TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

UNIVERSIDAD DE ALMERIA

ESCUELA SUPERIOR DE INGENIERÍA

"Aplicación de técnicas de tratamiento digital de imágenes en la reconstrucción 3D de viviendas a partir de planos "

Curso 2015/2016

Alumno/a:

Francisco Javier Molina Soler

Director/es:

María Mercedes Peralta López

José Antonio Torres Arriaza

AGRADECIMIENTOS

En primer lugar, quiero agradecer a mi padre y a mi madre por su apoyo incondicional que me han dado durante toda mi vida, tanto en el aspecto educativo, como en el aspecto personal, ellos me han enseñado todo lo posible para llegar hasta donde he llegado. Quiero darles las gracias por todo el esfuerzo que hacen día tras día, porque gracias a ello he conseguido luchar hasta conseguir la meta que deseaba.

Seguidamente, y no con menos mérito, me gustaría agradecer a mi novia, Paola por haber estado ahí como la que más, ayudándome en lo que necesitaba, dándome apoyo, aguantándome y sobretodo estando a mi lado.

A su vez quiero agradecer a mi familia y amigos por todo el apoyo que he recibido no solo en el ámbito educativo si no en el ámbito personal.

También me gustaría agradecer muy sinceramente a mis tutores, María Mercedes Peralta López y José Antonio Torres Arriaza. Su apoyo, dedicación, paciencia y consejos, además de lo que me han enseñado, merecen ser especialmente destacados. No han dudado nunca en compartir su tiempo conmigo para ayudarme.

Asimismo también quiero agradecer a la Universidad de Almería y a todos sus profesores por las enseñanzas aportadas.

Por último y no por eso menos importante quiero agradecer a mis compañeros de universidad con quienes en todos estos años hemos pasado mucho tiempo juntos, realizando prácticas, estudiando, y sobretodo pasándolo bien, porque aunque haya habido épocas de mucho agobio, siempre había alguien que conseguía sacar una sonrisa a todos y hacernos olvidar por un momento dicho agobio. Gracias por toda vuestra ayuda.

ÍNDICE

Página

ÍNDICE TABLAS

INDICE IMÁGENES

IMAGEN **Página**

1

MEMORIA DESCRIPTIVA DEL PROYECTO

1.1 Introducción

Existe muchas veces problemas del lado del usuario a la hora de interpretar un plano de una vivienda y hacerse la idea de cómo va a quedar en realidad ya que un plano en 2 Dimensiones (2d) no es muy representativo. Una técnica que puede acercar a la realidad es el paso de una vivienda a 3 Dimensiones (3d), ya que te permite ver con más realismo la distribución de la vivienda, los metros en una escala más pequeña, entre otras cosas. Existen varias herramientas para poder convertir un plano de una vivienda 2d a 3d, pero todas las herramientas existentes en la web o mercado solo te permiten el paso de forma manual, y a veces es un problema por su complejidad, sobre todo para los usuarios inexpertos. Por otra parte también puede ser un problema el excesivo tiempo que se puede llegar a invertir en autoformación para poder utilizar este tipo de herramientas o el paso del plano 2d a 3d en el caso de que la vivienda sea demasiado grande. Para facilitar y ayudar más a ese tipo de usuarios, proponemos un sistema que nos permite el paso automatizado de planos de viviendas 2d a 3d, mediante la técnica de tratamiento digital de imágenes. La idea del proyecto es crear un sistema inteligente que sea capaz de leer una imagen (un plano de una vivienda 2d) y el sistema sea capaz automáticamente de detectar paredes, puertas y ventanas y representarlas en 3d. El proyecto lo vamos a dividir en dos investigaciones diferentes:

La primera trata de poder interpretar el plano deseado por el usuario y realizar una extracción de información relevante para poder almacenarla en un fichero.

La segunda trata de interpretar la información extraída en la investigación anterior que está almacenarla en un fichero y poder representarla en 3d mediante la herramienta de google

Sketchup. Para ello crearemos un plugin[1] en ruby.

1.2 Motivación para el desarrollo

Desde que comencé en el mundo de la Informática me llamó la atención la variedad de campos en la que se puede llegar a dividir dicha profesión, pero realmente donde me di cuenta fue cuando comencé la carrera en Grado de Ingeniería Informática en la Universidad de Almería (UAL) ya que se puede llegar a dividir en tal variedad de campos y cada uno de ellos posee tal variedad de especializaciones, que resulta muy complicado conocer con gran profundidad cada uno de los mismos.

He de reconocer que a lo largo de mi carrera universitaria he llegado a aprender más de aquellas asignaturas que me resultaron duras, que me hicieron cuestionarme a mí mismo y sacar fuerzas desde donde no sabía que las tenía. Esas asignaturas supusieron un reto para mí. Por eso intenté buscar un proyecto en el que pudiera aprender aún más cosas y no solo mostrar los conocimientos que ya tenía.

Hubo un momento a lo largo de mi carrera que me tocó toparme con el mundo del diseño 3d, en el que tenía que convertir manualmente un plano de una vivienda 2d a 3d y pensé... ¿Por qué no hacerlo automáticamente? Y de ahí nació la idea de este proyecto.

1.3 Objetivos

El objetivo de este proyecto es diseñar una aplicación que sea capaz de detectar las paredes, puertas y ventanas de un plano de una vivienda en 2d y representarlo en 3d mediante la herramienta de google Sketchup. Vamos a diferenciar los objetivos en tres grupos principales:

- Objetivos formativos
- Objetivos generales
- Objetivos administrativos

1.3.1 Objetivos formativos

Desde el punto de vista formativo, me beneficiaré de la experiencia y enriqueceré mis conocimientos en el campo de la programación, tratamiento digital de imágenes y diseño 3d, con todo ello conseguiré:

- Conocer y utilizar las técnicas que se utilizan para el tratamiento digital de imágenes

[1] Aplicación que se relaciona con otra para aportarle una función nueva

- Conocer y utilizar la Application Programming Interface (API) de google Sketchup para el diseño 3d

- Conocer y utilizar el lenguaje de programación de Matlab para el tratamiento digital de imágenes

- Conocer y utilizar el lenguaje de programación Ruby para la representación en 3d

1.3.2 Objetivos generales

- Desarrollar algoritmos capaces de detectar objetos en imágenes, en este caso paredes, puertas y ventanas

- Desarrollar algoritmos capaces de representar en 3d unos parámetros leídos desde un fichero.

- Emplear el gran potencial de Matlab para el tratamiento digital de imágenes

1.3.3 Objetivos administrativos

Aprobar la asignatura "Trabajo Fin de Grado" para la consecución del título de Graduado en Ingeniería Informática. La finalidad de este proyecto es crear una base para futuras investigaciones en este campo, ya que la herramienta no detecta objetos o muebles como puede ser una cama o un armario, y para detectar una puerta debe estar en color rojo y una ventana en color azul.

1.4 Planificación

La planificación prevista para el proyecto, el cual duraría 267 días de trabajo aunque no todos los días se le va a dedicar 8 horas. El proyecto empezaría el jueves 17 de julio del 2014 acabando este el viernes 24 de julio 2015.
Para planificar el proyecto lo hemos divido en tres fases independientes, cada fase contiene sus propias tareas y subtareas estructuradas en el tiempo.
A continuación vemos el diagrama de gantt general del proyecto en las que aparecen las tres fases, más abajo se desglosa cada fase con su respectivo diagrama de gantt.

	Nombre de tarea	Duración
1	**Desarrollo del proyecto completo**	**267 días**
2	Creación plugin para representación 3d en google SketchUp	68 días
3	Creación de la aplicación para detectar paredes, puertas y ventanas	120 días
4	Memoria y presentación	68 días

Figura 1.1: Tabla de fases

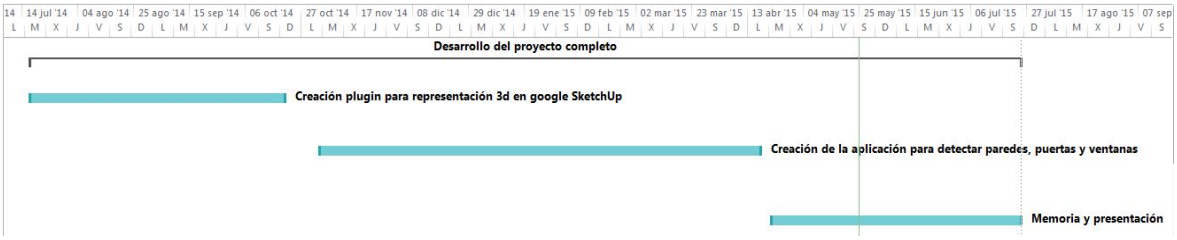


Figura 1.2: Diagrama de gantt de fases

1.4.1 Fase1-Creación del plugin para representación 3D

Esta fase es la encargada de crear el plugin para google SketchUp que representa la vivienda en 3D, que empezaría el jueves 17 de julio del 2014 y terminaría el lunes 20 de octubre del 2014. Para planificar esta fase del proyecto hemos creado dos tareas globales:

- Estudio e investigación previa sobre toda la tecnología a utilizar.

- Implementación y diseño del plugin de google SketchUp utilizado para la representación en 3D.

Estas dos tareas dentro de ellas se dividen en subtareas. A continuación vemos el diagrama de gantt[2] de esta fase:

[2]Es una útil herramienta gráfica cuyo objetivo es exponer el tiempo de dedicación previsto para diferentes tareas.

	Nombre de tarea	Duración
1	**Creación plugin para representación 3d en google sketchup**	**68 días**
2	**Estudio e investigación previa**	**21 días**
3	Introducción sobre Ruby	6 días
4	Introducción sobre google sketchup	5 días
5	Estudio e Investigación sobre la API de google sketchup	14 días
6	**Implementación y diseño**	**46 días**
7	Diseño del fichero de entrada	2 días
8	Diseño general del plugin	10 días
9	Construccion plugin	34 días
10	Compilación y pruebas	2 días

Figura 1.3: Fase1-Tabla de tareas

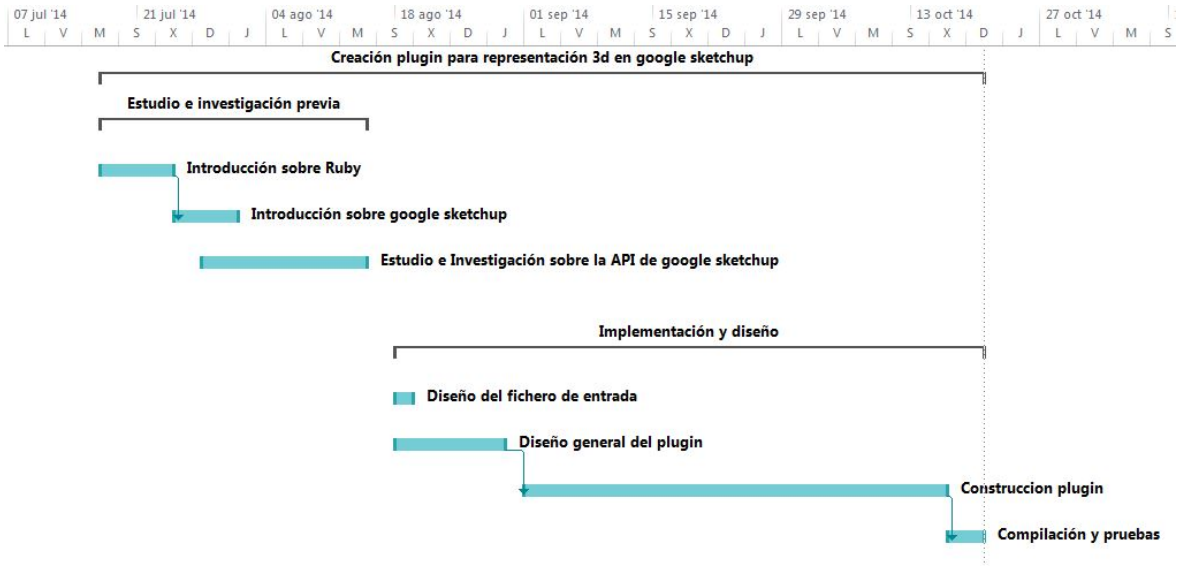


Figura 1.4: Fase1-Diagrama de gantt

1.4.2 Fase2-Creación de aplicación detección de paredes, puertas y ventanas

Esta fase es la encargada de crear una aplicación para poder detectar las paredes, puertas y ventanas del plano en 2d, que empezaría el lunes 3 de noviembre del 2014 y terminaría el viernes 17 de abril del 2015.
Para planificar esta fase del proyecto hemos creado tres tareas globales:

- Estudio e investigación previa sobre toda la tecnología a utilizar.

- Implementación y diseño de la aplicación que detecta paredes, puertas y ventanas.

Estas tres tareas dentro de ellas se dividen en subtareas. A continuación vemos el diagrama de gantt de esta fase:

	Nombre de tarea	Duración
1	**Creación de la aplicación para detectar paredes, puertas y ventanas**	**120 días**
2	**Estudio e investigación previa**	**39 días**
3	Introducción tratamiento digital de imágenes	5 días
4	Investigación y estudio sobre tecnicas de detección de bordes	15 días
5	Investigación y estudio sobre tecnicas de detección de colores	10 días
6	Introducción matlab	9 días
7	Investigación y estudio funciones relacionadas con Tratamiento digital de imágenes en matlab	13 días
8	**Implementación y diseño**	**73 días**
9	Diseño general de la aplicación	11 días
10	Construcción de la aplicación	55 días
11	Compilación y pruebas	7 días

Figura 1.5: Fase2-Tabla de tareas

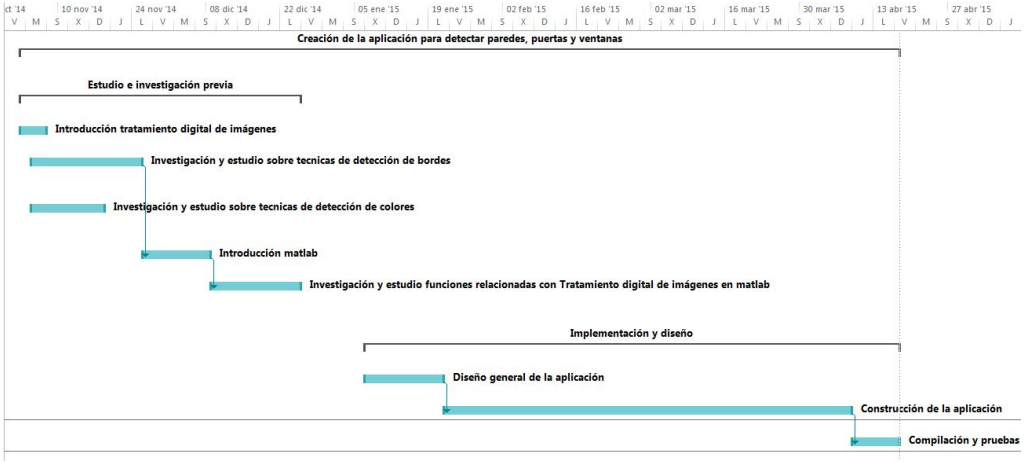


Figura 1.6: Fase2-Diagrama de gantt

1.4.3 Fase3-Memoria y presentación

En esta fase se realiza toda la documentación y la presentación del proyecto. Para realizar la documentación se va a utilizar LaTeX[3]. Esta fase empezaría el miércoles 22 de abril del 2015 y finalizaría el viernes 24 de julio de 2015.

	Nombre de tarea	Duración
1	**Memoria y presentación**	**68 días**
2	Formación y estudio sobre Latex	11 días
3	Revisión bibliografica	5 días
4	Realización documentación	38 días
5	Realización y preparación presentación	12 días

Figura 1.7: Fase3-Tabla de tareas

Figura 1.8: Fase3-Diagrama de gantt

1.5 Estructura de la memoria

La memoria está estructurada en los siguientes apartados:

- **Herramientas y lenguajes de programación utilizados:** En este apartado se describirá las herramientas que se han utilizado para realizar este proyecto al igual que los lenguajes de programación utilizados.
- **Introducción al tratamiento digital de imágenes:** En esta sección se explicarán conceptos básicos sobre la tecnología utilizada sobre el tratamiento digital de imágenes.
- **Introducción a la representación 3D:** En esta sección se explicarán los conceptos básicos sobre la representación 3D, así como la representación de forma gráfica, como la

[3]Es un sistema de composición de textos, orientado a la creación de documentos escritos que presenten una alta calidad tipográfica.

representación de forma automatizada utilizando la API de google sketchup (creación de plugins).

- **Diseño e Implementación:** Este apartado es el más importante de todos, ya que se describirán las técnicas, algoritmos y diseño utilizado para la creación del proyecto.
- **Simulaciones y resultados:** En este apartado se expondrán las simulaciones realizadas para evaluar el software y los resultados obtenidos.
- **Conclusiones y trabajos futuros:** En esta sección se presentarán las conclusiones a las que se ha llegado tras realizar el proyecto, así como posibles mejoras o cambios que se pudiesen realizar.
- **Bibliografía:** Contiene todas las referencias utilizadas para realizar del proyecto.

2

ESPECIFICACIONES GENERALES

2.1 Introducción al tratamiento digital de imágenes

[1] [6] [19]

El Procesamiento Digital de Imágenes (PDI) es el procesamiento de imágenes digitales por medio de una computadora digital. Las imágenes digitales incluyen a aquellas obtenidas del rango visible del espectro electromagnético, imágenes acústicas, electrónicas y sintéticas. Un paradigma del PDI suele clasificar los tipos de procesamientos computarizados en tres tipos: bajo, medio y alto.

Los antecedentes históricos del PDI se remontan a la impresión de periódicos en 1921, en donde la codificación y transmisión de datos se realizaba por cable submarino, razón por la cual no se le considera formalmente como procesamiento digital.

La historia del PDI está directamente relacionada con el desarrollo y evolución de las computadoras, dado que el PDI requiere un alto poder computacional para almacenar y procesar imágenes. Es a partir de las maquinas poderosas de los años 60´s, en conjunto con el Programa Espacial de los Estados Unidos, lo que dio origen a lo que conocemos hoy como PDI.

Los componentes fundamentales del PDI son: sensores, digitalizadores, hardware especial, computadora, software, almacenamiento, monitores, impresión y acceso a red.

En términos generales, los pasos fundamentales en el PDI son: adquisición de la imagen, realce de la imagen, restauración de la imagen, procesamiento de imagen en color, segmentación, representación y reconocimiento.

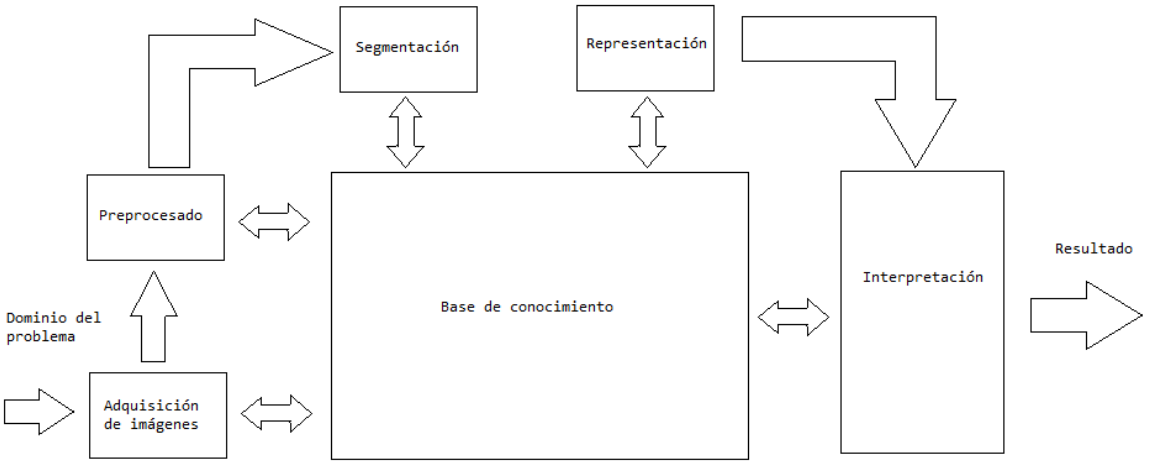


Figura 2.1: Pasos fundamentales PDI

[2] [3]

Consideramos una función bidimensional para definir a una imagen digital, el par de coordenadas (x, y) definen un único punto en la matriz y el valor de la función en ese punto $f(x, y)$ da el valor de la intensidad lumínica en ese punto, donde:

$$0 \leq f(x, y) \leq \infty$$

Figura 2.2: Convenio de ejes utilizado para la representación de imágenes

Esta es la definición de imagen monocromática, y el valor de $f(x, y)$ recibe el nombre de nivel de gris dentro de una escala: escala de grises, donde:

$f(x, y) = 0 \ldots$ Ausencia de luz

$f(x, y) = L \ldots$ Máxima luminosidad (Nunca puede ser infinito)

$$\text{negro} \leq f(x, y) \leq \text{blanco}$$

Entonces para realizar la representación de una imagen usamos matrices en memoria donde tenemos:
m . . . filas
n . . . columnas
p . . . nº de bits para la cuantificación de una muestra

Cada punto de la matriz es un píxel, entonces el número de píxeles es MxN, esto define la resolución, y cada píxel está cuantificado en un número determinado de bits definiendo desde a 0 a 2^p posibles valores lo que define la calidad de la imagen.
Entonces la matriz queda de un tamaño dado por: MxNxP. . . bits.

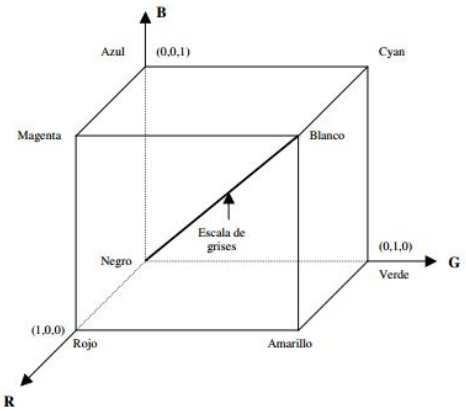


Figura 2.3: Cubo de valores RGB

2.1.1 Pre-Procesamiento de imágenes

[1] [4]
El primer paso tras la adquisición de la imagen es pre-procesarla. La etapa del pre-procesado es la aplicación de técnicas y herramientas para realizar una transformación de la imagen con el fin de mejorarla para los siguientes procesos. Estos procesos abarcan desde el recorte de la imagen de las zonas que nos interesan, como técnicas de suavizado para reducir ruido.
El primer paso en el pre-procesamiento es asegurarnos de que la imagen contiene las regiones de interés de la mejor manera posible. Para ello podemos aplicar técnicas de recorte, giro o cualquier otra técnica que nos permita delimitar físicamente la zona a estudiar.
La aplicación de filtros de eliminación de ruido también podría describirse como técnica de pre-procesado. Existen numerosos filtros que aplican transformaciones píxel a píxel, que no solo dependen de la tonalidad del píxel a tratar, si no también de la de sus vecinos. La aplicación de estos filtros ocurre mediante máscaras, matrices cuadradas con un número de filas impar. Algunos de los filtros más utilizados son:

- Filtros Gaussianos
- Filtros de mediana
- Filtros de suavizados
- Filtros SUSAN.

También procesos que modifican el contraste pueden aplicarse en el pre-procesamiento. La aplicación de contraste es una técnica que modifica la función de transferencia de tal manera que su pendiente sea distinta a uno.
La binarización de la imagen también es una técnica de ampliación de contraste, esta técnica es la que vamos a utilizar para la realización del proyecto. Es un caso extremo, en que la función de transferencia para los tonos oscuros y claros es cero y para las tonalidades intermedias es $\pi/2$. La imagen resultado se compone únicamente de tonalidades blancas y negras, con lo cual se puede simplificar enormemente tareas de segmentación o detección de bordes.

2.1.1.1 Binarización

Las imágenes binarias siempre se obtienen a partir de imágenes de niveles de gris. En la actualidad no existen cámaras comerciales que proporcionen imágenes binarias. El proceso de conversión de una imagen de nivel de gris a una imagen formada solo por dos valores o etiquetas (0 para el negro y 1 para el blanco) se conoce como binarización.

Figura 2.4: Imagen origintal

Figura 2.5: Imagen binarizada

La binarización tiene una gran utilidad en procesamiento automático de imagen pues reduce enormemente la cantidad de datos de la imagen de una forma muy sencilla. Si se parte de imágenes bien contrastadas, la binarización permite con muy poco procesamiento un análisis fiable de la imagen. Para la obtención de imágenes de gran contraste se suele recurrir a la utilización de técnicas de retroiluminación (contraluz). Estas imágenes obtenidas a contraluz pueden transformarse sin pérdida significativa de información a binarias.
La forma más inmediata de representar una imagen de niveles de gris mediante una imagen binaria es considerar únicamente el bit más significativo del nivel de gris de cada píxel. O lo

que es lo mismo, fijar un umbral en la mitad de la escala de gris que servirá de referencia para asignar en la imagen binaria un 0 si el nivel de gris del píxel es inferior o un 1 si supera este umbral.

$$f(x,y) = \begin{cases} 1 & \text{si } f(x,y) > T \\ 0 & \text{si } f(x,y) \leq T \end{cases}$$

T es el umbral previamente calculado.

2.1.1.2 Filtrado

[1] [19]

Un filtro puede verse como un mecanismo de cambio o transformación de una señal de entrada a la que se le aplica una función, conocida como función de transferencia, para obtener una señal de salida. Todas estas señales y funciones pueden ser discretas o continuas, y aunque en el tratamiento de imágenes se usan señales y funciones discretas.

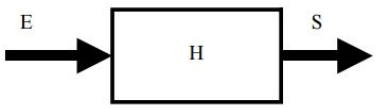


Figura 2.6: Filtro discreto con entrada E, salida S y función de transferencia H

Filtrado espacial

El empleo de máscaras espaciales para el procesamiento de las imágenes se denomina filtrado espacial, y las propias máscaras se llaman espaciales. Existen tres tipos de filtros espaciales:

- Filtros de paso bajo
- Filtros de paso alto
- Filtros de paso banda

Los filtros denominados de **paso bajo** atenúan o eliminan las componentes de alta frecuencia en el dominio de Fourier a la vez que dejan inalteradsa las bajas frecuencias, es decir deja pasar las frecuencias bajas. Las componentes de alta frecuencia caracterizan los bordes y los restantes detalles muy marcados de la imagen, y por ello el efecto de introducir un filtrado de paso bajo es el de hacer más borrosa la imagen. Los filtros denominados de **paso alto** atenúan o eliminan las componentes de baja frecuencia. Como estos componentes son responsables de las características lentamente variables de la imagen, como el contraste global y la intensidad media, el resultado

neto de un filtrado de paso alto es la reducción de estas características y, en correspondencia, una aparente agudización de los bordes y de los restantes detalles finos. Los filtros denominados de **paso banda** eliminan unas regiones elegidas de frecuencias intermedias. Estos filtros se emplean para la restauración de imágenes y son de escaso interés para la mejora de imágenes.

Filtros suavizantes

Los filtros suavizantes se emplean para hacer que la imagen aparezca algo borrosa y también para reducir el ruido. Es útil que la imagen aparezca algo borrosa en algunas etapas de preprocesado, como la eliminación de los pequeños detalles de una imagen antes de la extracción de un objeto, y el relleno de pequeños espacios entre líneas o curvas. La reducción del ruido puede realizarse mediante el aumento de la borrosidad con un filtro lineal o también con un filtro no lineal.

Filtrado espacial de paso bajo

La forma de repuesta de un impulso necesaria para implementar un filtro espacial de paso bajo (suavizante) indica que le filtro ha de tener todos sus coeficientes positivos. Para un filtro de 3x3, la construcción más simple consistiría en una máscara en la que todos los coeficientes fuesen iguales a 1.

Filtrado por la mediana

Una de las principales dificultades del método de suavizados expuesto en la sección anterior es que difuminan los bordes y otros detalles de realce. Cuando el objetivo es más la reducción del ruido que el difuminado, el empleo de filtros de mediana representan una posibilidad alternativa. En este caso, el nivel de gris de cada píxel se reemplaza por la mediana de los niveles de gris. en un entorno de este píxel, en lugar de por la media. Este método es particularmente efectivo cuando el patrón de ruido consiste en componentes fuertes y de forma picuda, y la característica que se desea preservar es la agudeza de los bordes. Como se indicó anteriormente, los filtros de mediana son no lineales. La m de un conjunto de valores es tal que la mitad de los valores del conjunto quedan por debajo de m y la otra mitad por encima. Con el fin de realizar el filtrado por la mediana en el entorno de un píxel, primero se deben extraer los valores del píxel y de su entorno, determinar la mediana y asignar este valor al píxel.

Filtros realzantes

El objetivo principal del realce es el de destacar los detalles finos de una imagen o intensificar detalles que han sido difuminados, bien sea por error o bien por efecto natural del método de adquisición de la imagen. Las utilidades del realce de las imágenes son variadas e incluyen aplicaciones que van desde la impresión electrónica y las imágenes médicas hasta las inspecciones

industriales e incluso la detección autónoma de objetivos en las armas inteligenes.

Filtrado espacial de paso alto

El perfil de la respuesta a un impulso necesaria para implementar un filtro espacial de paso alto (realce) indica que el filtro debe tener coeficientes positivos cerca de su centro y coeficientes negativos en la periferia. Para una máscara 3x3 esta condición se cumple escogiendo un valor positivo en el centro y tomando coeficientes negativos en el resto.

Mejora en el dominio de la frecuencia

Se trata de calcular la transformada de Fourier de la imagen a intensificar, multiplicar el resultado por la función de transferencia de un filtro y, finalmente, tomar la transformada de Fourier inversa para llegar a una imagen mejorada. Las ideas de pérdida de nitidez por reducción del contenido de altas frecuencias o de mejor definición incrementando la magnitud de las componentes de alta frecuencia en relación con las de baja frecuencia proceden de conceptos directamente relacionados con la transformada de Fourier. De hecho, la idea general del filtrado lineal es bastante más atractiva e intuitiva en el dominio de la frecuencia.

Filtrado de paso bajo

Como se indicó anteriormente, los bordes y otras transiciones bruscas en los niveles de gris de una imagen contribuyen significativamente al contenido en altas frecuencias de su transformada de Fourier. Por lo tanto el difuminado se consigue, en el dominio de la frecuencia, a base de atenuar un rango específico de componentes de alta frecuencia en la transformada de una imagen dada.

$$G(u,v) = H(u,v)F(u,v) \tag{2.1}$$

donde $F(u,v)$ es la transformada de Fourier de la imagen que hay que suavizar. El principal problema radica en seleccionar una función de transferencia del filtro, $H(u,v)$, que dé $G(u,v)$ atenuando las componentes de alta frecuencia de $F(u,v)$. La transformada de Fourier inversa dará entonces la imagen $g(x,y)$ con el suavizado deseado.

Filtro ideal

Un filtro de paso bajo bidimensional idea es aquel cuya función de transferencia verifica la relación:

$$H(u,y) = \begin{cases} 1 & \text{si } D(u,v) \leq D_0 \\ 0 & \text{si } D(u,v) > D_0 \end{cases} \tag{2.2}$$

donde D_0 es una cantidad especificada no negativa, y $D(u,v)$ es la distancia desde el punto (u,v) al origen de coordenadas del plano de frecuencias; es decir:

$$D(u,v) = (u^2 + v^2)^{\frac{1}{2}} \tag{2.3}$$

La Figura 2.7 muestra un dibujo en perspectiva tridimensional de $H(u,v)$ como función de u y de v El nombre de filtro ideal indica que todas las frecuencias dentro de un círculo de radio D_0 pasan sin atenuación, mientras que todas las frecuencias fuera de este círculo quedan atenuadas completamente.

Para este tipo de filtro, es suficiente especificar su sección transversal en términos de una función de la distancia al origen según una dirección radial, como se muestra en la Figura 2.8. Entonces la función transferencia completa puede generarse haciendo girar esta sección transversal 360° alrededor del origen.

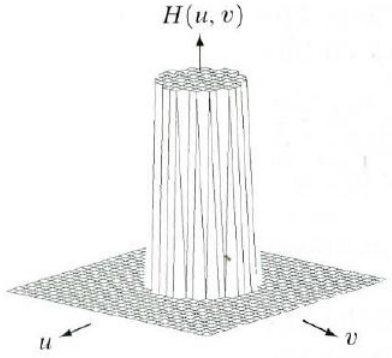


Figura 2.7: Perspectiva de la función de transferencia de un filtro paso bajo ideal

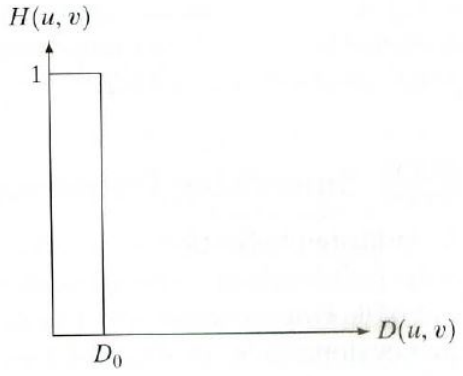


Figura 2.8: Sección transversal de este filtro

Filtro de Butterworth

La función de transferencia espacial de filtro de Butterworth de paso bajo de orden n, y con emplazamiento de frecuencia de corte a una distancia D_0 del origen, está definido por la relación

$$H(u,v) = \frac{1}{1 + [\frac{D(u,v)}{D_0}]^2 n} \tag{2.4}$$

donde $D(u,v)$ está dado por la ecuación (3.8). La Figura 2.9 muestra un dibujo en perspectiva y una sección transversal del filtro de Butterworth de paso bajo. Al contrario de lo que sucedía con el filtro ideal de paso bajo, la función de transferencia del filtro de Butterworth de paso bajo carece de una discontinuidad brusca que establezca un corte claro entre las frecuencias transmitidas y las filtradas. Para los filtros cuya función de transferencia cambie sin brusquedad, es habitual definir la frecuencia de corte a partir del lugar de los puntos donde la función $H(u,v)$ corresponde a una determinada fracción de su valor máximo. En el caso de la ecuación (3.9),

$H(u,v) = 0.5$ cuando $D(u,v) = D_0$. Otro valor comúnmente empleado es $1/\sqrt{2}$ del valor máximo de $H(u,v)$. Para la ecuación (3.9), la simple modificación siguiente da el valor deseado cuando $D(u,v) = D_0$:

$$H(u,v) = \frac{1}{1+[\sqrt{2}-1][\frac{D(u,v)}{D_0}]^2 n} = \frac{1}{1+0.414\frac{D(u,v)}{D_0}]^2 n} \tag{2.5}$$

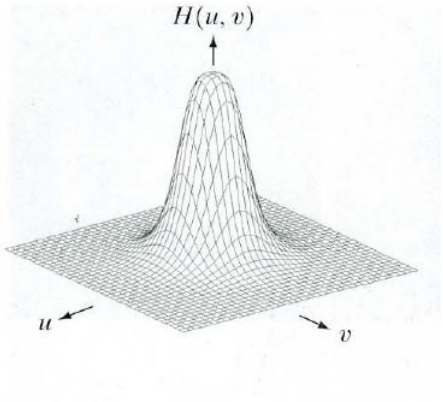


Figura 2.9: Filtro de Butterworth de paso bajo

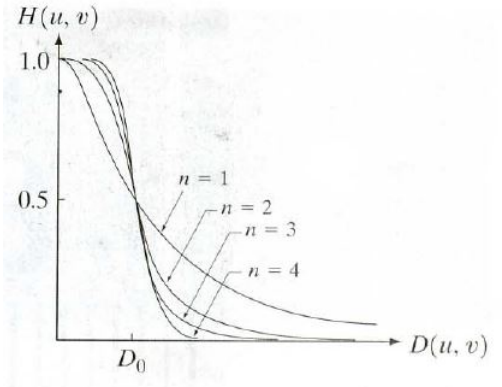


Figura 2.10: Sección transversal radial

2.1.2 Segmentación de imágenes

[1] [12]

La segmentación subdivide una imagen en sus partes constituyentes u objetos. El nivel al que se lleva a cabo esta subdivisión depende del problema a resolver. Esto es, la segmentación deberá detenerse cuando los objetos de interés de una aplicación hayan sido aislados. Los algoritmos de segmentación de imágenes monocromáticas generalmente se basan en una de las dos propiedades básicas de los valores del nivel de gris:

- Discontinuidad: Consiste en dividir una imagen basándose en los cambios bruscos de nivel de gris. Las principales áreas de interés de esta categoría son la detección de puntos aislados y la detección de líneas y bordes de una imagen.

- Similaridad: Consiste en dividir una imagen basándose en la umbralización, crecimiento de región, y división y fusión de regiones.

El concepto de segmentación de una imagen basado en la discontinuidad o similaridad de los valores del nivel de gris de sus píxeles es aplicable tanto a las imágenes estáticas como a las dinámicas. En el último caso, sin embargo, el movimiento puede utilizarse a menudo como un potente indicador para mejorar el rendimiento de los algoritmos de segmentación.

2.1.2.1 Detección de discontinuidades

Vamos a ver técnicas para la detección de los tres tipos básicos de discontinuidades de una imagen digital: puntos, líneas y bordes. En la práctica, la forma más común de ver las discontinuidades es pasar una máscara a través de la imagen. Para una máscara 3x3 como la que se muestra a continuación:

$$\begin{pmatrix} z_1 & z_2 & z_3 \\ z_4 & z_5 & z_6 \\ z_7 & z_8 & z_9 \end{pmatrix}$$

este procedimiento implica calcular la suma de los productos de los coeficientes por los niveles de gris contenidos en la región encerrada por la máscara. Esto es, la respuesta de la máscara en un punto cualquiera de la imagen es:

$$R = w_1 z_1 + w_2 z_2 + ... + w_9 z_9 = \sum_{i=1}^{9} w_i z_i \tag{2.6}$$

donde z_i es el nivel de gris asociado con el coeficiente de la máscara w_i. Como habitualmente, la respuesta de la máscara está definida con respecto a la posición de su centro. Cuando la máscara está centrada en un píxel de límite, la respuesta se calcula utilizando el entorno parcial apropiado.

Detección de puntos

La detección de puntos aislados de una imagen es directa. Utilizando la máscara que se muestra en la matriz 2.7, se sabe que se ha detectado un punto en la posición en la que está centrada la máscara si

$$|R| > T$$

donde T es un umbral no negativo, y R está dado por la ecuación 2.6. Básicamente, todo lo que hace esta formulación es medir las diferencias ponderadas entre el punto central y sus vecinos. Partiendo de la idea de que el nivel de gris de un punto aislado será bastante diferente del de sus vecinos.

La máscara de la matriz 2.7 es la misma que la utilizada para el filtrado de alta frecuencia espacial. Sin embargo, aquí se hace hincapié estrictamente en la detección de puntos. Esto es, solamente son de interés las diferencias lo suficientemente grandes para que se consideren puntos aislados de una imagen.

$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} \tag{2.7}$$

Detección de líneas

El siguiente nivel de complejidad implica la detección de líneas en una imagen. Considérense las máscaras que se muestra en la figura 2.8. Si la primera máscara se trasladara por toda una imagen, podría responder con más fuerza a líneas orientadas horizontalmente. Con un fondo constante, la respuesta máxima resultará cuando la línea pase por la fila central de la máscara. Esto se comprueba fácilmente esbozando una simple matriz de 1 con una línea de nivel de gris diferente recorriendo horizontalmente la matriz. Un experimento similar podría revelar que la segunda máscara de la Figura 2.8 responde mejor a líneas en la dirección de -45^o. Estas direcciones pueden también establecerse al observar que la dirección preferida de cada máscara está ponderada con un coeficiente mayor que las otras direcciones posibles.

$$\begin{pmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{pmatrix} \quad \begin{pmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{pmatrix} \quad \begin{pmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{pmatrix} \quad \begin{pmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{pmatrix} \tag{2.8}$$

Sean $R_1, R_2, R_3 y R_4$ las respuestas de las máscaras de la Figura 2.8, de izquierda a derecha, donde R están dados por la ecuación 2.6. Supóngase que todas las máscaras pasan sobre una imagen. Si en un cierto punto de la imagen $|R_i| > |R_j|$, para todo $j \neq i$, este punto será el que tenga la mayor probabilidad de estar asociado con una línea en la dirección de la máscara i.

Detección de bordes

Aunque la detección de punto y línea son evidentemente elementos de cualquier presentación de la segmentación, la detección de bordes es con mucho el método más común para detectar discontinuidades significativas en el nivel de gris. La razón es que los puntos aislados y las líneas delgadas no son de frecuente aparición en la mayor parte de las aplicaciones prácticas.

Formulación Básica

Un borde es la frontera entre dos regiones con propiedades de nivel de gris relativamente distintas. Básicamente la idea que subyace en la mayor parte de las técnicas de detección de bordes es el cálculo de un operador local de derivación. La imagen de la izquierda de la Figura 2.11 muestra una imagen de una banda clara sobre un fondo oscuro, el perfil del nivel de gris a lo largo de una línea de exploración horizontal de la imagen, y la segunda derivada del perfil. La derivada segunda es positiva en la parte de la transición asociada con el lado oscuro del borde, negativa en la parte de la transición asociada con el lado claro y cero en las zonas de nivel de gris constante.

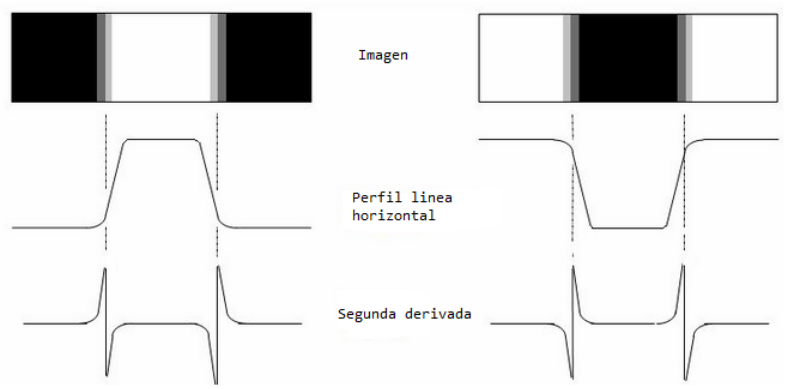


Figura 2.11: Detección de bordes por operadores de derivación

Por tanto, el módulo de la **derivada primera** se puede utilizar para detectar la presencia de un borde en una imagen, y el signo de la **derivada segunda** puede utilizar para determinar si un píxel borde está situado en el lado oscuro o claro del mismo. Se observa que la derivada segunda tiene un paso por cero en el punto medio de una transición de nivel de gris.

Operadores gradiente

El gradiente de una imagen $f(x,y)$ en la posición (x,y) es el vector.

$$\nabla f = \begin{pmatrix} G_x \\ G_y \end{pmatrix} = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix} \tag{2.9}$$

Se sabe del análisis vectorial que el gradiente de un vector indica la dirección de la máxima variación de f en (x,y). Una importante cantidad en la detección de bordes es el módulo de este vector, al que generalmente se hace referencia, para simplificar, como **gradiente**, con la notación ∇f, donde:

$$\nabla f = mag(\nabla f) = [G_x^2 + G_y^2]^{\frac{1}{2}} \tag{2.10}$$

Esta cantidad es igual a la máxima variación de $f(x,y)$ por unidad de distancia en la dirección de ∇f. Es práctica común aproximar el gradiente por sus valores absolutos:

$$\nabla f \approx |G_x| + |G_y| \tag{2.11}$$

La **dirección** del vector gradiente es también una cantidad importante. Sea $\propto (x,y)$ la representación del ángulo de dirección del vector ∇f en (x,y). Entones, del análisis vectorial:

$$\propto (x,y) = tan^{-}1\left(\frac{G_y}{G_x}\right) \tag{2.12}$$

donde el ángulo se mide con respecto al eje x.

De las ecuaciones 2.9 y 2.10 se deduce que el cálculo del gradiente de una imagen se basa en la

obtención de las derivadas parciales $\partial f/\partial x$ y $\partial f/\partial y$ en cada posición de píxel.

Laplaciano

El laplaciano de una función bidimensional $f(x,y)$ es una derivada de segundo orden definido por:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{2.13}$$

Como en el caso del gradiente, la ecuación 2.14 puede implementarse de forma digital de varias maneras. Para una región 3x3, la forma que se encuentra más frecuentemente en la práctica es:

$$\nabla^2 f = 4z_5 - (z_2 + z_4 + z_6 + z_8) \tag{2.14}$$

Como el laplaciano responde a las transiciones de intensidad, rara vez se utiliza en la práctica para la detección de bordes por varias razones:

- Como es una derivada de segundo orden normalmente es inaceptablemente sensible al ruido.
- Produce bordes dobles.
- Incapaz de detectar direcciones de borde.

2.1.2.2 Enlazado de bordes y detección de límites

Los algoritmos de detección de bordes normalmente se continúan por procedimientos de enlazado y detección de límites diseñados para reunir píxeles del borde en límites que tengan algún sentido, en el caso de nuestro proyecto algunas técnicas se utilizan para la extracción de características.

Procesamiento local

Uno de los procedimientos más simples para enlazar puntos del borde consiste en analizar las características de los píxeles de una pequeña vecindad respecto a cada uno de los puntos (x,y) de una imagen que ha sufrido una detección de borde. Todos los puntos que son similares se enlazan, formando un límite de píxeles que comparten algunas propiedades comunes.

Las dos propiedades utilizadas en este tipo de análisis para establecer la similaridad de los píxeles del borde son:

- La intensidad de la respuesta del operador gradiente utilizado para producir el píxel del borde.

- La dirección del gradiente.

Transformada de Hough

[1] [13]

La transformada de Hough considera las relaciones globales entre píxeles de borde permitiendo encontrar ciertos patrones en la imagen como líneas y círculos.

Es técnica propuesta por Paul Hough (1962) para encontrar la ecuación de una línea que pase por un conjunto de n puntos en el plano xy, expresada de la forma,

(2.15) $$y = ax + b$$

Considerando un punto de partida (x_i, y_i), podemos pensar que existen infinitas líneas que pasan por ese punto y que cumplan con la ecuación 2.15, para valores variables de a y b. Sin embargo, escribiendo la ecuación 2.1.2.2 y considerando el plano ab[1] se obtiene la ecuación de una única línea para un par determinado (x_i, y_i).

(2.16) $$y = -ax + b$$

Además, un segundo punto (x_j, x_j) también tiene una línea en el espacio parámetro asociado con él, y esta línea corta a la línea asociada con (x_i, y_i) en (a', b'), donde a' es la pendiente y b' la ordenada en el origen de la línea que contiene a (x_i, y_i) y (x_j, x_j)en el plano (xy)

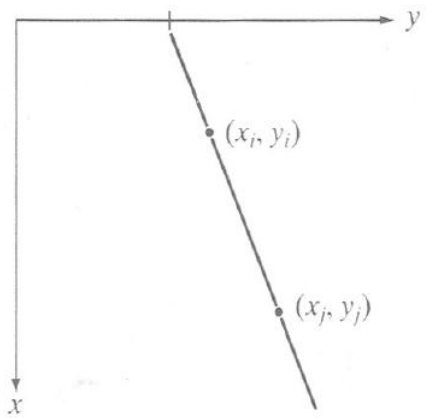


Figura 2.12: Plano xy

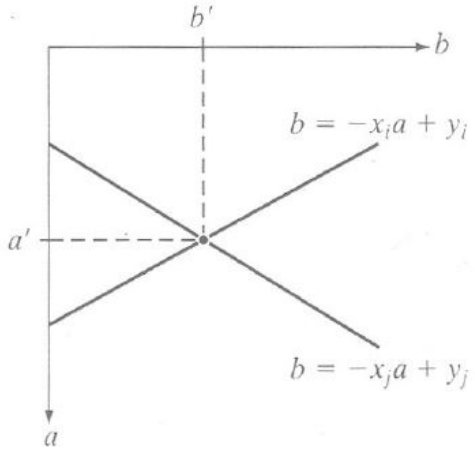


Figura 2.13: Espacio parámetro

El atractivo de la transformada de Hough proviene de subdividir el espacio de parámetros en celdas acumuladoras, como se puede ver en la Figura 2.14, donde (a_min, a_max) y (b_min, b_max) son los rangos esperados para la pendiente y la ordenada al origen. La celda de coordenadas (i, j)

[1]Espacio parámetro

con un valor de acumulador $A(i,j)$ corresponde al cuadrado asociado con las coordenadas (a_i, b_j) del espacio de parámetros. Inicialmente se ponen todos los acumuladores a cero. Entonces para cada punto (x_k, y_k) de la imagen, permitimos que el parámetro a pueda tomar cualquier valor de entre los a_i permitidos y calculamos b usando la ecuación 2.1.2.2. Los valores resultantes para el parámetro b se redondean hasta los b_j permitidos. Si para un valor a_p resultó un valor b_q se tiene que

$$A(p,q) = A(p,q) + 1$$

Al final, un valor de M en el acumulador $A(i,j)$ significa que M puntos del plano xy caen sobre la recta $y = a_i x + b_j$. La precisión en la colinealidad de estos puntos depende del número de celdas del espacio de parámetros.

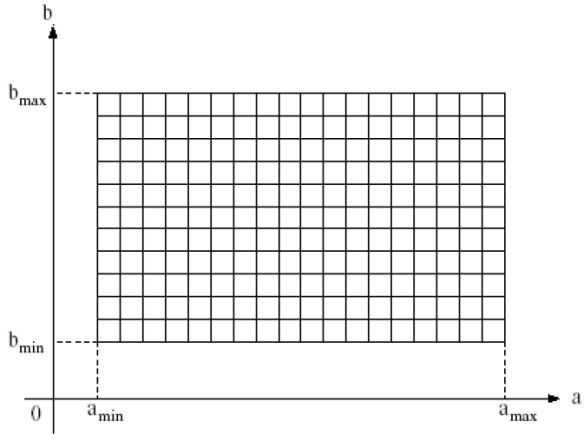


Figura 2.14: Celdas acumuladoras en el espacio de parámetros.

Un problema que surge al emplear la ecuación de la recta $y = ax + b$ para representar una línea es que tanto la pendiente como la ordenada al origen pueden llegar a valer infinito, según la línea se hace vertical. Una forma de solventar este problema consiste en utilizar la representación normal de la recta

$$x cos\theta + y sen\theta = p$$

En la Figura 2.15 se puede ver el significado de los nuevos parámetros (p, θ). El uso de esta representación para construir la tabla de acumuladores es similar al método explicado para las rectas en la forma pendiente y ordenada al origen. A cada punto del plano xy corresponde ahora una sinusoide en el plano $p\theta$ en lugar de una recta.

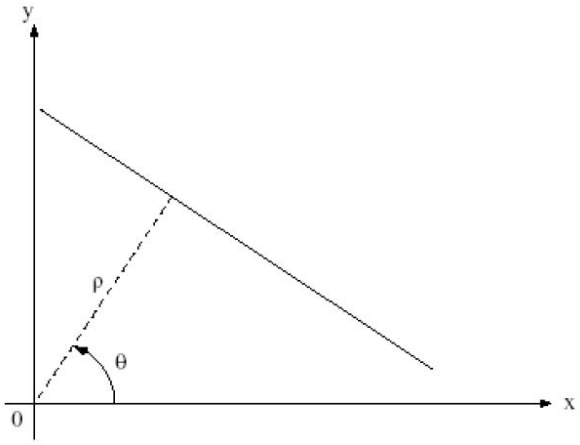


Figura 2.15: Representación normal de una recta

Al igual que antes, M puntos colineales a la recta $xcos\theta_j + ysen\theta_j = p_i$ darán lugar a M sinusoides que se cortan en el punto (p_i, θ_j) en el espacio de parámetros. Incrementando θ y calculando p, obtendremos M entradas en el acumulador $A(i, j)$ correspondiente al par (p_i, θ_j).

En la Figura 2.17 podemos ver la tabla de acumuladores del espacio de parámetros en este caso. El rango para el ángulo θ es $\pm 90^o$, medido con respecto al eje de abscisas. Se permiten valores negativos de p para rectas por detrás del origen de coordenadas del plano xy. Por ejemplo, una recta horizontal corresponde a un valor de $\theta = 0^o$ y un valor de p igual a la ordenada al origen, mientras que una recta vertical corresponde a un valor de $\theta = 90^o$ y un valor de p igual a la abscisa en el origen.

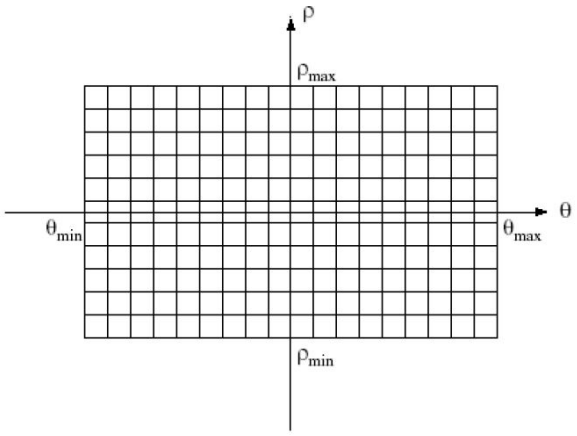


Figura 2.16: Celdas acumuladoras en el espacio $p\theta$.

2.2 Introducción a la representación 3D

2.2.1 Concepto de representación 3D

También llamado modelado 3D es el proceso de desarrollar una representación matemática de cualquier objeto tridimensional (ya sea inanimado o vivo) a través de un software especializado. Al producto se le llama modelo 3D. Se puede visualizar como una imagen bidimensional mediante un proceso llamado renderizado 3D o utilizar en una simulación por computadora de fenómenos físicos. El modelo también se puede crear físicamente usando dispositivos de impresión 3D.
Los modelos 3D representan un objeto 3D usando una colección de puntos en el espacio dentro de un espacio tridimensional, conectados por varias entidades geométricas tales como triángulos, líneas, superficies curvas, etc. Casi todos los modelos 3D pueden ser divididos en dos categorías.

- **Sólidos**: Estos modelos definen el volumen del objeto que representan (como una roca). Estos son más realistas, pero más difíciles de construir. Los modelos sólidos son mayormente usados para simulaciones no visuales, tales como médicas y de ingeniería.
- **Carcasa/contorno**: Estos modelos representan la superficie, por ejemplo el contorno del objeto, no su volumen. Es más fácil trabajar con ellos que con modelos sólidos. Casi todos los modelos visuales usados en juegos y películas son modelos protectores.

2.2.1.1 Conceptos matemáticos sobre la representación en 3D

Para poder expresarnos y comunicarnos de un modo correcto en el ámbito 3D, necesitamos tener claros y dominar los fundamentos geométricos, es decir, asimilar conceptos como "sistema de coordenadas", "vector", "normal", etc.
Un **sistema de coordenadas** es un conjunto de valores que permiten definir unívocamente la posición de cualquier punto de un espacio geométrico respecto de un punto denominado origen.

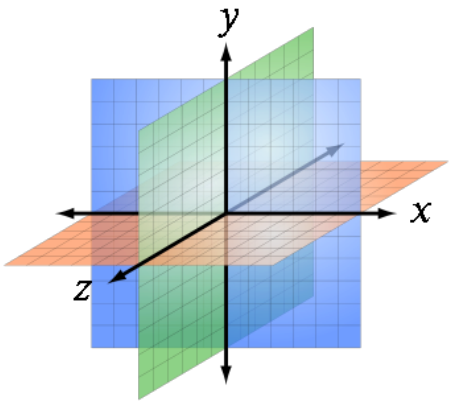


Figura 2.17: Sistema de coordenadas.

Segmento es aquella parte de una línea recta que queda entre dos puntos señalados sobre ella.

Un **vector** es todo segmento de recta dirigido en el espacio. Cada vector posee unas características que son:

- **Origen**: Punto exacto sobre el que actúa el vector.
- **Módulo**: Longitud del vector.
- **Dirección**: Orientación en el espacio de la recta que lo contiene.
- **Sentido**: Se indica mediante una punta de flecha situada en el extremo del vector.

Se considera un **polígono** a cualquier forma plana y cerrada (con su primer y último vértice perfectamente coincidentes). Un polígono también puede ser una figura 2D, una forma cerrada cuyos primer y último vértice coinciden.

2.2.2 Representación en 3D con Google Sketchup utilizando su API

[5]
La API de SketchUp Ruby es una forma de que los programadores de Ruby puedan ampliar las capacidades de SketchUp para satisfacer sus necesidades. Mediante la creación de un script y colocándolo en el directorio de plugins de SketchUp, puede hacer que SketchUp realice todo tipo de cosas:

- Automatizar tareas comunes, como la generación de escenas de un conjunto de rotaciones.
- Adjuntar atributos para los elementos de dibujo como el costo, proveedor, etc.
- Leer atributos para generar informes, listas de materiales, etc.
- ... y mucho más.

Para empezar a experimentar con los comandos y los métodos de Ruby lo más recomendable es utilizar la consola de ruby ya que puedes introducir comandos sin necesidad de crear un fichero ruby aparte. Para acceder a la consola debemos entrar en la pestaña "Windows" y dentro de ella encontraremos dicha consola.

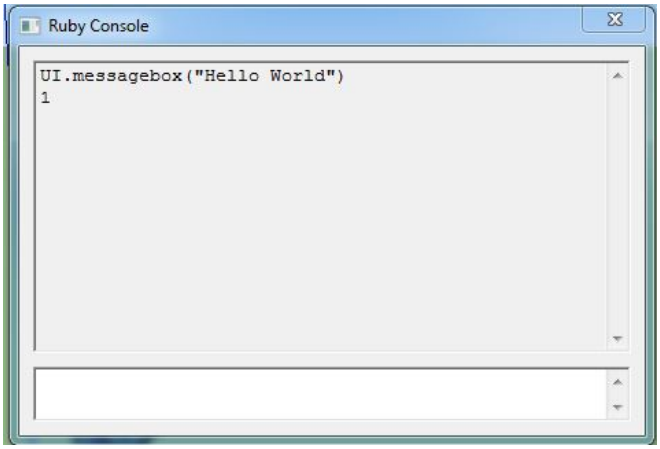


Figura 2.18: Consola ruby en Google Sketchup.

2.3 Fases del desarrollo

Para realizar este proyecto hemos decido dividirlo en dos partes y utilizar un fichero como punto de unión entre las dos. Las dos partes son las siguientes:

1. Detección de paredes, puertas y ventanas de un plano 2d
2. Representación en 3d mediante Ruby

2.3.1 Detección de paredes, puertas y ventanas de un plano 2d

Esta parte trata de obtener la imagen del plano de la vivienda en 2d y detectar las paredes, puertas y ventanas utilizando técnicas de tratamiento digital de imágenes, una vez detectado todo lo necesario obtenemos las coordenadas referentes a cada objeto y las almacenamos en un fichero de texto (*.txt). Todo ello está implementado con Matlab.

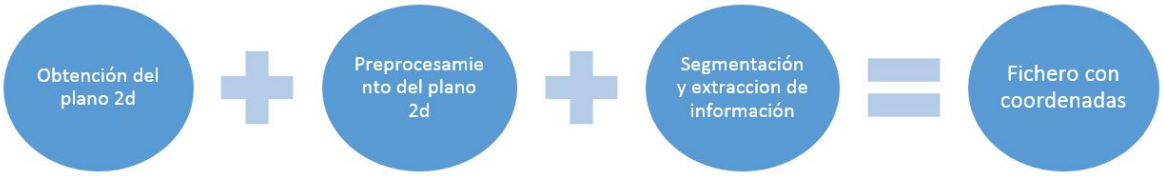


Figura 2.19: Desglose por fases de la detección y extracción de información del plano 2d.

Obtención del plano 2d: Se dispone de un plano de una vivienda en 2d, este deber ser una imagen.

Procesamiento del plano 2d: La función básica es mejorar la imagen de forma que se aumente las posibilidades de éxito en los siguientes procesos. En nuestro caso debemos

aplicar las técnicas de mejorar el contraste, eliminar el ruido y aislar áreas cuya textura indica la probabilidad de información.

La mayoría de procesamiento de imágenes se aplica directamente sobre imágenes en escala de grises, debido al bajo consumo de cómputo de éste; la mayoría de métodos matemáticos deterministas que se utilizan en su procesamiento, están basados en la diferencia de niveles de grises, por lo que no existen muchos métodos para procesamiento de imágenes en color, aun cuando estos pueden ser utilizados en este tipo de formato de imagen.

- Reducción del área de trabajo: Consiste en simplificar la imagen a la región que corresponde al objeto de interés
- Realce del contraste: es necesario realzar el contraste de estas imágenes antes de un procesamiento posterior o de realizar un análisis.
- Reducción de ruido: El ruido es información no deseada que contamina la imagen. Este aparece durante el proceso de adquisición y digitalización, haciendo necesario implementar un método de reducción de ruido, que retenga tanto como sea posible las características de importancia.

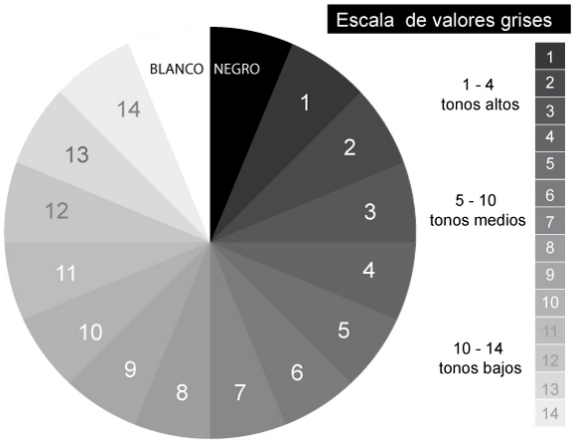


Figura 2.20: Círculo cromático en escala a grises.

Segmentación y extracción de la información: Es el proceso de dividir una imagen digital en varias partes (grupos de píxeles) u objetos. El objetivo de la segmentación es simplificar y/o cambiar la representación de una imagen en otra más significativa y más fácil de analizar. La segmentación se usa tanto para localizar objetos como para encontrar los límites de estos dentro de una imagen.

Para guardar los datos validos primero debemos de pasar por otra etapa que está basada en el reconocimiento e interpretación. Reconocimiento en nuestro caso sería por ejemplo

obtener una línea recta y aplicarle la etiqueta de pared. Un borde se caracteriza por presentar una transición de claro a oscuro o de oscuro a claro. Existen varios procesos para la detención de bordes (paredes del plano) que veremos más adelante.

$$\text{Roberts:}\quad R_{x'} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, \qquad R_{y'} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$\text{Sobel:}\quad S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

$$\text{Prewitt:}\quad P_x - \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, \quad P_y - \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Figura 2.21: Diferentes operadores

Fichero con coordenadas: Fichero donde se almacena toda la información obtenida. Este fichero contiene las coordenadas de una pared como de una ventana o puerta. Debemos de almacenar estructuradamente toda la información obtenida de la imagen para después realizar un script[2] en Ruby que sea capaz de leer esos datos y crearte la casa reflejada en el plano.

2.3.2 Representación en 3d mediante Ruby

Esta parte se encarga de interpretar el fichero que contiene las coordenadas de las paredes, puertas y ventanas (resultado de la fase anterior) y representarlas en google Sketchup. Para todo esto hemos implantando un plugin desarrollado con la API de google Sketchup que utiliza como lenguaje de programación Ruby.

Figura 2.22: Desglose por fases de la representación en 3d (Creación plugin).

[2]Conjunto de instrucciones que deben ser interpretadas línea a línea en tiempo real para su ejecución

2.4 Estado del arte

Hoy en día, una gran cantidad de usuarios utilizan la representación 3D para poder visualizar de forma más realista sus viviendas, existen muchas aplicaciones que permiten representar en 3D una vivienda, pero todas ellas tienen en común el gran trabajo que se requiere por parte del usuario, ya que todos ellos obligan a diseñar el plano en 3D manualmente. A continuación vemos algunas aplicaciones:

- **Homestyler:** Autodesk Homestyler es una interesante y sencilla herramienta on-line de los creadores de AutoCAD. Accediendo al sitio web de Autodesk Homestyler podremos diseñar, de forma totalmente intuitiva, nuestro hogar de ensueño.

  El diseño puede realizarse en 2D y luego levantar el plano para poder verlo en 3D. Las posibilidades de modificación y personalización de paredes y elementos externos a la casa van a permitirnos añadir setos, árboles, jardineras o personalizar tanto las formas como los colores de las paredes.

  El funcionamiento de la herramienta se basa en arrastrar y soltar los distintos elementos que queramos colocar en nuestro hogar. La paleta de muebles, complementos y elementos de la casa es bastante amplia. Se incluye también el catálogo de algunos fabricantes de muebles de baño y cocina estadounidenses.

  El sitio web está realizado utilizando la tecnología Adobe Flash, con lo cual, el único requisito para poder utilizarlo es tener instalado el plugin de Flash en nuestro navegador. Podemos registrarnos en el sitio para guardar nuestros diseños y modificarlos posteriormente o simplemente probarlo creando un diseño desde cero o bien modificar alguno de los ya existentes en su galería.

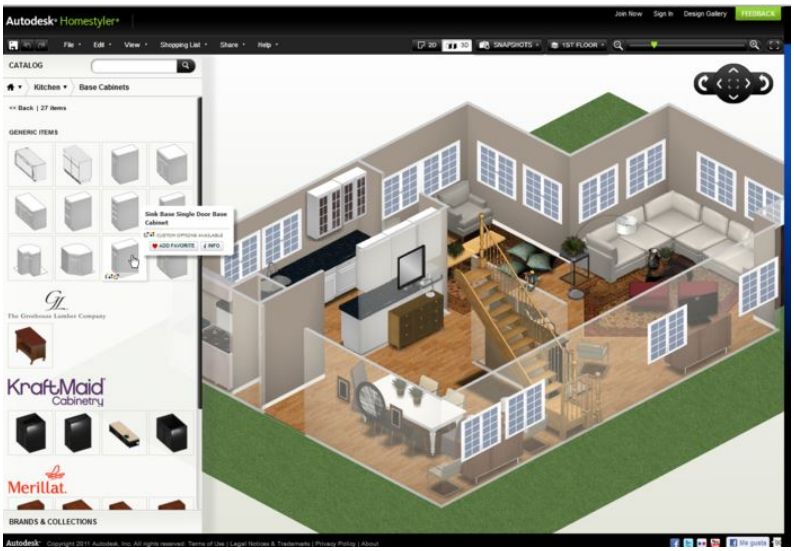


Figura 2.23: Aplicación homestyler

- **AutoCad:** es un software CAD utilizado para dibujo 2D y modelado 3D. Actualmente es desarrollado y comercializado por la empresa Autodesk. El nombre AutoCAD surge como creación de la compañía Autodesk, en que Auto hace referencia a la empresa creadora del software y CAD a Dibujo Asistido por Computadora (por sus siglas en inglés "Computer Assisted Drawing"), teniendo su primera aparición en 1982.1 AutoCAD es un software reconocido a nivel internacional por sus amplias capacidades de edición, que hacen posible el dibujo digital de planos de edificios o la recreación de imágenes en 3D; es uno de los programas más usados por arquitectos, ingenieros, diseñadores industriales y otros. Además de acceder a comandos desde la solicitud de comando y las interfaces de menús, AutoCAD proporciona interfaces de programación de aplicaciones (API) que se pueden utilizar para determinar los dibujos y las bases de datos.

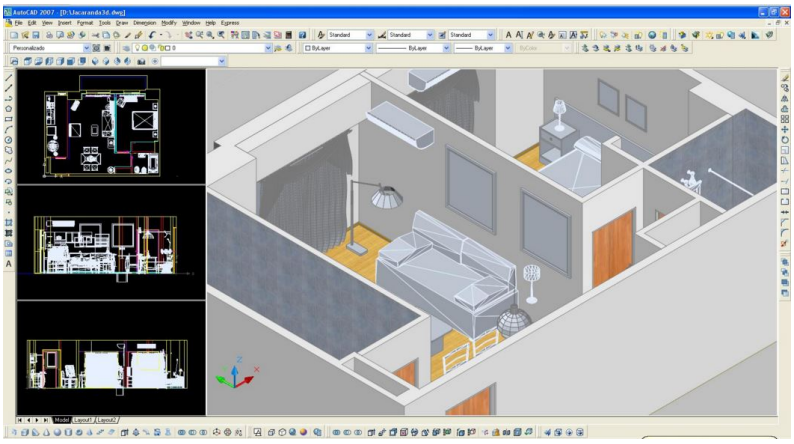


Figura 2.24: Aplicación AutoCad

- **Floorplanner:** Floorplanner permite crear y publicar planos interactivos en línea fácilmente. Los usuarios más comunes de la herramienta de dibujo de Floorplanner son agentes inmobiliarios o empresas que proporcionan servicios de dibujo y diseño para agentes inmobiliarios. Sin embargo, la herramienta es tremendamente útil para cualquier profesional del diseño. Te permite la posibilidad de trabajar sobre el interior de nuestra casa o proyecto sea cual sea el mismo. También permite integrar los planos de Floorplanner para trabajarnos desde tu móvil.

Figura 2.25: Aplicación Floorplanner

- **Sweet Home 3D:** Sweet Home 3D es un editor CAD de ingeniería, arquitectura y construcción bajo licencia GNU General Public License para el diseño de los muebles de una vivienda en un plano 2D, y una vista previa en 3D. Desarrollado en Java este software está disponible en formulario de una aplicación Java Web Start que requiere la preinstalación de Java virtual machine, o en forma de una aplicación con una máquina virtual Java (para evitar al usuario instalar un Java propio). Este software se ejecuta en cualquier plataforma con una máquina virtual Java y la biblioteca Java 3D, que le permite ser ligero y funcionar en Mac OS X, Windows, Linux, Solaris y posiblemente otros...

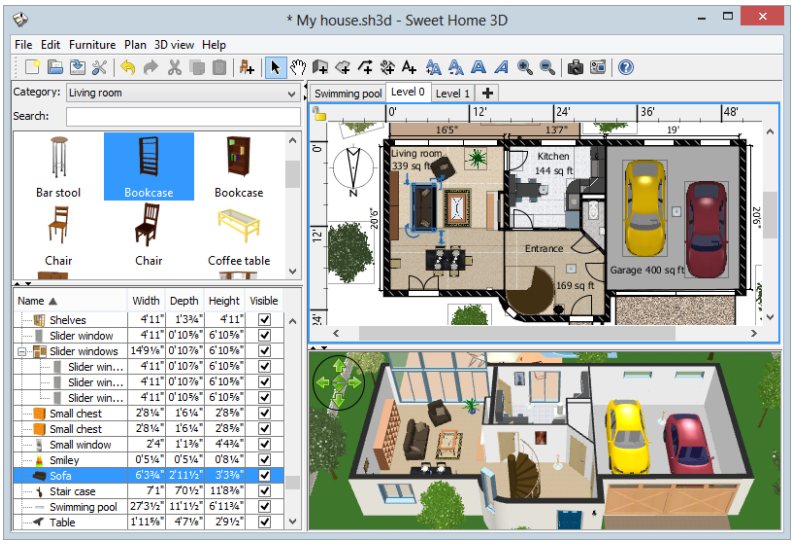


Figura 2.26: Aplicación Sweet Home 3D

2.4.1 Tabla comparativa

La tabla comparativa se ha realizado asignándole un valor del 1 al 10 a cada aplicación, siendo 10 la máxima puntuación y 1 la mínima según se cumplan dichos aspectos.

Aspectos a evaluar	HomeStyler	AutoCad	Floorplanner	Sweet Home 3D	TFG
Representación 3D automática	1	1	1	1	10
Interfaz intuitiva	8	5	9	7	9
Rapidez en representación 3D	10	7	10	10	1
Obtiene resultados correctos	10	10	10	10	5
Permite insertar objetos	10	10	10	10	1

Table 2.1: Tabla comparativa de aplicaciones

En el mercado no existe ninguna aplicación similar a la realizada en este proyecto. Lo más parecido que ofrecen son funcionalidades como las explicadas anteriormente. Por último cabe desatacar que el valor de resultados correctos se puede mejorar si se desarrollan las mejoras expuestas en trabajos futuros.

3

HERRAMIENTAS Y LENGUAJES DE PROGRAMACIÓN UTILIZADOS

3.1 Herramientas utilizadas

Para la creación de este proyecto hemos utilizado las siguientes herramientas:

- Ordenador personal
- Matlab 7
- Aplicación de google sketchUp
- Planos de viviendas

3.1.1 Ordenador Personal

El ordenador personal que se ha utilizado para la realización del proyecto es un ordenador portátil marca ASUS que contiene las siguientes características:

- Procesador Intel Core i3
- Memoria Random-Access Memory (RAM) 4Gigabyte (GB)
- Sistema Operativo Windows 7 Home Premium 64 bits
- Tarjeta Gráfica AMD Radeon HD 7470M 1GB
- Disco duro 500GB

3.1.2 Matlab 7

Es una herramienta de software matemático que ofrece un Entorno de Desarrollo Integrado (IDE) con un lenguaje de programación propio (lenguaje M). Es un lenguaje de alto nivel desarrollado por Math Works cuyas funciones están optimizadas para el uso de matrices. Gracias a la diversidad de sus librerías aplicables a diversos campos del conocimiento, este programa es ampliamente utilizado en la educación y en la industria. Los usos más familiares de Matlab son:

- Matemática y Computación
- Desarrollo de algoritmos
- Modelamiento, simulación y prototipado
- Gráficas científicas e ingenieriles
- Desarrollo de aplicaciones, incluyendo construcción de interfaces gráficas de usuario
- Análisis de datos, exploración y visualización

El Toolbox de Procesamiento de Imágenes proporciona a MATLAB un conjunto de funciones que amplía las capacidades del producto para realizar desarrollo de aplicaciones y de nuevos algoritmos en el campo del proceso y análisis de imágenes.

El entorno matemático y de creación de MATLAB es ideal para el procesado de imágenes, ya que estas imágenes son, al fin y al cabo, matrices. Este toolbox incorpora funciones para:

- Diseño de filtros.
- Mejora y retocado de imágenes.
- Análisis y estadística de imágenes.
- Operaciones morfológicas, geométricas y de color.
- Transformaciones 2D, etc.

3.1.3 Aplicación de google sketchUp

Es un programa de diseño gráfico y modelado en 3D basado en caras. Para entornos arquitectónicos, ingeniería civil, diseño industrial, Sistema de Información Geográfica (SIG), videojuegos o películas.
Sketchup publica el lenguaje en el que está escrito en Ruby los comandos para que los usuarios puedan escribir segmentos de programa para cambiar la funcionalidad. Existe una gran

variedad de estos con aplicaciones particulares como el dibujo automatizado de techumbres[1], piezas de acero, etc.
La última versión (2015) de SketchUp funciona bajo Windows Vista, Windows 7 y Windows 8 y en entornos OS X Mac OS 10.7 o superior. Versiones anteriores (8 o 2013) funcionan con Windows XP o Mac OS X 10.4 (Tiger) o superior. Esta última versión trae cambios en el diseño de su logo tradicional y agrega nuevas herramientas al programa. Aún no hay una versión disponible para Linux. .

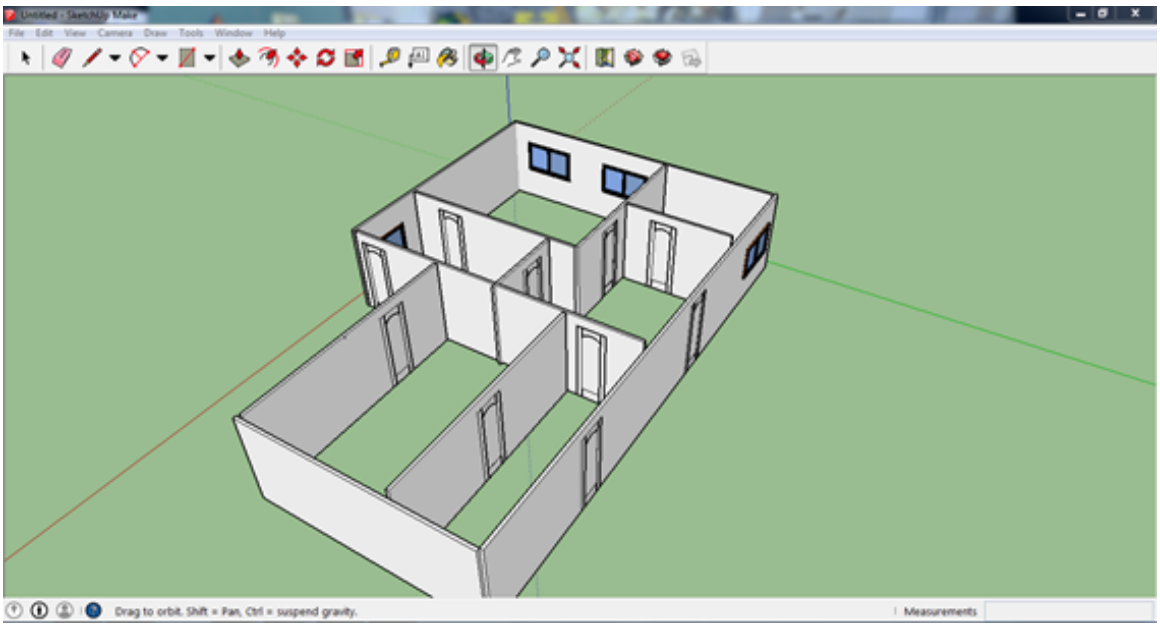


Figura 3.1: Interfaz gráfica de google SketchUp

3.1.4 Planos de viviendas

Un plano es un dibujo en 2d donde se muestran las estructuras de una obra, con los señalamientos específicos en cuanto a medidas y distribución de espacios.

[1]Conjunto de elementos que conforman la parte superior de una edificación, que la cubre y cierra.

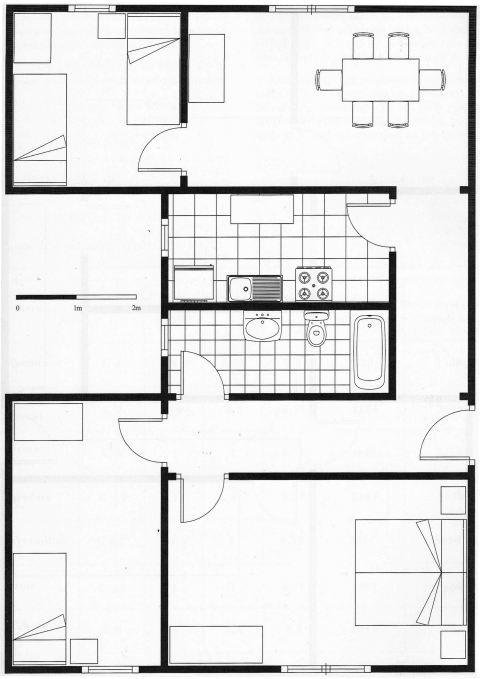


Figura 3.2: Plano de una vivienda

3.2 Lenguajes de programación utilizados

3.2.1 Ruby

[16]

Ruby es un lenguaje de programación interpretado, reflexivo y orientado a objetos .Su creador, Yukihiro "Matz" Matsumoto, mezcló partes de sus lenguajes favoritos (Perl, Smalltalk, Eiffel, Ada y Lisp) para formar un nuevo lenguaje que incorporara tanto la programación funcional como la imperativa. A menudo ha manifestado que está "tratando de hacer que Ruby sea natural, no simple", de una forma que se asemeje a la vida real.

Desde su liberación pública en 1995, Ruby ha atraído devotos desarrolladores de todo el mundo. En el 2006, Ruby alcanzó reconocimiento masivo, formándose grupos de usuarios activos en las ciudades más importantes del mundo y llenando las capacidades de las conferencias relacionadas a Ruby.

Su implementación oficial es distribuida bajo una licencia de software libre.

Las principales características son:

- Orientado a objetos
- Cuatro niveles de ámbito de variable: global, clase, instancia y local.

- Manejo de excepciones.
- Posibilidad de redefinir los operadores.
- Altamente portable.
- Amplia librería estándar.
- Soporta alteración de objetos en tiempo de ejecución.

3.2.2 Matlab

[17]

La programación se lleva a cabo mediante un lenguaje que es muy parecido a lenguajes de alto nivel como Beginner's All-purpose Symbolic Instruction Code (BASIC) o C. Esto permite que el usuario pueda agrupar sentencias que utiliza frecuentemente dentro de un programa que puede ser invocado posteriormente. De este modo se ahorra tiempo y esfuerzo en sucesivas sesiones pues no es necesario escribir todas las sentencias de nuevo como se mostrará.
Se maneja (en su mayor parte) escribiendo sentencias dentro de una ventana llamada de órdenes. Al arrancar el programa aparecen varias ventanas, pero en una primera toma de contacto es mejor cerrar todas excepto la principal, que es la ventana de órdenes. Las órdenes se escriben una a una pulsando la tecla de retorno al final. Por ejemplo, si se escribe **sqrt(16)** el programa realiza la operación indicada (raiz cuadrada) y responde en la pantalla con el resultado.
Lo que se aprecia en la ventana de órdenes será algo como:

```
1  % Raiz cuadrada de 20
2 >> sqrt(20)
3 ans =
4 4.4721
```

En MATLAB todos los objetos son matrices de números complejos en punto flotante. Un escalar no es más que una matriz 1 × 1. Cada elemento de la matriz se almacena en la memoria como un número complejo $a + bi$ siendo $\sqrt{-1}$. Cuando se trabaja con números reales MATLAB considera simplemente que b = 0. Finalmente, todos los números son tratados en formato de punto flotante. Esto quiere decir que son números con parte entera, parte decimal y un exponente.

4

Diseño e Implementación

[20] En este capítulo se va a explicar el proceso seguido para el diseño e implementación del proyecto, tanto de la aplicación capaz de detectar puertas, ventanas y paredes en un plano 2D como del plugin creado para la generación en 3D. Debemos indicar que la aplicación está desarrollada en MATLAB y el plugin en ruby.

4.1 Detección de objetos en un plano 2D con Matlab

En esta sección vamos a explicar el diseño e implementación de la aplicación desarrollada en Matlab. Esta aplicación tiene como entrada un plano en 2D y es capaz de detectar las puertas, ventanas y paredes de dicho plano para posteriormente guardar sus coordenadas en un fichero de texto, este fichero de texto será el punto de unión con el plugin que explicaremos en la sección 5.2.

4.1.1 Características del plano de entrada

El plano de entrada debe cumplir una serie de características para que la aplicación sea capaz de reconocer las ventanas, paredes y puertas.
Para que la aplicación reconozca perfectamente las **puertas** deben estar su base en color rojo, es decir toda línea de color rojo la aplicación la interpretará como puerta.

Las **ventanas** deben de tener su base de color azul para que la aplicación sea capaz de reconocerlas, toda línea de color azul la aplicación la detectará como ventana.

Por último tenemos las **paredes**, estas deben de ser de color azul y deben tener un grosor mayor que los objetos secundarios del plano, como pueden ser un sofá, una mesa etc...

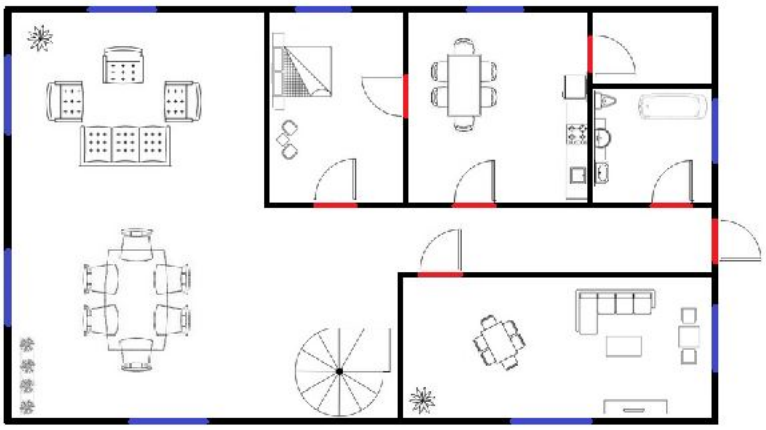


Figura 4.1: Ejemplo plano de entrada

Figura 4.2: Ejemplo ventana

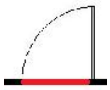


Figura 4.3: Ejemplo puerta

4.1.2 Detección de paredes

En este apartado se explicarán las diferentes técnicas y fases utilizadas para detectar paredes en el plano 2D.

4.1.2.1 Pre-procesamiento

4.1.2.1.1 Binarización y eliminación de ruido

Binarización

[8]

La imagen será binarizada mediante la técnica de detección de umbral. Hemos elegido esta técnica ya que el uso de umbrales en el análisis de imágenes constituye una de las principales técnicas en los sistemas de visión artificial para la detección de objetos. Suponiendo que se quiere aislar un objeto del entrono mediante umbralización, será necesario seleccionar un umbral adecuado que separe los dos tonos de intensidad.

Para obtener dicho umbral, uno de los métodos más utilizado es el conocido método de Otsu, que utiliza métodos estadísticos para la resolución de dicho problema. Este método proporciona el umbral óptimo ("threshold") para la segmentación de la imagen, bajo el criterio de máxima varianza entre fondo ("background") y objeto ("foreground").

Así, se calcula la varianza entre todas las posibles divisiones, y se toma el umbral que representa

la máxima varianza entre clases, de tal modo que:

$T = max(\sigma^2)$ ... Umbral optimo

siendo

$\sigma^2 = w_B(\mu_B - \mu)^2 + w_F(\mu_F - \mu)^2$ ... varianza

$w_k = \sum_{i=0}^{k} pi$ ... probabilidad acumulada

$\mu_k = \sum_{i=0}^{k} i.p_i$ ... media acumulada

$\mu_c = \frac{\mu_k}{w_k}$ ... media de la clase (B o F)

Donde p_i es la probabilidad de aparición de un determinado nivel i.

Implementación en Matlab

Para la binarización de nuestro plano de una vivienda utilizamos la función *graythresh*, esta función calcula el umbral T mediante la aplicación del método de Otsu, introduciendo una imagen de intensidades *imagen* como parámetro de entrada. Además, la función im2bw binariza la imagen con el umbral T previamente hallado, obteniendo una imagen binaria.

```
1  %Binarizacion imagen
2  imagen=rgb2gray(imagen);
3  threshold = graythresh(imagen);
4  imagen =im2bw(imagen,threshold);
```

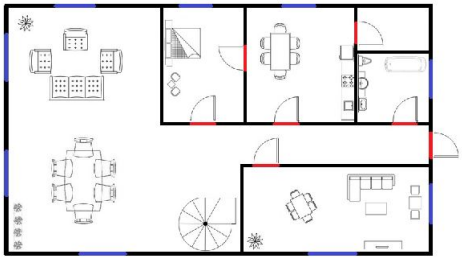


Figura 4.4: Plano original

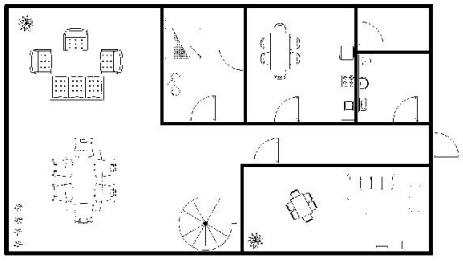


Figura 4.5: Plano binarizado

Eliminación de ruido

Eliminar el ruido en las imágenes es muy importante ya que podemos eliminar los objetos indeseados, en nuestro caso consideramos ruido cualquier objeto que no signifique una pared. Por ejemplo una cama, mesa, etc..

Implementación en Matlab

Para ello utilizamos la función *bwareaopen*, esta función elimina de una imagen binaria todos los componentes que no están interconectados, produciendo otra imagen binaria. Esta operación se conoce como una abertura área.

```
1  %Eliminacion de ruido
2  imagen = bwareaopen(imagen,P);
```

Donde P es el número de píxeles que se desean remover de una imagen binaria. La variable *imagen* contiene una imagen binarizada y esto produce otra imagen binaria almacenada en una nueva variable llamada *imagen*.

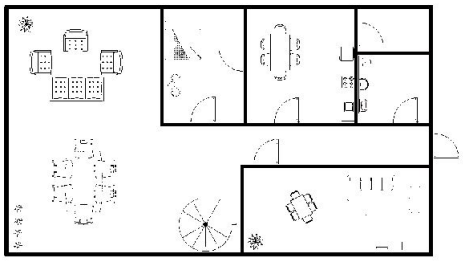


Figura 4.6: Plano binarizado

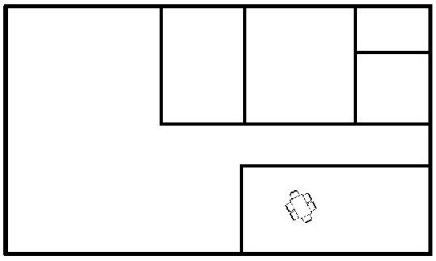


Figura 4.7: Plano binarizado y sin ruido

4.1.2.2 Segmentación

4.1.2.2.1 Detección de bordes

[8] [11] [12]
Una vez que se ha eliminado todo lo que no interesa como puede ser mesas, camas, etc, y se han resaltado las líneas de interés debemos utilizar las técnicas de segmentación para obtener los objetos que van a ser de utilidad, en este caso las líneas rectas que hacen referencia a las paredes. Para poder detectar las lineas se va a utilizar la técnica de detección de bordes, para entrar mas en profundidad sobre el funcionamiento teórico de esta técnica véase la sección *2.1.2.1 Detección de discontinuidades*.

Implementación en Matlab
Para ello utilizamos la función **edge** (disponible en el toolbox de Matlab). Esta función detecta todos los bordes de una imagen. La salida de esta función se corresponde con una matriz binaria, de tal manera que en cada posición de la matriz, se encontrará un 1 o un 0 dependiendo de si se ha detectado un borde o no.

- **1=** se ha detectado un borde (sobre el fondo de imagen el píxel se representa de color blanco).
- **0=** no se ha detectado ningún borde (sobre el fondo de imagen el píxel se representa de color negro).

La función **edge** presenta diversos métodos para la detección de bordes:

- **Canny**: Encuentra bordes mediante la búsqueda de los máximos locales de la pendiente de I. Calcula el gradiente utilizando el derivado de un filtro de Gauss. Este método utiliza

dos umbrales para detectar bordes fuertes y débiles, incluyendo bordes débiles en la salida si están conectados a los bordes fuertes. Mediante el uso de dos umbrales. El método de Canny es menos probable que los otros métodos que se deje engañar por el ruido, y más probabilidades de detectar verdaderos bordes débiles.

- **Log (Laplaciano de Gauss)**: Encuentra bordes mediante la búsqueda de cruces por cero después de filtrar I con un Laplaciano del filtro de Gauss.

- **Prewitt**: Encuentra bordes usando la aproximación Prewitt al derivado. Devuelve bordes en aquellos puntos en los que la pendiente de I es el máximo.

- **Roberts**: Encuentra bordes usando la aproximación Roberts a la derivada. Devuelve bordes en aquellos puntos en los que la pendiente de I es el máximo.

- **Sobel**: Encuentra bordes usando la aproximación Sobel al derivado. Devuelve bordes en aquellos puntos en los que la pendiente de I es el máximo.

Para la resolución del proyecto se ha utilizado el operador **Prewitt**, por el hecho de ser poco sensible al ruido y tener la virtud de ofrecer la magnitud y orientación (vertical, horizontal) del borde, esto último es muy importante para la resolución del proyecto.

```
1   %Deteccion bordes verticales
2   bn = edge(imagen,'prewitt','vertical')
```

En nuestro caso siendo *imagen* una imagen pre-procesada para la **detección de paredes**.

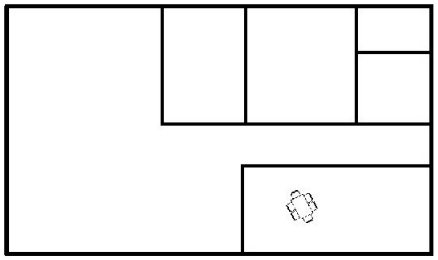


Figura 4.8: Plano pre-procesado

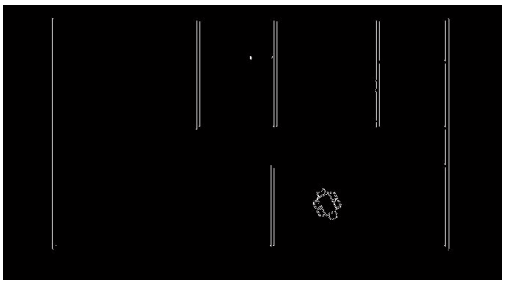


Figura 4.9: Detección bordes verticales (paredes)

```
1   %Deteccion bordes horizontales
2   bn = edge(imagen,'prewitt','horizontal')
```

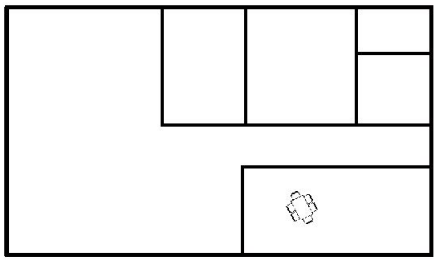


Figura 4.10: Plano pre-procesado

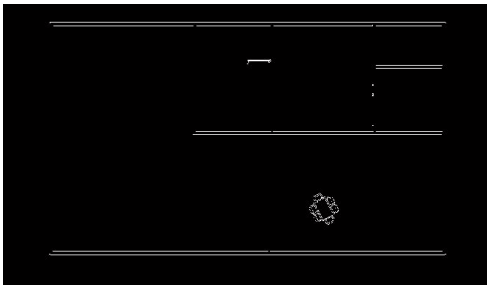


Figura 4.11: Detección bordes horizontales (paredes)

4.1.2.3 Extracción de información

[8] [9] [10]
Una vez segmentada la imagen, en la que podemos encontrar divididos los bordes horizontales de los verticales llegamos a esta última fase en la que se detectará dichos bordes y se extraerá las coordenadas (x, y) donde comienza y finaliza cada borde (paredes), es decir por cada borde encontrado se obtendrá dos pares de coordenadas.
Veamos un ejemplo en el que imaginamos que la línea recta negra es un borde vertical encontrado.

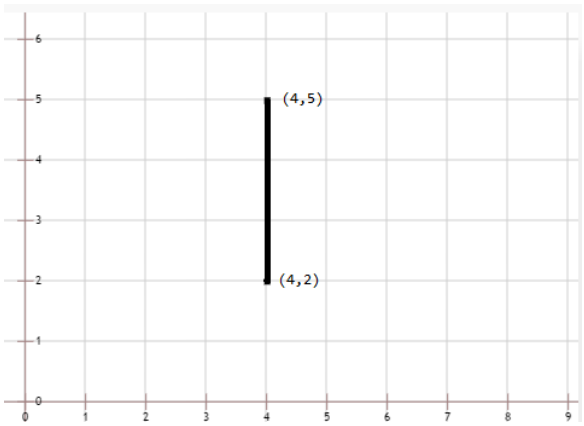


Figura 4.12: Ejemplo borde en eje de coordenadas

En este ejemplo la información a extraer sería los dos pares de coordenadas $(4,2)$ y $(4,5)$. Estas coordenadas se almacenarán en un archivo de texto como pared vertical.
En nuestro caso para identificar las líneas rectas en la imagen que representan paredes utilizaremos el método de la **transformada de Hough**. Para más información sobre la **transformada de Hough** véase el apartado *2.1.2.2 Enlazado de bordes y detección de límites*.

Implementación en Matlab
Para ello utilizamos la función **hough** (disponible en el toolbox de Matlab), dicha función puede

adquirir los siguientes parámetros de entrada:
[H,T,R]=hough(f,dtheta,drho)

- **f**: Imagen de bordes binaria.
- **dtheta**: Espaciado (en grados) de la transformada de Hough a lo largo del eje θ.
- **drho**: Espaciado de la transformada de Hough en el eje p.

En la salida se obtendrá.

- **H**: Matriz con la transformada de Hough.
- **T**: Vector que contiene el ángulo (en grados) correspondiente a cada columna de H.
- **R**: Vector que contiene los valores de p correspondiente a cada fila de H.

Para la resolución del proyecto solo especificamos la imagen ya segmentada **(bn)**, al no especificar los demás parámetros la función asigna los valores por defecto.

```
1  %Transformada de hough
2  [H,T,R]=hough(bn);
```

Una vez obtenida la matriz H y los vectores θ (T) y p (R) aplicamos la función **houghpeaks** (disponible en el toolbox de Matlab) que nos selecciona las celdas (coordenadas (x, y)) del espacio de parámetros que mayor valor tienen. Esta función recibe cuatro parámetros al ser llamada:
[r,c,hnew]=houghpeaks(h,numpeaks,threshold,nhood)

- **h**: Matriz con la transformada de Hough.
- **numpeaks**: Máximo número de picos que debe buscar.
- **threshold**: Umbral a partir del cual una celda no será considerada como pico. Por defecto se tendrán en cuenta todas las líneas cuya longitud en puntos sea como mínimo la mitad de la línea más larga detectada
- **nhood**: vector de dos elementos que específica el tamaño para suprimir píxeles vecinos (debe ser positivo entero y par). El entorno de vecindad alrededor de cada pico identificado se pone a cero.

En la salida se obtendrá:

- **r,c**: Vectores con las filas y las columnas de las coordenadas de los picos identificado.
- **hnew**: Transformada de hough con los píxeles vecinos a un pico suprimidos.

En nuestro caso especificamos que como máximo puede encontrar 40000 líneas, y el umbral a partir del cual una celda no será considerada como pico es de 63, siendo H la matriz con la transformada de Hough.

```
1  %Deteccion maximos
2  peaks=houghpeaks(H,40000,'threshold',63);
```

Como hemos visto con la función Houghpeaks, podemos seleccionar la cantidad de segmentos a detectar y la longitud de los segmentos detectados en base al segmento de mayor longitud detectado.

Por último se utiliza la función HoughLines, que se encarga de extraer los segmentos de las líneas basándose en la transformada de Hough. Esta función recibe seis parámetros al ser llamada:
lines=houghlines(IB,Theta,Rho,Peaks,fillgap,minlength)

- **IB**: Es la imagen sobre la que se detectarán los segmentos.
- **Theta**: Es el array de valores θ (en grados) que devuelve la función Hough y que indica los valores de θ sobre los que fue generada la matriz H de la transformada de Hough.
- **Rho**: Es el array de valores que devuelve la función Hough y que indica los valores de p sobre los que fue generada la matriz H de la transformada de Hough.
- **Peaks**: Es una matriz que genera la función hougpeaks y que contiene las coordenadas en fila y columna de las celdas de la transformada de Hough usadas para la búsqueda de los segmentos.
- **fillgap**: Separación de píxeles a partir de la cual considera segmentos distintos (valor por defecto 20).
- **minlength**: Número de píxeles mínimo que debe tener un segmento para ser considerado como tal (valor por defecto 40).

Como resultado:

- **lines**: Estructura que devuelve esta función de longitud igual al número de segmentos de línea encontrados. Cada elemento de esta estructura tiene los siguientes campos.
  - **point1**: Punto inicial del segmento; vector de dos elementos.
  - **point2**: Punto final del segmento; vector de dos elementos.
  - **length**: Distancia entre point 1 y point 2.

Para la resolución del proyecto especificamos que la separación de píxeles a partir de la cual considera segmentos distintos es de 5px, también que el número de píxeles mínimo que debe tener un segmento para ser considerado como tal es de 10px.

```
1  %Extraccion de lineas
2  lines=houghlines(bn,T,R,peaks,'fillgap',5,'minlength',10);
```

El resultado de esta última función nos devuelve las coordenadas de las paredes detectadas, dichas coordenadas se guardarán en un fichero de texto para su posterior representación en 3d.

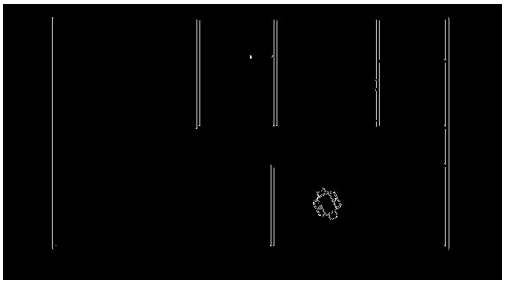


Figura 4.13: Detección bordes verticales (paredes)

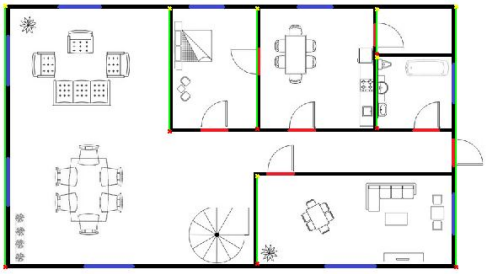


Figura 4.14: Detección de paredes verticales

Para detectar las paredes horizontales se debería realizar una segmentación sobre líneas horizontales (edge) y al resultado aplicarle la transformada de hough.

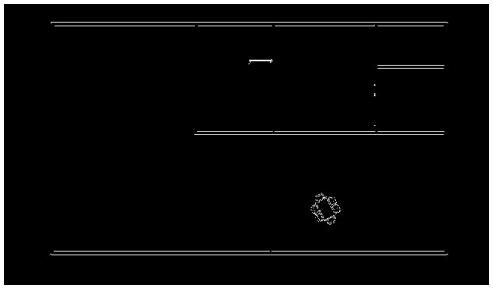


Figura 4.15: Detección bordes horizontales (paredes)

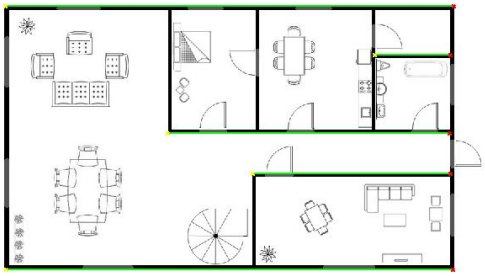


Figura 4.16: Detección de paredes horizontales

4.1.3 Detección de puertas

En este apartado se explicarán las diferentes técnicas y fases utilizadas para detectar puertas en el plano 2D.

4.1.3.1 Pre-Procesamiento

En la fase de detección de puertas no se aplica ningún proceso de pre-procesamiento, ya que todas las técnicas se utilizan en la fase de segmentación.

4.1.3.2 Segmentación

[7]

Para la detección de puertas debemos de tener en cuenta que dichas puertas son representadas con el color rojo, a consecuencia de esto vamos a utilizar un algoritmo de segmentación de imágenes en color. Para la detección de líneas rojas debemos buscar la banda de interés, en este caso es la banda roja. Para esto es necesario descomponer la imagen RGB original en 3 imágenes simples que representan a cada banda.

Implementación en Matlab

```
1    imR=double(im(:,:,1));
2    imG=double(im(:,:,2));
3    imB=double(im(:,:,3));
```

Se realiza además una transformación al tipo de variable "double" para poder realizar operaciones aritméticas con las matrices, pues con imágenes uint8 no es posible. Un píxel rojo se caracteriza por tener un valor elevado en la banda roja, y valores menores en las bandas verde y azul. De esta forma el rojo ideal corresponde al valor RGB 255,0,0. Pero también es un rojo intenso un valor RGB 195,23,14. En base a este razonamiento podemos afirmar que no será posible "aislar" cualquier tipo de rojo analizando sólo la banda roja, para ello necesitamos imponer una condición lógica que considere las 3 bandas en sus conjunto.

El criterio utilizado es que la banda roja menos las bandas verde y azul debe ser a lo menos mayor a 20, un valor por debajo de 20 no se considerará como rojo. Por ejemplo: Un punto RGB que cumple este criterio sería 255,155,79. (En este caso 255-155-79=21). Obviamente todos los puntos donde la banda roja es más predominante también cumplen con el criterio: (255,10,5), (192,20,32), etc.

```
1    imR2=(imR-imG-imB);
```

Una vez establecido un umbral para decidir qué punto RGB se considera como rojo (en nuestro caso 20) procedemos a segmentar la imagen, la idea principal es que todos los puntos que cumplen dicha condición quedan con valor "1" y el resto con valor "0". La instrucción masc=(imR2>20) permite lograr una imagen binaria que representa a la "máscara" de todos los puntos rojos que cumplen la condición. Una vez definida la máscara realizamos una multiplicación elemento a elemento de toda la matriz de la imagen con la máscara.

```
masc=(imR2>20);
imR2=imR2.*masc;
```

En las siguientes figuras *4.17* y *4.18* vemos el resultado de segmentar la imagen por color rojo.

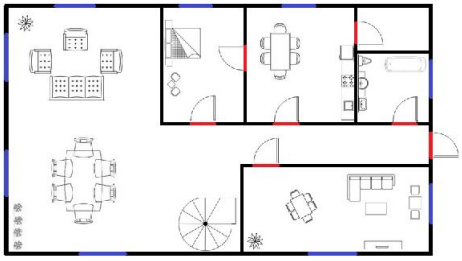


Figura 4.17: Plano original

Figura 4.18: Segmentación por color rojo (puertas)

Una vez segmentada la imagen se deben detectar los bordes horizontales como verticales, para ello se utiliza la misma técnica que se ha utilizado en el apartado *4.1.2.2.1 Detección de bordes*.

Figura 4.19: Segmentación de puertas (color rojo)

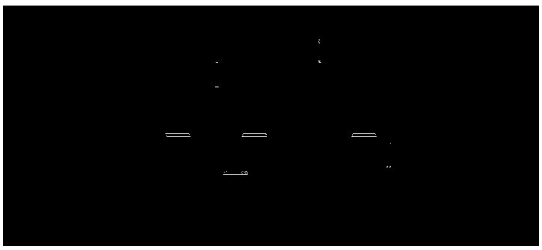


Figura 4.20: Detección bordes verticales (puertas)

Figura 4.21: Segmentación de puertas (color rojo)

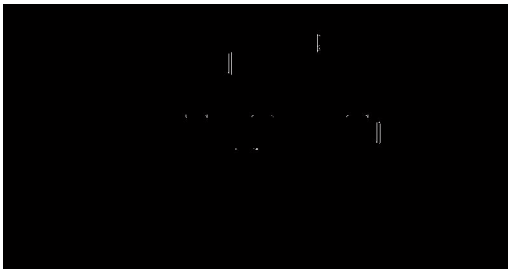


Figura 4.22: Detección bordes horizontales (puertas)

4.1.3.3 Extracción de información

Para la extracción de información se utiliza la misma técnica que en el apartado *4.1.2.3 Extracción de información*.
En el siguiente ejemplo vemos como partiendo de un plano segmentado para la detección de puertas se obtiene las coordenadas de las puertas horizontales y se le asigna una etiqueta a cada una de ellas, en este caso una línea amarilla.

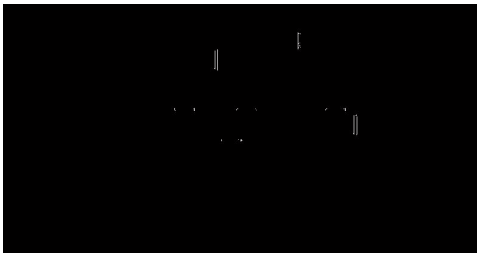


Figura 4.23: Bordes de puertas verticales

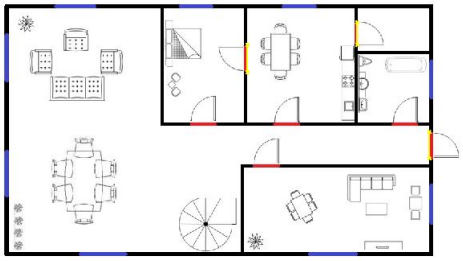


Figura 4.24: Detección puertas verticales (puertas)

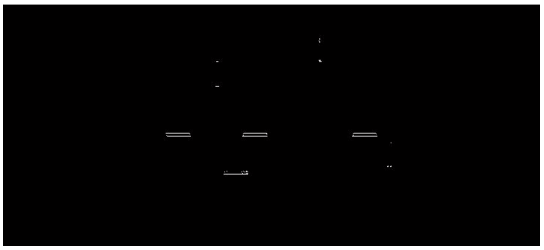


Figura 4.25: Bordes de puertas horizontales

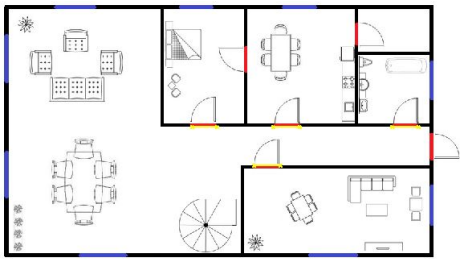


Figura 4.26: Detección puertas horizontales (puertas)

4.1.4 Detección de ventanas

En este apartado se explicarán las diferentes técnicas y fases utilizadas para detectar ventanas en el plano 2D.

4.1.4.1 Pre-procesamiento

En la detección de ventanas no se aplica ningún proceso de pre-procesamiento, ya que todo el proceso de detección se realiza en la fase de segmentación.

4.1.4.2 Segmentación

[7]
Para la detección de ventanas debemos de tener en cuenta que dichas ventanas son representadas con el color azul, a consecuencia de esto vamos a utilizar un algoritmo de preprocesamiento de imágenes en color.

Para la detección de líneas azules debemos buscar la banda de interés, que la banda azul. Para esto es necesario descomponer la imagen RGB original en 3 imágenes simples que representan a cada banda.

Implementación en Matlab

```
1    imR=double(im(:,:,1));
2    imG=double(im(:,:,2));
3    imB=double(im(:,:,3));
```

Se realiza además una transformación al tipo de variable "double" para poder realizar operaciones aritméticas con las matrices, pues con imágenes uint8 no es posible. Un píxel azul se caracteriza por tener un valor elevado en la banda azul, y valores menores en las bandas verde y roja. De esta forma el azul ideal corresponde al valor RGB 0,0,255. Pero también es un azul intenso un valor RGB 23,14,195. En base a este razonamiento podemos afirmar que no será posible "aislar" cualquier tipo de azul analizando sólo la banda azul, para ello necesitamos imponer una condición lógica que considere las 3 bandas en sus conjunto.

El criterio utilizado es que la banda azul menos las bandas roja y verde debe ser a lo menos mayor a 20, un valor por debajo de 20 no se considerará como azul.

```
1    imR2=(imB-imR-imG);
```

Una vez establecido un umbral para decidir que punto RGB se considera como azul (en nuestro caso 20) procedemos a binarizar la imagen, la idea principal es que todos los puntos que cumplen dicha condición quedan con valor "1" y el resto con valor "0". La instrucción masc=(imR2>20) permite lograr una imagen binaria que representa a la "máscara" de todos los puntos azules que cumplen la condición. Una vez definida la máscara realizamos una multiplicación elemento a elemento de toda la matriz de la imagen con la máscara.

```
1    masc=(imR2>20);
2    imR2=imR2.*masc;
```

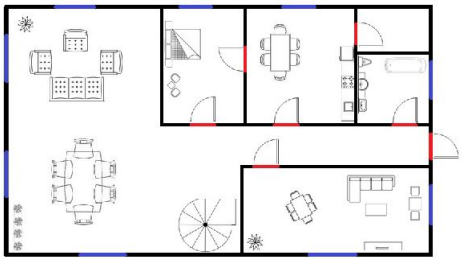


Figura 4.27: Plano original

Figura 4.28: Segmentación ventanas (color azul)

Una vez realizado deben detectar los bordes horizontales como verticales, para ello se utiliza la misma técnica que se ha utilizado en el apartado *4.1.2.2.1 Detección de bordes*.

Figura 4.29: Segmentación ventanas (color azul)

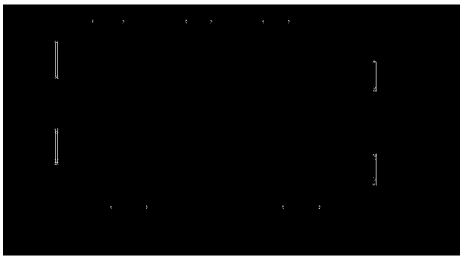


Figura 4.30: Bordes de ventanas verticales

Figura 4.31: Segmentación ventanas (color azul)

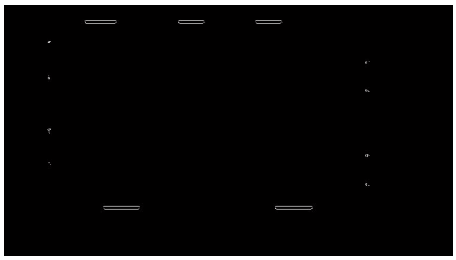


Figura 4.32: Bordes de ventanas horizontales

4.1.4.3 Extracción de información

Para la extracción de información se utiliza la misma técnica que en el apartado *4.1.2.3 Extracción de información*.
En las siguientes imágenes vemos como partiendo de un plano segmentado para la detección de ventanas se obtiene las coordenadas horizontales y se le asigna una etiqueta a cada una de ellas, en este caso una línea amarilla.

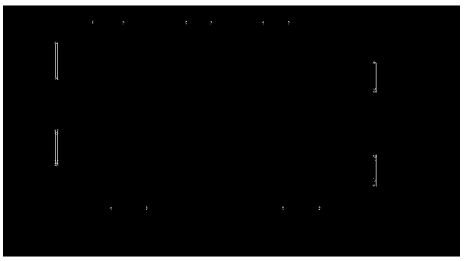


Figura 4.33: Bordes de ventanas verticales

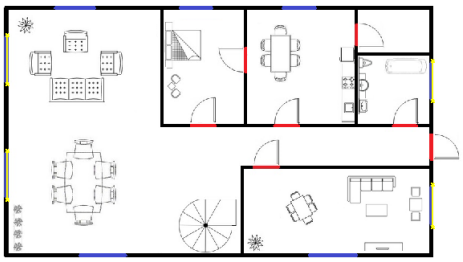


Figura 4.34: Detección de ventanas verticales (ventanas)

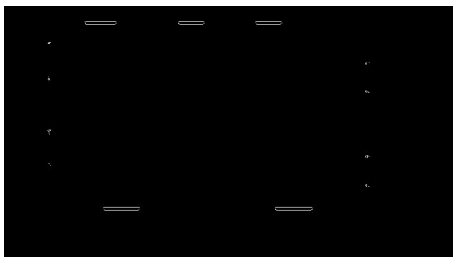


Figura 4.35: Bordes de ventanas horizontales

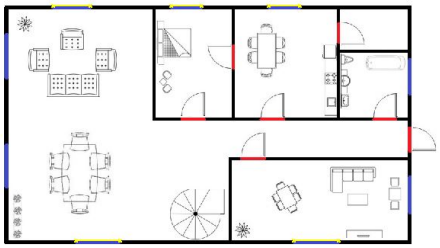


Figura 4.36: Detección de ventanas horizontales (ventanas)

4.1.5 Fichero de salida

El resultado de aplicar todas las técnicas anteriormente explicadas es el que aparece en la imagen *4.37*. En el que cada objeto tiene su etiqueta correspondiente, por ejemplo las lineas

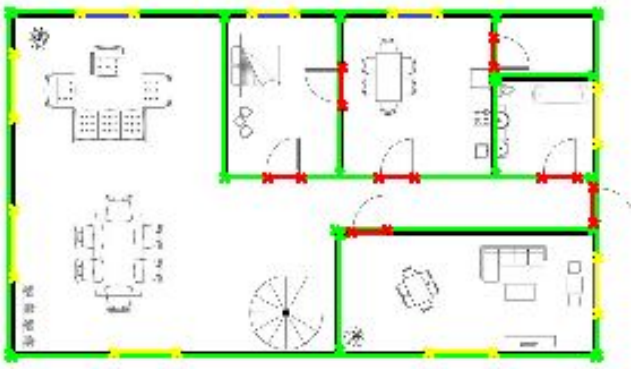


Figura 4.37: Plano totalmente tratado

verdes hacen referencia a las paredes, las rojas a las puertas y las amarillas a las ventanas.

Como ya hemos explicado en apartados anteriores, las coordenadas de las paredes, ventanas y puertas se guardan en un fichero que sirve como punto de unión entre la fase de detección

de dichos elementos con la fase de representación en 3d. Este fichero sigue una estructura determinada para que llegue a entenderse con el plugin de representación 3d creado también en este proyecto en el que explicaremos más adelante. La primera línea del fichero contiene el nombre de la puerta que se va a insertar, este nombre está pensado para futuras mejoras, ya que este proyecto no contempla elegir el tipo de puerta. La segunda línea contiene el nombre de la ventana que se va a insertar, este nombre también está pensado para futuras mejoras, ya que este proyecto tampoco contempla elegir el tipo de ventana. Justo después aparecen las coordenadas de las paredes verticales en el siguiente orden y estructura (x, y, x, y) por linea, las primera coordenada (x, y) hace referencia a un extremo de la pared y la segunda coordenada (x, y) hace referencia al otro extremo. Después de las coordenadas de las pareces verticales nos encontramos una barra media "-" que específica que las siguientes líneas hacen referencia a las paredes horizontales, estas coordenadas siguen el mismo orden y estructura que las paredes verticales. Una vez terminadas las coordenadas de las paredes horizontales la siguiente línea específica "Puertas" eso quiere decir que las siguientes coordenadas que nos encontramos son las de puertas verticales, el orden y estructura que sigue es el siguiente, coordenada (x, y) por linea, esa coordenada hace referencia al punto donde debe ir un extremo de la puerta, el otro extremo no se contempla ya que la puerta tiene una anchura especificada, al acabar las coordenadas de las puertas verticales nos encontramos una barra baja "_", que específica que las siguiente coordenadas hacen referencia a puertas horizontales, estas coordenadas siguen el mismo orden y estructura que las de puertas verticales. Por último nos encontramos la misma estructura que en las puertas pero para las ventanas, cambiando la barra baja que hace referencia a puertas horizontales por el signo igual "=" que hace referencia a ventanas horizontales.

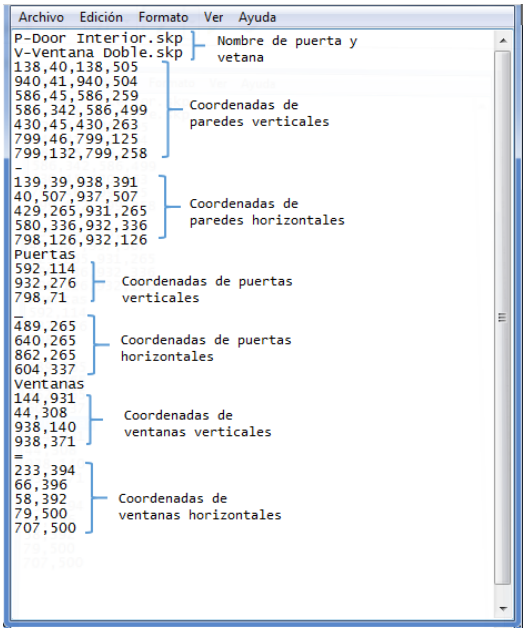


Figura 4.38: Estructura del fichero

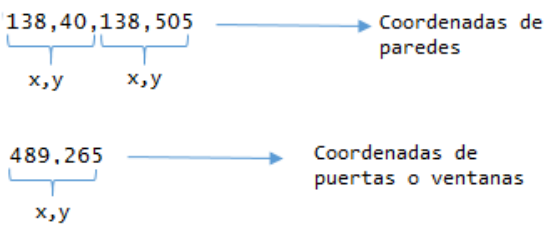


Figura 4.39: Estructura coordenadas

4.2 Representación 3D en google SketchUp con ruby (plugin)

[16] [21]

En esta sección se va a explicar los pasos seguidos para la creación del plugin capaz de leer el fichero previamente generado y representar en 3d mediante la aplicación google Sketchup cada uno de los objetos detectados basándose en las coordenadas escritas en dicho fichero. Cabe destacar que para el desarrollo de este plugin se ha utilizado la API de google Sketchup.

4.2.1 Representación de paredes

Para añadir una pared debemos declarar un array de esquinas y cada esquina será un vector con unas coordenadas concretas (x, y, z), dichas coordenadas harán referencia a la posición de una esquina concreta.

Por ejemplo: En la figura 4.40 disponemos de dos paredes diferentes (pared1 y pared2), lo que tendremos que representar cada pared por separado.

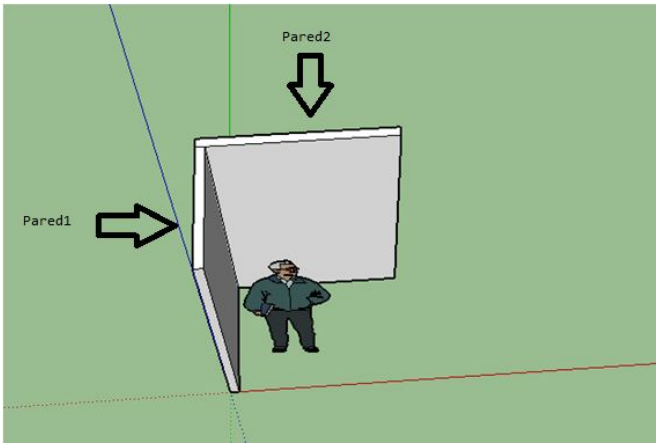


Figura 4.40: Representación paredes en google SketchUp

Para representar la pared 1 debemos fijarnos que tiene 4 esquinas lo que tendremos un array con 4 posiciones, y cada posición contendrá las coordenadas de dicha esquina. En la figura 4.41 lo vemos más claramente en un eje de coordenadas.

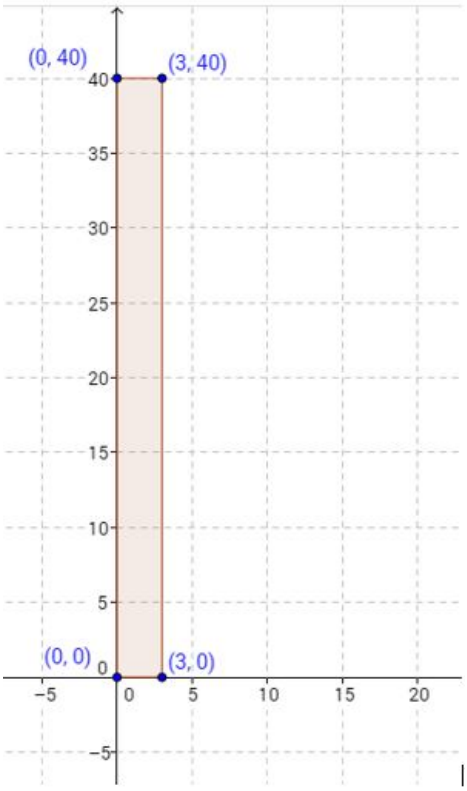


Figura 4.41: Representación pared en eje de coordenadas

Al leer el fichero que se ha generado anteriormente se obtendrán dos coordenadas de cada pared que hace referencia a una línea recta, para darle grosor a la pared se generará otras dos coordenadas sumándole un cierto número en función del grosor que se le quiera dar, la suma sería en la coordenada x para paredes verticales y en la coordenada y para paredes horizontales.

Implementación en ruby mediante la API de google SketchUp

Para la representación de dichas paredes debemos activar un modelo y dentro de ese modelo debemos activar una entidad.

```
1  #activar modelo
2  model = Sketchup.active_model
3  #activar entidad
4  entities = model.active_entities
```

Procedemos a insertar las coordenadas en el vector de esquinas, este vector después será utilizado para crear la pared basándose en las coordenadas que tiene insertadas. Dichas coordenadas habrán sido leídas anteriormente del fichero generado.

```
pts = []
pts[0] = [pared1x, pared1y, 0]
pts[1] = [pared1x+3,pared1y,0]
pts[2] =  [paredf1x+3,paredf1y,0]
pts[3] = [paredf1x,paredf1y,0]
```

Como vemos en cada array introducimos unas coordenadas (x, y, z), pero la coordenada z es 0 ya que con esa coordenada no vamos a trabajar. También en algunas coordenadas del eje x se le suma un 3, esa suma lo que quiere decir es que esa pared va a tener un grosor de 3 milímetros. Si nos fijamos más detenidamente vemos que se le suma al eje x lo que quiere decir que esa pared es vertical.
Por último añadimos una cara a la entidad con las coordenadas de la matriz anteriormente rellenada, esa cara hace referencia a una pared.

```
#aniadir pared
pared = entities.add_face pts
#aniadir altura
pared.pushpull altura
```

Una vez creada la pared le asignamos una altura.

4.2.2 Representación de puertas

Para la representación de puertas tenemos que averiguar qué posición y rotación respecto al eje x que tiene por defecto. Ya que según la posición y rotación de la puerta debe realizarse unas transformaciones u otras.

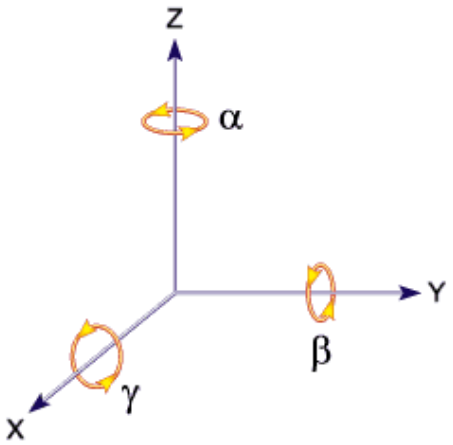


Figura 4.42: Angulos de rotacion sobre ejes

La puerta que se va a utilizar su estado por defecto como vemos en la figura 4.43 es tumbada respecto al eje x y en la posición 0 también respecto al eje x, esto significa que debemos realizar una serie de operaciones para que obtenga la posición y rotación que deseamos. Las operaciones a realizar si por ejemplo se ha leído en el fichero que se debe insertar una puerta vertical son las siguientes:

- Rotación de 90 grados sobre el eje x.
- Rotación de 90 grados sobre el eje z.

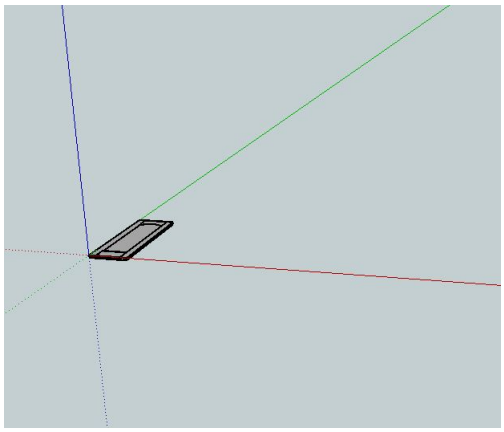


Figura 4.43: Puerta cargada por defecto

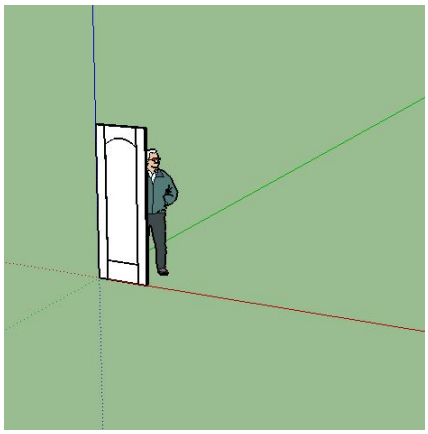


Figura 4.44: Puerta rotada respecto al eje x

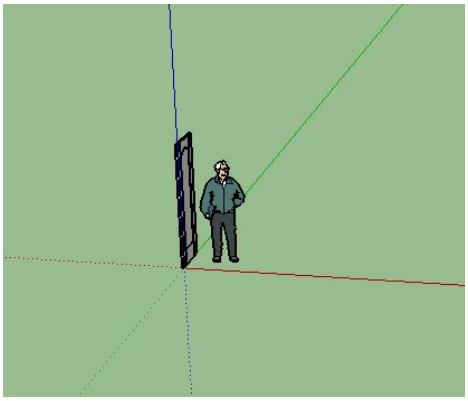


Figura 4.45: Puerta rotada respecto al eje z

Implementación en ruby mediante la API de google SketchUp

Para la representación de puertas en ruby primero realizamos las transformaciones como puede ser rotaciones y cambio de posición y después se añade la puerta.

En este proyecto para añadir puertas hemos utilizado funciones principales:

- **rotation:** Este método nos permite rotar una figura, en el segundo parámetro pasado se define sobre que eje queremos realizar la rotación, para ello creamos un nuevo objeto que al crear recibe 3 parámetros que hacen referencia a las 3 coordenadas (x, y, z) especificando con un 1 en el eje que queremos rotar y con un 0 en los que no. En el tercer parámetro que recibe esta función especificamos los grados de dicha rotación.
- **translation:** Este método nos permite trasladar una figura, recibe por parámetros las 3 coordenadas (x, y, z).
- **scaling:** Este método nos permite definir una escala de una figura.

Vamos a ver el ejemplo explicado anteriormente al completo implementado en ruby.

- Rotación de 90 grados sobre el eje x.

```
1 #Rotacion para poner la puerta hacia arriba 90 grados
2 t1 = Geom::Transformation.rotation(Geom::Point3d.new(0, 0,
    0), Geom::Vector3d.new(1, 0, 0), 90.degrees)
```

- Rotación de 90 grados sobre el eje z.

```
1 t3 = Geom::Transformation.rotation(Geom::Point3d.new(0, 0,
    0), Geom::Vector3d.new(0, 0, 1), 90.degrees)
```

- Establecer la puerta en unas coordenadas concretas.

```
1 t2=Geom::Transformation.translation([pared1x+3,pared1y,0])
```

- Establecer una escala

```
1 t = Geom::Transformation.scaling 1,2,1
```

Una vez declaradas todas las transformaciones debemos añadir la puerta.

```
1 #Activamos el modelo
2 model = Sketchup.active_model
3 Activamos la entidad
4 entities = model.active_entities
5 #Cargamos la puerta
6 path = Sketchup.find_support_file "Door Interior.skp","
    Components/Components Sampler/"
7 definitions = model.definitions
8 componentdefinition = definitions.load path
```

```
9  #Aniadimos la puerta y aplicamos todas las transformaciones
10 instance = entities.add_instance componentdefinition, t1
11 instance = instance.transform!(t)
12 instance = instance.transform!(t3)
13 instance = instance.transform!(t2)
```

Para añadir puertas horizontales la rotación sobre el eje z no se debe realizar.

4.2.3 Representación de ventanas

Para representar ventanas se sigue la misma metodología que para representar puertas, la única diferencia es respecto a la visualización de ventanas por ambas caras de la pared, ya que si se inserta una ventana solo se aprecia por una cara de la pared y no por las dos. Para solucionar este problema, insertamos dos ventanas por cada cara de la pared.
La ventana que se va a utilizar su estado por defecto como vemos en la figura 4.46, lo que para representar ventanas que su estado debe ser vertical respecto al eje x, no es conveniente realizar ninguna rotación, y para representar ventanas en horizontal respecto al eje x solo se le debería aplicar una rotación respecto al eje z. Las operaciones a realizar si por ejemplo se ha leído en el fichero que se debe insertar una ventana horizontal son las siguientes:

- Rotación de 90 grados respecto al eje z

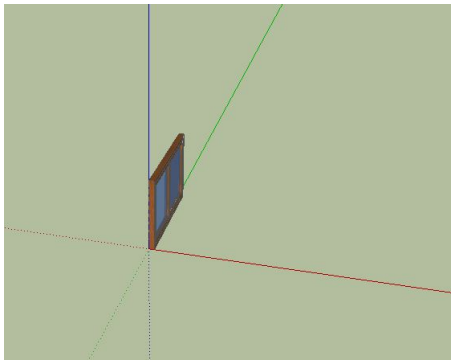


Figura 4.46: Ventana cargada por defecto

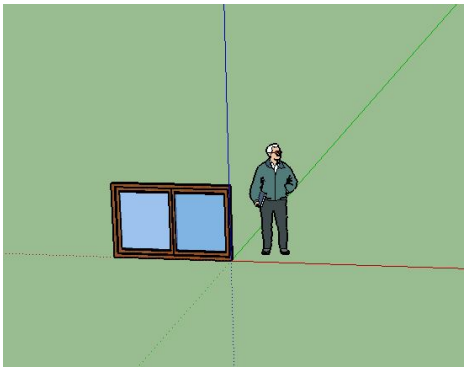


Figura 4.47: Ventana rotada respecto al eje z

Cabe destacar que en la representación de ventanas el eje z hace referencia a la altura que se va a encontrar la ventana de la superficie.

Implementación en ruby mediante la API de google SketchUp

Para la representación de ventanas en ruby primero realizamos las transformaciones como

puede ser rotaciones y cambio de posición y después se añade la ventana.
En este proyecto para añadir ventanas se han utilizado funciones principales:

- **rotation:** Este método nos permite rotar una figura, en el segundo parámetro pasado se define sobre que eje queremos realizar la rotación, para ello creamos un nuevo objeto que al crear recibe 3 parámetros que hacen referencia a las 3 coordenadas (x, y, z) especificando con un 1 en el eje que queremos rotar y con un 0 en los que no. En el tercer parámetro que recibe esta función especificamos los grados de dicha rotación.
- **translation:** Este método nos permite trasladar una figura, recibe por parámetros las 3 coordenadas (x, y, z).
- **scaling:** Este método nos permite definir una escala de una figura.

Vamos a ver el ejemplo explicado anteriormente al completo implementado en ruby.

- Rotación de 90 grados respecto al eje z.

```
1 #Rotacion para poner la puerta hacia arriba
2 t3 = Geom::Transformation.rotation(Geom::Point3d.new(0, 0,
      0), Geom::Vector3d.new(0, 0, 1), 90.degrees)
```

- Establecer la ventana en unas coordenadas concretas. La coordenada z que es igual a 30 hace referencia a la altura que estará la ventana sobre la superficie.

```
1 t4=Geom::Transformation.translation([pared1x,pared1y,30])
```

- Establecer una escala.

```
1 t = Geom::Transformation.scaling 0.8,0.8,0.8
```

Una vez declaradas todas las transformaciones debemos añadir la ventana.

```
1 #Activamos el modelo y entidad
2 model = Sketchup.active_model
3 entities = model.active_entities
4 # Cargamos la ventana
5 path = Sketchup.find_support_file "Ventana Doble.skp","
    Components/Components Sampler/"
6 definitions = model.definitions
7 componentdefinition = definitions.load path
8 # Aniadimos la ventana y aplicamos todas las transformaciones
9 instance = entities.add_instance componentdefinition, t
```

```
10 instance = instance.transform!(t3)
11 instance = instance.transform!(t4)
```

4.2.4 Integración plugin en Google SketchUp

Cuando SketchUp se inicia, se carga automáticamente cada .rb o .rbs, es decir todos los script o plugin que se encuentran en el directorio de plugins. La ubicación de la carpeta "Plugins" varía dependiendo de si se está ejecutando en Mac o Windows. Para la realización de este proyecto el plugin se guarda en la siguiente ruta:
C:\Users\<nombreUsuario>\AppData\Roaming\SketchUp\SketchUp 2015\SketchUp\Plugins

4.3 Entorno gráfico para la detección de paredes puertas y ventanas

[15]
Para que la experiencia de usuario con la aplicación sea lo mejor posible se ha diseñado un entorno gráfico que automatiza y unifica todas las fases de la detección y representación en 3d del plano.

Para ello se ha diseñado una interfaz lo más simple posible en la que al arrancar la aplicación aparece un botón de cargar plano como aparece en la figura 4.48, la función de este botón es seleccionar el plano a representar 3d.

Para implementar el botón con el fin de que obtenga la funcionalidad de elegir un plano en que está almacenado en nuestro equipo debemos introducir la siguiente función.

```
1 [nombre direccion]=uigetfile('*.*', 'Abrir');
```

Dicha función devuelve el nombre de la imagen y la dirección donde se encuentra. Una vez obtenido dicha información utilizamos la siguiente función para cargar el plano de la vivienda. Donde la variable *img* contendrá la matriz de del plano cargado.

```
1 img=imread(fullfile(direccion,nombre));
```

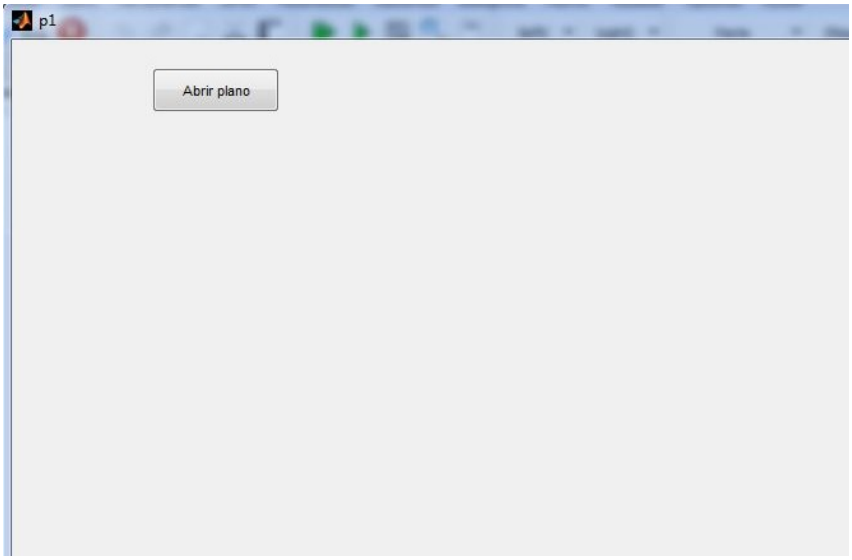


Figura 4.48: Captura cargar plano

Una vez cargado el plano, aparecerá dos botones más como vemos en la figura 4.49, el primer botón nos permiten detectar paredes, puertas y ventanas y el segundo botón nos permite representar el plano en 3D mediante la aplicación google SketchUp, cabe destacar que primero se debe detectar los objetos y por último representar en 3D, ya que si no se ha detectado ningún objeto tampoco se verá reflejado en 3D.

Para implementar dichos botones unificamos todos los métodos explicados en el capítulo *4.1.Detección de objetos en un plano 2D con Matlab* en la función asociada a cada botón.

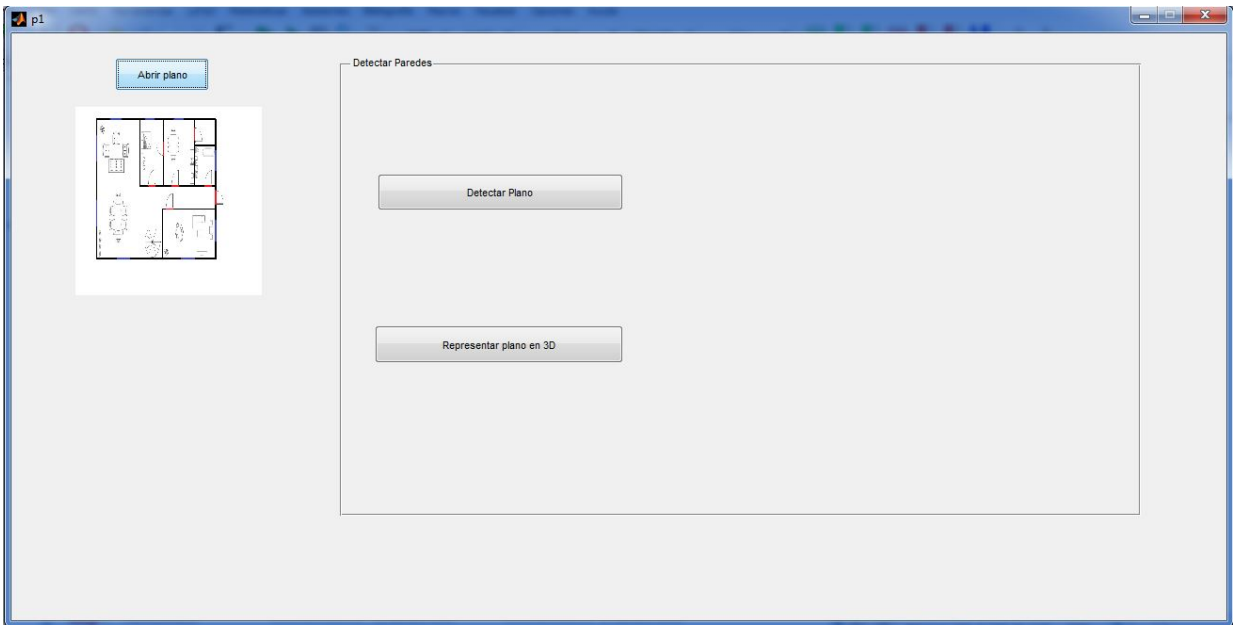


Figura 4.49: Captura detectar y representar en 3D

La gran diferencia en la implementación de los dos botones es que el botón "Detectar Plano" muestra en una ventana el plano con sus diferentes etiquetas y el botón "Representar plano en 3D" guarda las coordenadas en un fichero, cabe destacar que los dos botones realizan las mismas operaciones para la detección.
Una vez que se ha detectado el plano, cada objeto aparecerá con una etiqueta, en este caso una línea de un determinado color, las paredes aparecerán con una línea de color verde, las puertas de color rojo y las ventanas de color amarillo, como podemos ver en la figura 4.50. Una vez detectados correctamente los objetos representamos el plano en 3 dimensiones pulsando el botón "Representar plano 3d".

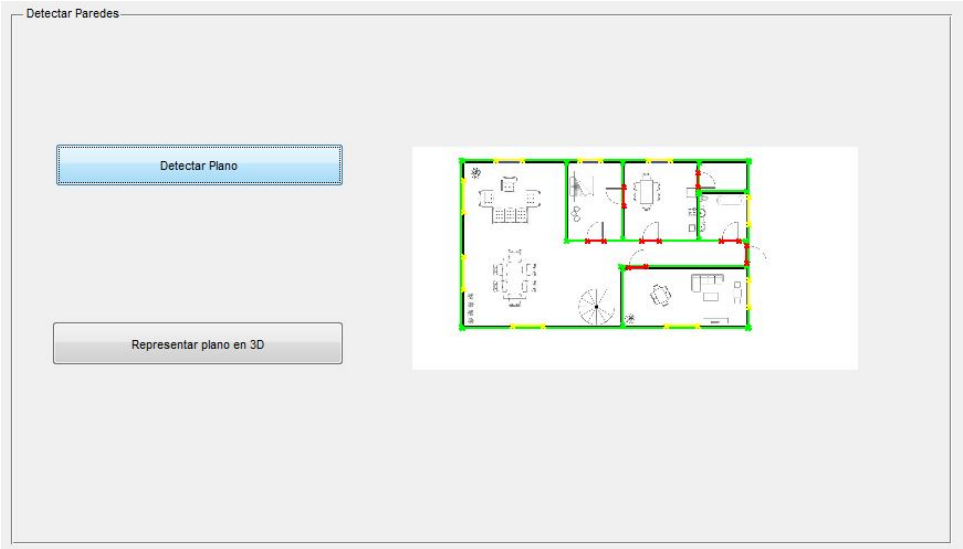


Figura 4.50: Captura plano detectado correctamente

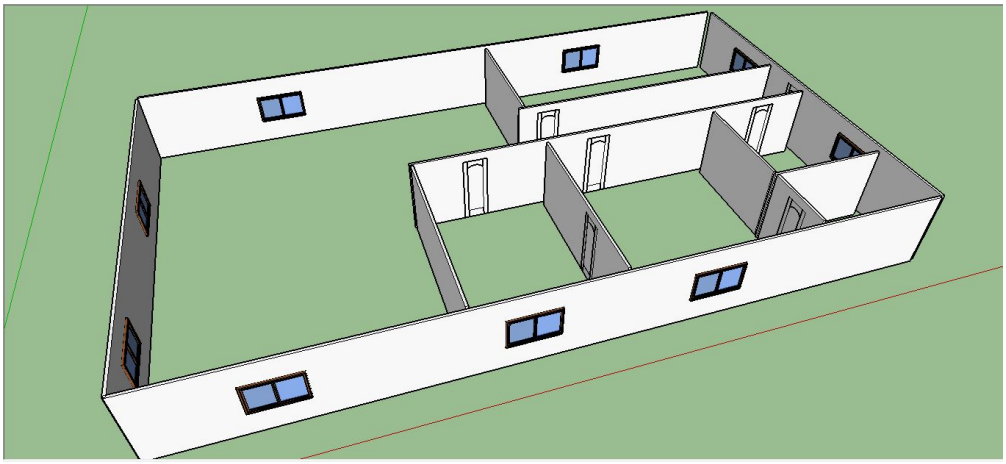


Figura 4.51: Captura plano representado en 3D

5

Simulaciones y resultados

Para realizar las simulaciones y obtener resultados se ha representado en 3D varios planos 2D con diferentes características, con el fin de medir el tiempo desde que se detecta el plano hasta que se representa en 3D. En la tabla 5.1 se puede ver la comparativa de todas las pruebas realizadas. Dicha tabla contiene los siguientes atributos:

- **ID Plano:** (Identificador de plano): Nombre dado al plano.
- **Paredes:** Número de paredes tanto verticales como horizontales.
- **Puertas verticales:** Número de puertas verticales que contiene el plano.
- **Puertas horizontales:** Número de puertas horizontales que contiene el plano.
- **Ventanas verticales:** Número de ventanas verticales que contiene el plano.
- **Ventanas horizontales:** Número de ventanas horizontales que contiene el plano.
- **Tiempo (segundos):** Tiempo que tarda la aplicación desde que se detecta el plano hasta que se representa en 3D.
- **Fallo:** Fallo al detectar o representar el plano.

ID Plano	Paredes	Puertas verticales	Puertas horizontales	Ventanas verticales	Ventanas horizontales	Tiempo (segnudos)	Fallo
1	4	0	0	0	0	-	Si
2	4	1	1	1	1	16,88	No
3	6	1	1	1	1	14,9	No
4	12	0	1	0	0	-	Si
5	12	2	2	2	2	16,21	No
6	17	4	4	4	5	16,35	No
7	20	3	7	0	2	-	Si

Table 5.1: Tabla comparativa de las pruebas realizadas

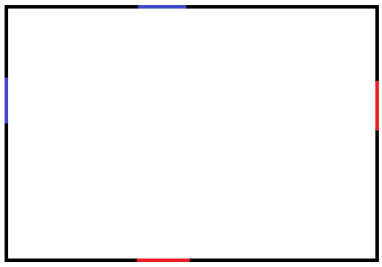
5.0.1 Imágenes estudiadas

Imagen 1:

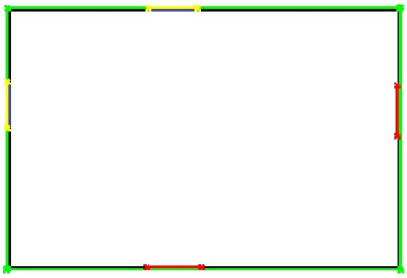
Al procesar esta imagen la aplicación falla, ya que no dispone de puertas ni ventanas.

1-Imagen original

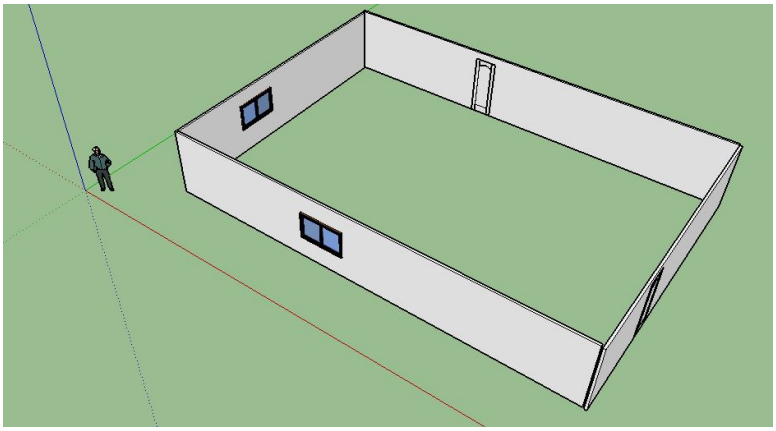
Imagen 2:



2- Imagen original



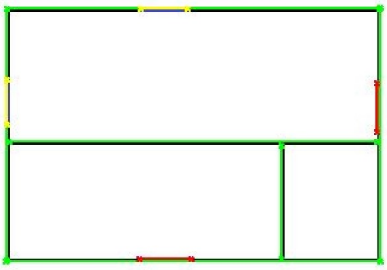
2- Imagen con objetos detectados y etiquetados



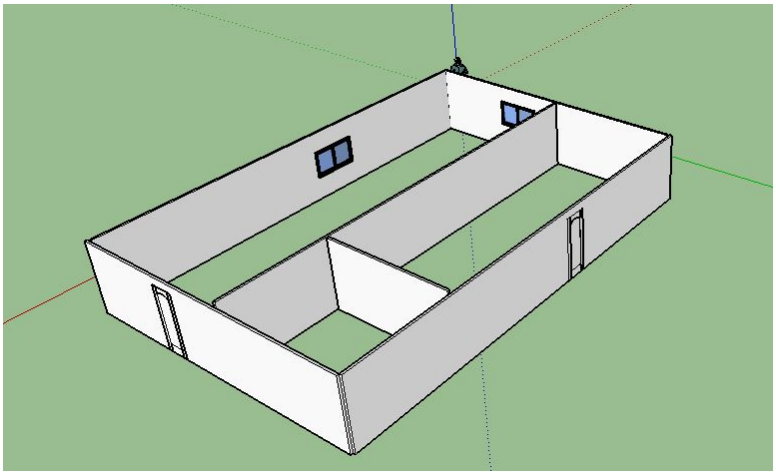
2- Imagen representada en 3d

Imagen 3:

3- Imagen original

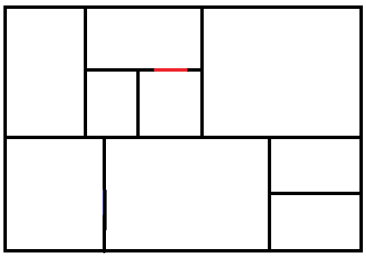


3- Imagen con objetos detectados y etiquetados



3- Imagen representada en 3d

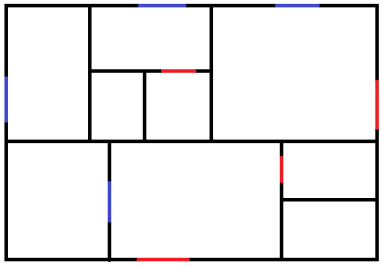
Imagen 4:



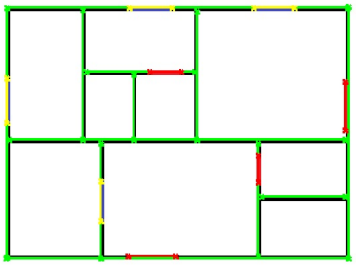
4-Imagen original

Al procesar esta imagen la aplicación falla, ya que no dispone de puertas verticales ni ventanas verticales y horizontales.

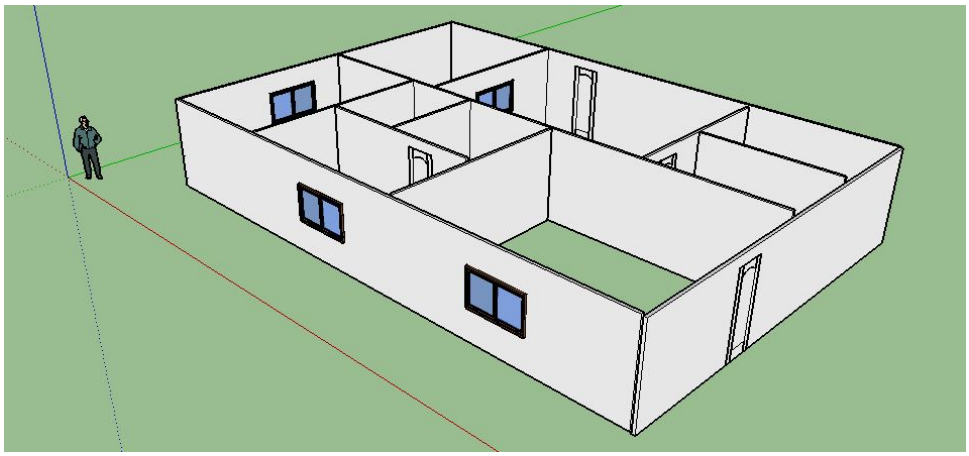
Imagen 5:



5- Imagen original

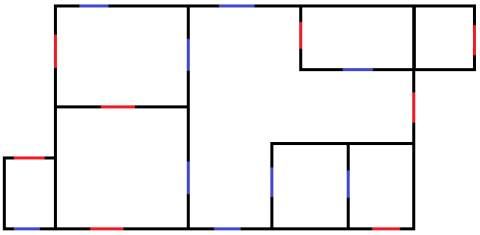


5- Imagen con objetos detectados y etiquetados

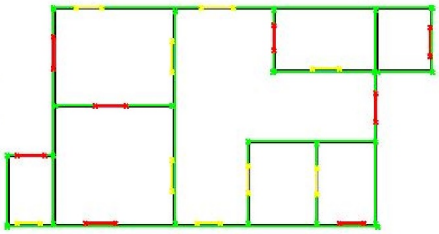


5- Imagen representada en 3d

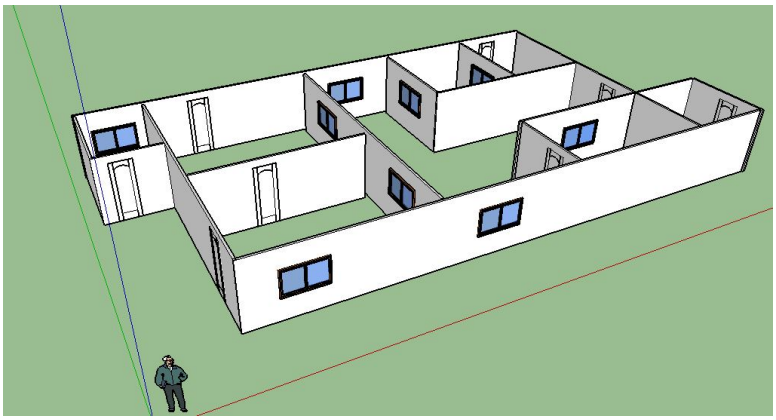
Imagen 6:



6- Imagen original

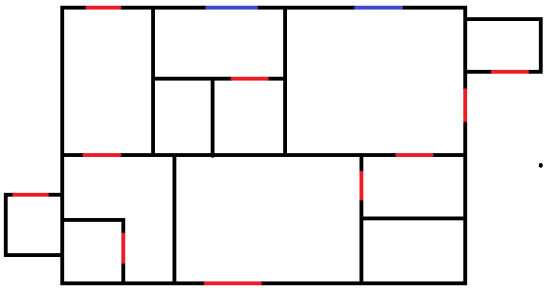


6- Imagen con objetos detectados y etiquetados



6- Imagen representada en 3d

Imagen 7:



7-Imagen original

Al procesar esta imagen la aplicación falla, ya que no dispone de ventanas verticales.

Al estudiar los resultados representados en la tabla 5.1 se obtiene como conclusión que el tiempo de cómputo de la aplicación es muy igualado, eso quiere decir que no depende del número de paredes, puertas o ventanas de las que dispone el plano, también vemos que la aplicación falla si el plano no dispone de puertas verticales u horizontales al igual que si no dispone de ventanas verticales u horizontales.

6

Conclusiones y Trabajos futuros

En este proyecto se han conseguido los objetivos propuestos en un principio, de detectar un plano 2D y representarlo automáticamente en 3D.
Como hemos visto en el desarrollo de este TFG, el procesamiento y la interpretación correcta de los objetos contenidos en una imagen es una tarea compleja y computacionalmente cara. Cualquier reconocimiento de objetos requiere una capacidad de cómputo muy elevada debido a que las imágenes a procesar son matrices tridimensionales grandes. A lo largo de este documento se han ido aplicando distintas operaciones que se han realizado de forma específica para el plano sobre el que se ha trabajado en este proyecto. Es por ello que se han usado en varias funciones parámetros específicos que no siempre serán válidos cuando se utilice planos con características menos comunes. Una de las tareas más complicadas fue diferenciar las líneas que pertenecían a paredes con las líneas que pertenecían a objetos del plano como pueden ser sofás, armarios, duchas etc...
Como conclusión se puede definir la realización de un programa con un balance de error admisible al igual que el tiempo de cómputo depende de las características del equipo donde se ejecute.
Este proyecto tiene muchos trabajos futuros ya que el objetivo era crear una base sobre la que empezar a investigar.

Ya que el plano debe cumplir unas ciertas características como puede ser la diferenciación de puertas, ventanas y paredes por colores, se propone una mejora referente a la detección de dichos objetivos mediante su símbolo estándar en un plano 2D, por ejemplo: Para detectar una puerta, se deberá utilizar como referencia el símbolo de la figura 6.1.

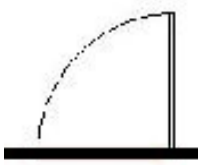


Figura 6.1: Símbolo estándar de una puerta.

Otra mejora debería ser el dar la posibilidad al usuario de introducir planos en 2D que no contengan puertas, ventanas o paredes horizontales o verticales, ya que lo desarrollado hasta el momento obliga al usuario a que un plano contenga tantas puertas ventanas y paredes verticales como horizontales. También otra de las mejoras podría ser el permitir al usuario detectar planos que contengan paredes curvas ya que el proyecto desarrollado solo permite paredes totalmente rectas, como también poder detectar los símbolos estándar del mobiliario de la casa y ser capaz de representarlos en 3D.
Por último cabe destacar que el plugin está capacitado para poder elegir diferentes tipos de puertas y ventanas, pero no se ha llegado a dar esa opción al usuario debido a que Google SketchUp tiene un repositorio donde los usuarios pueden subir diferentes tipos de puertas o ventanas y cada una de ellas aparece con una posición y orientación diferente lo que dificulta la representación en 3D ya que primero se debería averiguar la orientación y posición de cada objeto y trabajar referente a eso por lo que debido a la falta de tiempo no se ha llegado a implementar.

BIBLIOGRAFÍA

[1] RAFAEL C. GONZÁLES y RICHARD E. WOODS, *Tratamiento digital de imágenes*, segunda edición, Limusa, Mèxico, DF, 2004.

[2] RUBÉN WAINSCHENKER, JOSÉ MARÍA MASSA y PAULA TRISTAN *Procesamiento Digital de Imágenes*

[3] Introducción tratamiento digital de imágenes: `http://www.secyt.frba.utn.edu.ar/gia/introd-al-proc-de-imagenes.PDF`

[4] STEVE ROSSIUS, *Reconocimiento de objetos mediante WebCam en tiempo real*, Proyecto final de carrera. Universidad Politécnica de Valencia.

[5] API Google Sketchup: `http://www.sketchup.com/intl/en/developer/`

[6] VICTOR MANUEL GARCÍA LUNA, *Introducción al Procesamiento Digital de Imágenes*, Universidad tecnológica de la Mixteca, Oaxaca, Mexico.

[7] PABLO RONCAGLIOLO B., *Ejemplo práctico de procesamiento de imágenes en color: efecto publicitario "rojo sobre grises"*

[8] JUAN LUIS BARREDA SÁNCHEZ, *Seguimiento visual de líneas de cultivo*, Proyecto final de carrera. Universidad de Murcia

[9] ÁREA DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA, *Transformada de Hough*, Departamento de Ingeniería electrónica, Telecomunicación y Automática. Universidad de Jaén

[10] JAVIER HERNÁNDEZ ALFONSO, *Sistema de detección y almacenamiento de variables visualizadas por un dispositivo mediante pantalla o display a través de un sistema de visión de bajo costo*, Trabajo fin de grado, Universidad de la Laguna.

[11] JORGE ELOY-GARCÍA VARGAS-MACHUCA, *Recuperación de Imágenes Usando Atributos Difusos*, Escuela técnica superior de Ingeniería Informática de la Universidad de Málaga

[12] NORA LA SERNA PALOMINO y ULISES ROMÁN CONCHA *Técnicas de Segmentación en Procesamiento Digital de Imágenes*, Facultad de Ingeniería de Sistemas e Informática, Universidad Nacional Mayor de San Marcos

[13] *Procesamiento Global empleando la Transformada de Hough*, Cátedra Visión Artificial, Universidad Nacional de Quilmes – Ing. en Automatización y Control Industrial

[14] RUBÉN LLOBREGAT RUBIO, *Procesado digital de imágenes de vídeo para la detección de humo*, Proyecto final de carrera. Universidad politécnica de Valencia, Escuela politécnica superior de Gandia.

[15] SANTIAGO GALÁN MORALES, *Interfaz Gráfica Para La Simulación De Modelos Dinámicos*, Proyecto final de carrera. Universidad Carlos III de Madrid, Escuela politécnica superior.

[16] Tutorial ruby: `http://rubytutorial.wikidot.com/`

[17] Tutorial Matlab: `https://www.tutellus.com/seminarios/ingenierias/aprende-a-programar-con-matlab-3185`

[18] Tutorial Matlab: `https://procesamientodigitalimagenes.wordpress.com/2012/11/02/transformada-hough/`

[19] Tratamiento digital de imágenes: `https://es.wikipedia.org/wiki/Procesamiento_digital_de_imágenes`

[20] Procesamiento de imágenes con Matlab: `http://matlabimagenes.blogspot.com.es/`

[21] Representación en 3 dimensiones: `http://graficacion-suirot18.blogspot.com.es/2013/10/31-representacion-de-objetos-en-tres.html`

Lista Acrónimos

API: Application Programming Interface
3d: 3 Dimensiones
2d: 2 Dimensiones
UAL: Universidad de Almería
RAM: Random-Access Memory
GB: Gigabyte
IDE: Entorno de Desarrollo Integrado
SIG: Sistema de Información Geográfica
BASIC: Beginner's All-purpose Symbolic Instruction Code
PDI: Procesamiento Digital de Imágenes
RGB: Red, Green, Blue
TFG: Trabajo Fin de Grado

UNIVERSIDAD DE ALMERÍA

Resumen/Abstract

El objetivo de este proyecto es crear una aplicación integrada con google sketchup capaz de interpretar un plano en 2D con una serie de condiciones y representarlo en 3 dimensiones mediante google sketchup.

La aplicación será implementada utilizando dos lenguajes de programación diferentes, uno de ellos es Matlab que se encarga de la extracción de información del plano 2 dimensiones y el otro de ellos es ruby, que se utilizará para representar en 3 dimensiones dicho plano.

Para la realización de este proyecto se siguen diferentes etapas, la primera es la extracción de información del plano 2 dimensiones, dicha información se guardará en un fichero y la segunda etapa se encarga de interpretar la información de dicho fichero y representar el plano en 3 dimensiones. Para la representación en 3 dimensiones se ha utilizado la API de google sketchup.

La finalidad de esta aplicación es facilitar a los usuarios inexpertos la creación en 3 dimensiones de sus viviendas sin necesidad de dedicar tiempo a su autoformación en este ámbito.