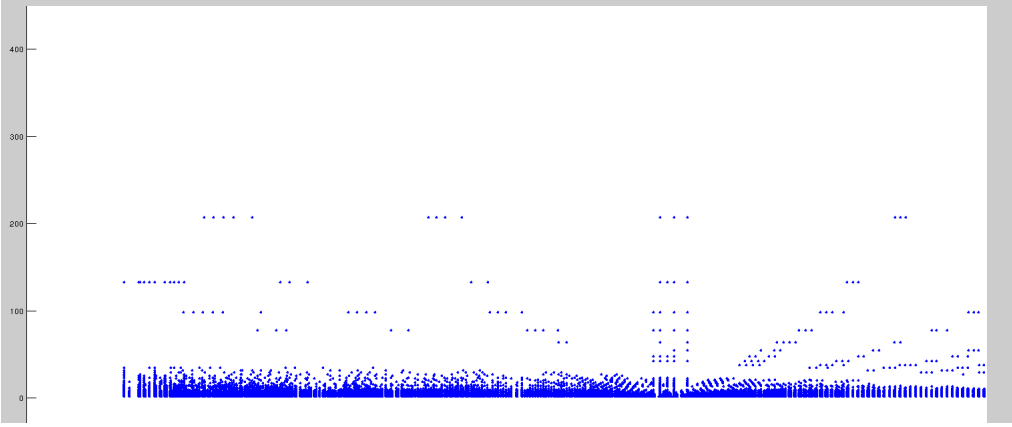

Figura 57: Mapa vista de pájaro sin filtrar

Como se puede observar, existe una gran acumulación de puntos en el centro, y numerosos puntos dispersos. Eso se debe al error comentado antes, que hace que las distancias tiendan a infinito. Por ese motivo, y tras examinar el rango en el que funciona correctamente este algoritmo, se ha limitado la distancia máxima a 30 metros, arrojando como resultado la figura 58:

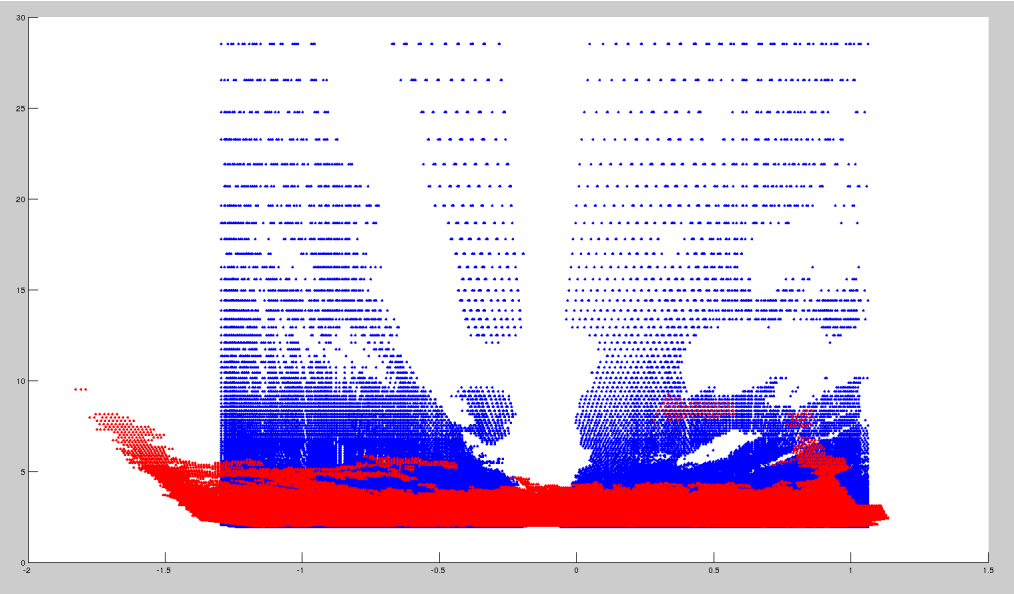

Figura 58: Mapa vista de pájaro

Los puntos rojos representan la visión de Kinect, mientras que los azules representan los píxeles etiquetados como suelo. Pese a que intuitivamente se esperan contornos más definidos, la lente de la Kinect realiza una proyección con forma de tetraedro, lo cual provoca que los elementos parezcan deformados.

Para agilizar todos estos pasos se ha creado un *script* en Matlab que automatiza todos los pasos, dando como resultado el mapa buscado. Con ello se ha pasado de 5 metros de rango a 30, logrando así un aumento del rango en un 600 %. Con esto se da por finalizado el algoritmo.

6. Funcionamiento del algoritmo con la cámara Kinect

Conociendo el funcionamiento del algoritmo estático y las distintas posibilidades existentes para lograr un funcionamiento satisfactorio en cada una de las secciones, ahora se pasará a analizar el funcionamiento del algoritmo funcionando de forma dinámica.

6.1. Problemas que surgieron en el funcionamiento dinámico

Al comenzar a trabajar en el sistema dinámico surgieron varios problemas. El primero venía derivado de la utilización de iluminación artificial en el interior. La iluminación artificial tiene un funcionamiento de 50Hz, por lo que la luz no es constante. Aunque eso es prácticamente inapreciable para el ojo humano, la cámara capta un frame, por lo que cabe la posibilidad de que el frame que capte sea más oscuro. Eso causa problemas en el etiquetado, ya que un frame oscuro tras varios claros hace que las gaussianas aprendidas sean diferentes a la nueva y estropee el funcionamiento general. Por tanto, la iluminación natural es más recomendable si se busca un funcionamiento perfecto.

Por otro lado, los suelos no mates generan multitud de brillos que pueden provocar, a su vez, problemas en el etiquetado de suelo, como se verá en las pruebas siguientes. Estos problemas surgen especialmente si no hay brillos en el rango de la Kinect, pero sí en el de la cámara RGB, ya que una vez que se aprenda una gaussiana que represente los brillos el error cesará.

La actualización de gaussianas inicialmente fue causa de errores, por dos motivos que iban de la mano. Primeramente, la condición utilizada para determinar si dos gaussianas eran iguales resultó ser excesivamente permisiva. Eso hacía que al actualizar la gaussiana acabase resultando una gaussiana tan dispersa que abarcase todas las demás. Por tanto, de forma empírica se ajustó la condición para actualizar, y se tomó la decisión de saturar la gaussiana una vez llegada a una masa de 5000 puntos, impidiendo así su crecimiento excesivo.

Finalmente, la distancia de Mahalanobis es preciso actualizarla en función del terreno para su correcto funcionamiento. En las pruebas posteriores se podrá comprobar como una distancia de Mahalanobis que resulta apta en un caso no lo es para otro tipo de suelo.

6.2. Pruebas en funcionamiento dinámico

En esta sección se mostrará el comportamiento del algoritmo en un caso dinámico, obteniendo los datos de la cámara Kinect y procesándolos en tiempo real.

Una consideración muy importante corresponde a la distancia de Mahalanobis. Por problemas relacionados con el espacio de color RGB, una distancia correcta en

un lugar no tiene por qué ser válido para otro. Como ejemplo, se mostrará el resultado de la distancia de Mahalanobis calculada como correcta en la fase estática. Se obtiene lo visto en la figura 59:

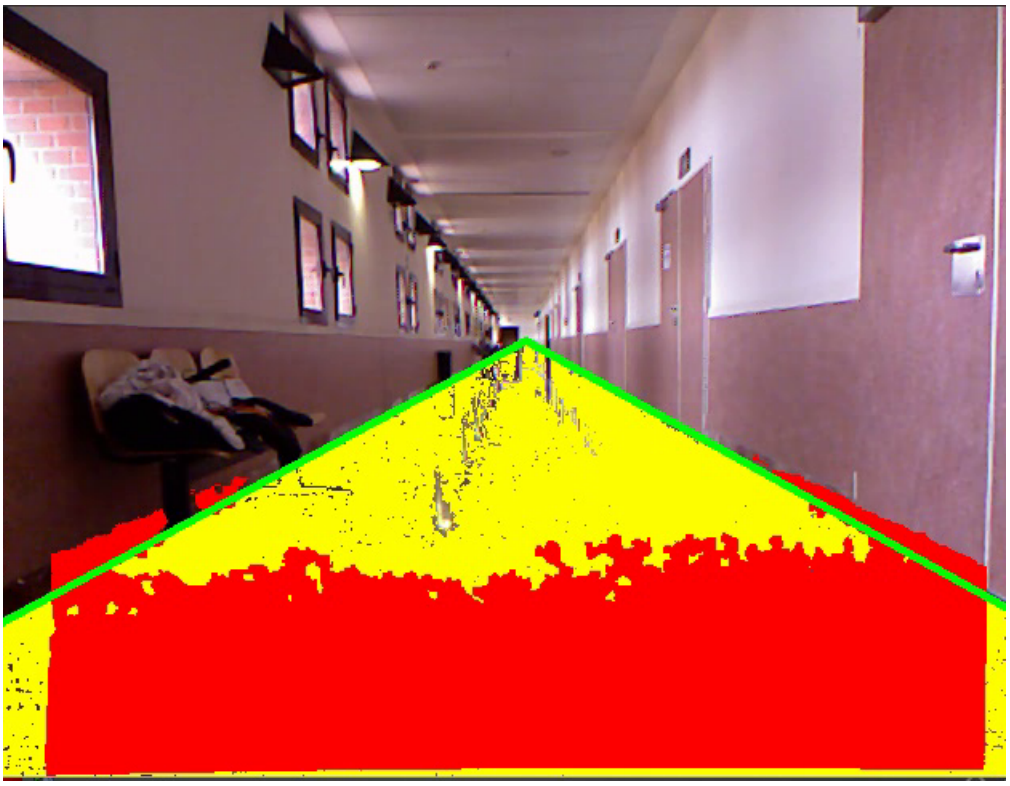

Figura 59: Pasillo con distancia de Mahalanobis 2.7

Como se puede apreciar, toma prácticamente todo como suelo, incluidas las patas de las sillas o la papelera del fondo pese a ser de un color radicalmente opuesto. Por eso es preciso hacer la distancia más restrictiva en este caso, obteniendo la figura 60:

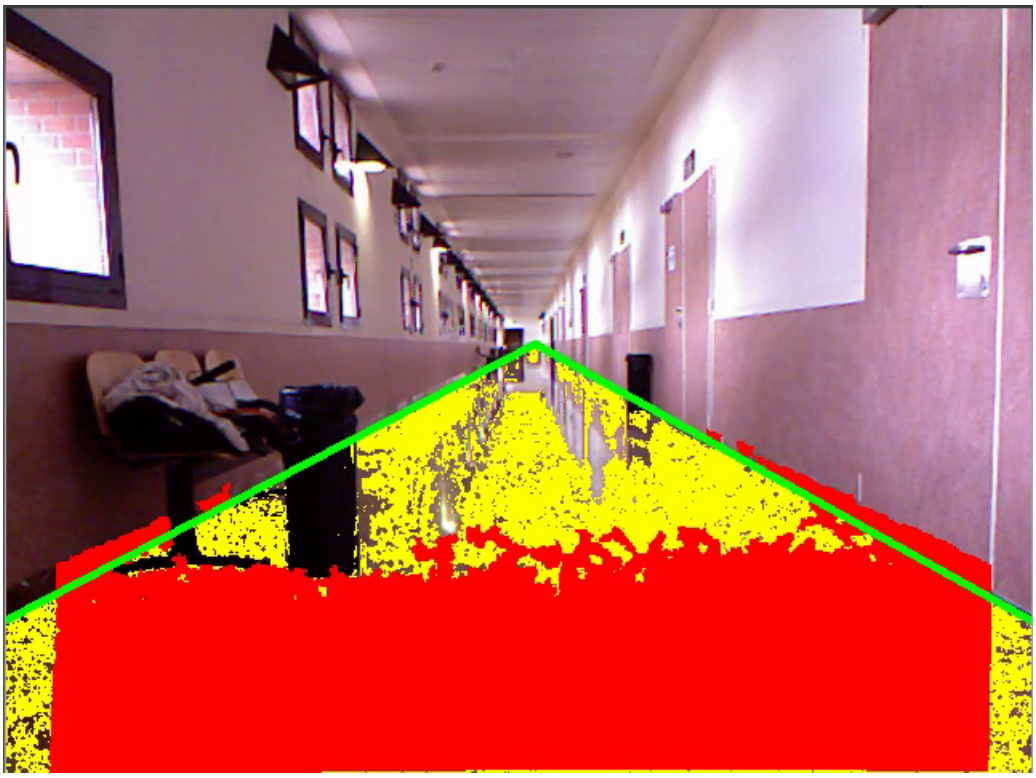

Figura 60: Pasillo con distancia de Mahalanobis 2

En este caso es posible apreciar como evita tanto los objetos que se encuentran en medio como los brillos producidos por la luz y los reflejos. Ee esta información se genera el mapa de vista de pájaro (figura 61):

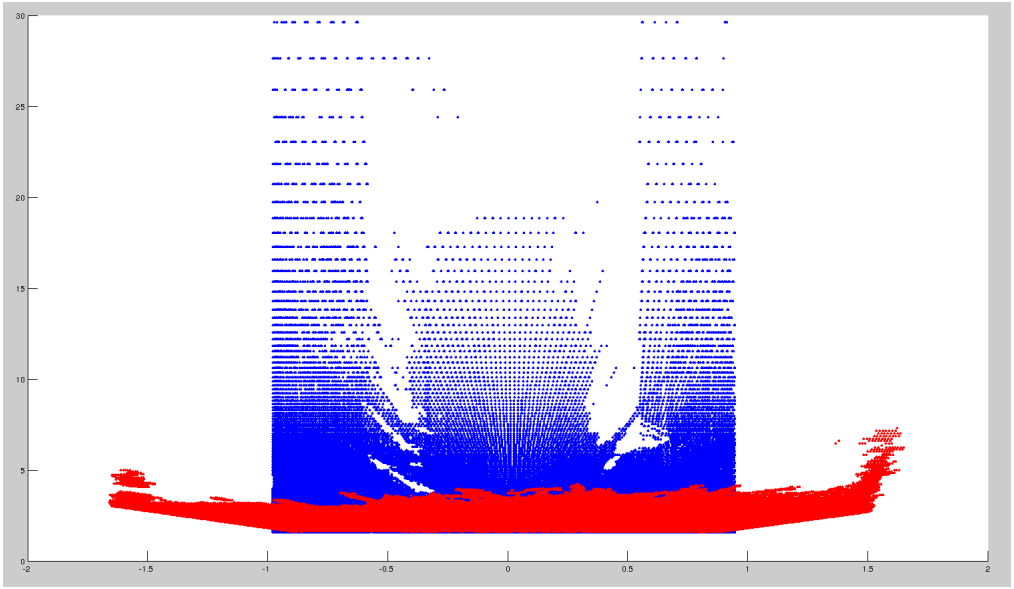

Figura 61: Mapa vista de pájaro

En él se puede ver como la zona central corresponde con la zona amarilla y como los obstáculos producen espacios blancos. Además, debido a que la distancia

real no es lineal con los píxeles, el espacio es cada vez mayor, tiendo inicialmente zonas muy rellenas y a medida que la distancia aumenta los puntos están más separados.

Trasladando la misma distancia de Mahalanobis a otro pasillo, el resultado es la figura 62:

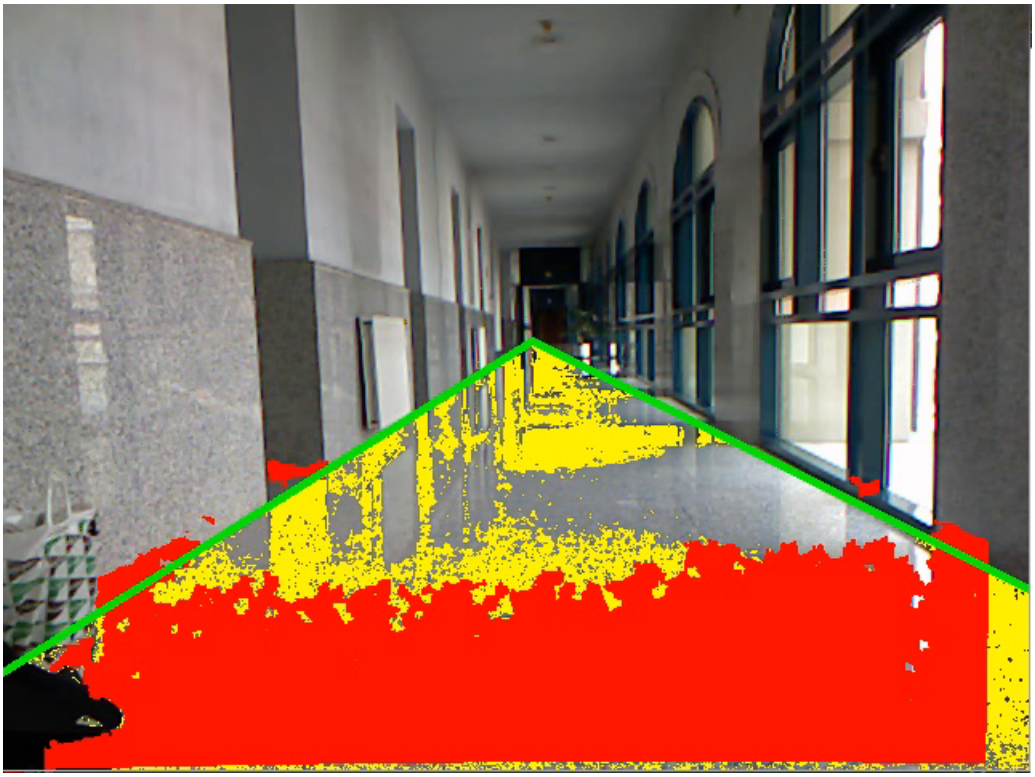

Figura 62: Pasillo con distancia de Mahalanobis 2

Se aprecia un resultado muy pobre, por lo que en este caso la distancia de Mahalanobis parte como algo menos restrictivo. Volviendo a la utilizada en estático se obtiene la figura 63:

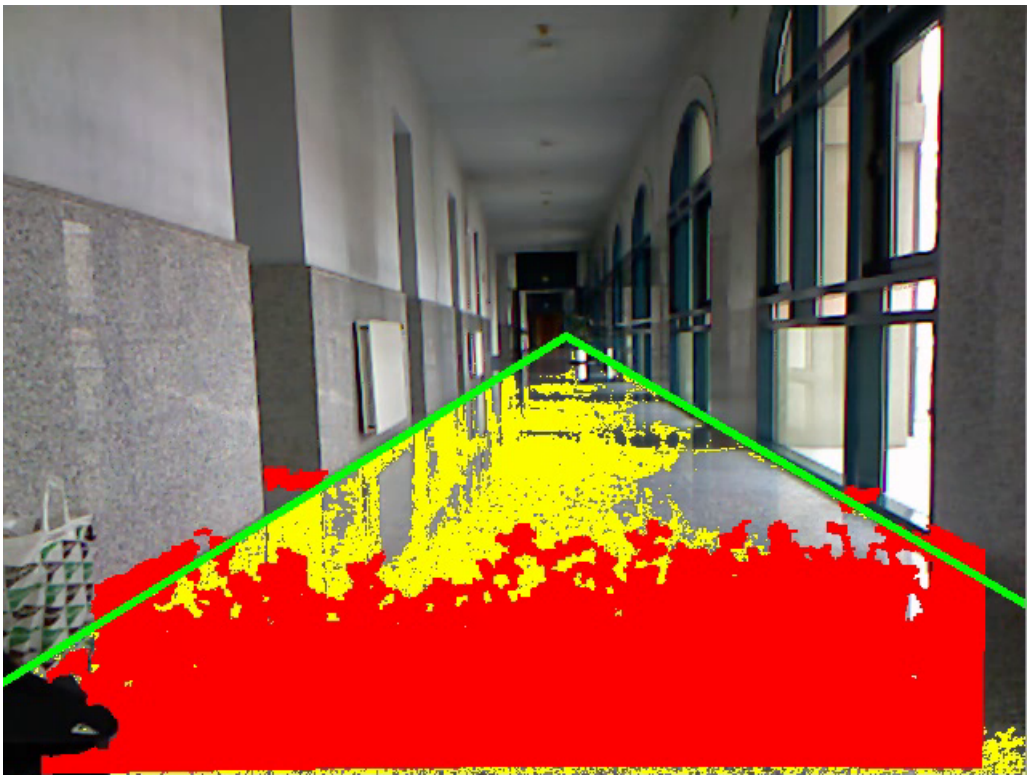

Figura 63: Pasillo con distancia de Mahalanobis 2.7

Un resultado mucho mejor, que además evita los objetos que aparecen en medio. Por tanto, la elección correcta de una distancia de Mahalanobis es imprescindible al cambiar de superficie, y se debe ajustar de un modo experimental.

7. Conclusiones y trabajo futuro

En esta última sección se van a analizar las conclusiones finales del proyecto, además de las limitaciones que posee y sus posibles ampliaciones o mejoras.

Aunque el algoritmo original demostró ser muy potente, su aplicación a entornos de interior genera numerosas limitaciones.

Por un lado, la cámara solo funciona en interiores. Debido a la tecnología basada en infrarrojos, su uso en exteriores es limitado o casi nulo debido a las interferencias de la luz solar. Además, en interiores con iluminación artificial surge otra limitación, y es la frecuencia a la que funciona la luz artificial. Al funcionar a 50Hz, pese a que es un caso que no suele ocurrir, se puede captar un frame en el que todo esté muy oscuro debido a que se ha capturado la parte nula de la onda. Mediante el uso de otros espacios de color como el HSV o LAB, este problema puede ser paliado en parte.

El rango de actuación de los sensores de Kinect es una limitación intrínseca que se ha aliviado para zonas concretas (como el suelo).

Por otro lado se encuentran limitaciones que si pueden ser eliminadas expandiendo u optimizando el código. El primer problema se encuentra en la lentitud del programa. Dicha lentitud viene dada por los numerosos cálculos que se realizan, pero puede ser mejorada con un uso más eficiente de los recursos y con una explotación mayor de las virtudes de la programación orientada a objetos, además de proponer una versión paralelizada de este algoritmo. Estas mejoras no se han implementado debido a que para lograrlo necesitan una curva de aprendizaje muy larga y un trabajo posterior muy largo, equivaliendo así a otro proyecto completo.

Otra limitación que se encuentra es su uso exclusivo en pasillos. Es un código que solo trabaja en terrenos ideales, necesitando superficies relativamente lisas, y sobretodo, la localización de pared derecha, izquierda y suelo para poder funcionar correctamente. En el algoritmo inicial las carreteras se distinguen muy bien del resto del terreno, mientras que en interiores es habitual encontrar suelos y paredes de colores similares.

Finalmente, algunos cálculos pueden ser agilizados con el uso de librerías de C++ dedicadas para tal fin, como la simplificación de tareas algebraicas utilizando las librerías *Eigen* para ello.

Este proyecto ha sido el fruto de varios meses de trabajo e investigación, abarcando en el proceso numerosos campos, tales como la programación orientada a objetos, la estadística y la geometría, aplicando además todas las facultades y conocimientos aprendidos a lo largo de la carrera.

A. Presupuesto

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

A.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
Planteamiento	Familiarización (h)	0	30
	Estudio del algoritmo (h)	10	20
	Diseño (h)	0	10
Desarrollo	Desarrollo del algoritmo (h)	5	300
	Investigación (h)	5	120
Pruebas	Pruebas en pasillo inicial (h)	10	80
	Pruebas en distintos pasillos (h)	1	5
Memoria	Redacción (h)	0	60
	Corrección (h)	15	15
Total	Total (h)	46	640
	Salario/hora	50	30
	Salario total (€)	2300	19200
	Total (€)		21500

Tabla 6: Costes desglosados de personal

A.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
Hardware	Ordenador	900 (€total)	100
	Kinect	100 (€total)	11.11
Software	Software	0	0
Oficina	Internet	30 (€/mes)	120
Total (€)			231.11

Tabla 7: Costes desglosados de material

A.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 8: Costes desglosados totales

Leganés, 11 de Febrero de 2013

El ingeniero

Referencias

[1] Dahlkamp, Hendrick, Kaehler, Adrian, Stavens, David, Thrun, Sebastian and Bradski, Gray. En *Self-supervised monocular road detection in desert terrain*. Disponible en: `http://robots.stanford.edu/papers/dahlkamp.adaptvision06.pdf` (25 de septiembre de 2012).

[2] Riyad A. El-laithy, Jidong Huang, Michael Yeh. En *Study on the use of Microsoft Kinect for Robotics Applications*. Disponible en: `http://ieeexplore.ieee.org.strauss.uc3m.es:8080/stamp/stamp.jsp?tp=&arnumber=6236985` (25 de septiembre de 2012).

[3] Xiaoyu Wu, Cheng Yang, Youwen Wang, Hui Li, Shengmiao Xu. En *An intelligent interactive system based on hand gesture recognition algorithm and Kinect*. Disponible en: `http://ieeexplore.ieee.org.strauss.uc3m.es:8080/stamp/stamp.jsp?tp=&arnumber=6405998` (25 de septiembre de 2012).

[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` (25 de septiembre de 2012).

[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` (25 de septiembre de 2012).

[6] Michael E. Farmer, Anil K. Jain. En *A wrapper-based approach to image segmentation and classification*. Disponible en: `http://ieeexplore.ieee.org.strauss.uc3m.es:8080/stamp/stamp.jsp?tp=&arnumber=1532306` (25 de septiembre de 2012).

[7] 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` (25 de septiembre de 2012).

[8] Mahalanobis, Prasanta Chandra (2006, 9 de marzo [últ. mod. 16 de agosto 2012]). En *Wikipedia, la enciclopedia libre*. Disponible en: `http://es.wikipedia.org/wiki/Prasanta_Chandra_Mahalanobis` (27 de septiembre de 2012).

[9] VGA (2004, 12 de agosto [últ. mod. 14 de junio 2012]). En *Wikipedia, la enciclopedia libre*. Disponible en: `http://es.wikipedia.org/wiki/Video_Graphics_Array` (27 de septiembre de 2012).

[10] Análisis Multivariante. En *Scribd*. Disponible en: `http://es.scribd.com/doc/52565764/Analisis-multivariado` (3 de octubre de 2012).

[11] API (2003, 16 de septiembre [últ. mod. 16 de septiembre 2012]). En *Wikipedia, la enciclopedia libre*. Disponible en: `http://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones` (3 de octubre de 2012).

[12] Stroustrup, Bjarne (2007, 15 de septiembre [últ. mod. 2 de octubre 2012]). En *Wikipedia, la enciclopedia libre*. Disponible en: `http://es.wikipedia.org/wiki/Bjarne_Stroustrup` (3 de octubre de 2012).

[13] Unix (2001, 19 de diciembre [últ. mod. 2 de octubre 2012]). En *Wikipedia, la enciclopedia libre*. Disponible en: `http://es.wikipedia.org/wiki/Unix` (3 de octubre de 2012).

[14] Stallman, Richard (2002, 1 de marzo [últ. mod. 5 de septiembre 2012]). En *Wikipedia, la enciclopedia libre*. Disponible en: `http://es.wikipedia.org/wiki/Richard_Stallman` (3 de octubre de 2012).

[15] Teknomo, Kardi. En *Ejemplo numérico de Agrupamiento K-medias*. Disponible en: `http://people.revoledu.com/` (10 de octubre de 2012).