Sistemas Informáticos
Curso 2004-05

Reconocimiento Visual de Instrucciones de un entorno 3D para un Robot Autónomo

Carlos León Aznar
Jorge Mendoza Castejón
Diego Sánchez Nieto

Dirigido por:
Prof. Gonzalo Pajares Martinsanz
Dpto. Sistemas Informáticos y Programación

Facultad de Informática
Universidad Complutense de Madrid

Índice general

Resumen del proyecto

Resumen

El proyecto consiste en el desarrollo de un motor de visión por computador modular, que tiene como objetivo el reconocimiento de gestos de las manos y de órdenes escritas, adquiridas a través de interfaz de imágenes como una cámara y otros, y la traducción de las mismas a órdenes inteligibles. Asimismo, se crearán un microrrobot y un entorno de visualización en tres dimensiones, que muestren, mediante el movimiento del mismo y el desplazamiento del personaje en el entorno, respectivamente, la ejecución de dichas órdenes.

Abstract

The project consists on the development of a modular machine vision engine, that has as objective the reconoissance of hand gestures and written orders, adquired through an image interface like a camera and others, and the translation of these to understandable orders. Also, a micro-robot and a three dimension environment will be created that show, by the movement and the displacement of the character in the environment, respectively, the execution of the mentioned orders.

Palabras clave

Visión, reconocimiento, OCR, neuronal, entorno, 3D, robot, pipeline, módulo, imagen.

Autorización

Autorizo a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria, como el código, la documentación y/o el prototipo desarrollado.

Firmado:

Parte I

Definición del proyecto

Capítulo 1

Introducción

1.1. Introducción

Esta sección ofrece una primera toma de contacto con el proyecto, explicando de la manera más simple y concisa posible los primeros detalles generales que hemos considerado de interés para el conocimiento de "Reconocimiento visual de instrucciones".

1.1.1. Objetivos

El objetivo del proyecto consiste desarrollar en un motor modular de visión no dependiente del entorno y el reconocimiento de mensajes, a través de las imágenes capturadas desde una cámara, para la implantación del sistema de órdenes de un robot. Para la demostración de la funcionalidad se usarán un entorno en 3D y un robot construido desde cero.

La idea fundamental ha sido partir desde cero y emplear una gran parte de los conocimientos adquiridos durante la carrera para desarrollar un motor de reconocimiento de visión, con unos resultados útiles, diferenciables, y, en la medida de lo posible, vistosos. Mediante la implementación, además, hemos procurado desarrollar un sistema eficiente y sólido, a la vez que dotado de gran generalidad, para que el flujo de producción de la aplicación y las posteriores modificaciones fuesen lo más cómodas y sencillas posibles.

1.1.2. Organización de la memoria

La memoria está organizada en tres partes. La primera, en la que nos encontramos (parte I), se dedica a la explicación del proyecto como tal, con el propósito de proporcionar una visión general y directa de *Reconocimiento visual de instrucciones*. Aquí explicamos la arquitectura de "tubería" de la

aplicación, y el posible uso futuro del trabajo realizado, entre otras cosas. Esta primera parte es la base para adquirir una visión general y completa del trabajo, necesaria tanto para conocerlo como para usarlo o ampliarlo.

Como segunda parte de la memoria hemos añadido un primer anexo (II) en el que detallamos los módulos que usamos, yendo un nivel más abajo en el detalle de explicación de los mismos. Aquí damos ya una visión menos general, pero más profunda, de las diferentes tecnologías características que hemos usado para desarrollar las diferentes partes de la aplicación. Hemos intentado que este módulo de la memoria sirva como base para el uso metódico del proyecto, y que ofrezca al lector una primera aproximación a la verdadera arquitectura e implementación del trabajo. Por tanto, es de interés para aquellos que vayan a usar el proyecto de una manera más avanzada que la de la simple prueba.

Además, adjuntamos en el CD de la aplicación toda la documentación del API de la aplicación, como HTML, para posibles consultas a la implementación del proyecto, o para ampliaciones del mismo.

1.2. Especificación

Vamos a exponer a continuación los requisitos y características que definen el proyecto de Visión por Computador que hemos desarrollado. Esta enumeración ha sido la base a partir de la cuál se ha desarrollado el diseño y la posterior implementación de cada uno de los módulos que constituyen el proyecto. Por tanto describe las funcionalidades que establecimos inicialmente como objetivos que se debían satisfacer una vez concluido el proyecto. Posteriormente será necesario realizar una serie de pruebas que confirmen que cada uno de las siguientes características han sido implementadas adecuadamente para corroborar el éxito del proyecto. Estás pruebas se detallarán de forma exhaustiva en el apartado correspondiente.

1.2.1. Interacción con una cámara de video

Se deberá poder inicializar, configurar y utilizar una cámara que se comunique con nuestro proyecto suministrando de forma adecuada las imágenes que deberán ser analizadas.

1.2.2. Adquisición y tratamiento de las imágenes

Los datos suministrados por la cámara deberán ser modificados convenientemente para ajustarse a un formato útil para su manipulación por parte

de nuestro proyecto. Además, será necesario que la imagen recibida a través de la cámara sea tratada mediante una serie de filtros que resaltarán y corregirán adecuadamente los elementos necesarios para su posterior análisis.

1.2.3. Tolerancia a diferentes condiciones de iluminación

Debido a que se trata con elementos visuales (imágenes de una cámara de video), es necesario que el sistema sea tolerante a las diferentes condiciones de iluminación que se pueden dar en distintos entornos. Se debe proporcionar una solución a este respecto que permita la pertinente graduación de los procesos de filtrado y corrección de imágenes para poder adaptarse a unas condiciones ambientales variables (obviamente siempre dentro de unos márgenes mínimos de iluminación).

1.2.4. Análisis de las imágenes

Una vez tratadas adecuadamente, las imágenes deberán ser analizadas para poder determinar si corresponden a un item reconocido por el sistema. El sistema deberá reconocer dos tipos de items: gestos realizados por el usuario que se interpretarán como comandos de navegación del robot, y textos escritos en carteles. Los comandos se generarán a partir de los gestos que el usuario realice con sus dos manos. En cada una de ellas deberá portar unos guantes con unos puntos de color diferenciados para cada mano. Los gestos de una de las manos se interpretarán como órdenes de navegación del tipo avanzar, girar o parar mientras que los gestos de la segunda mano se interpretarán como el grado de intensidad con la que la orden actual debe ser ejecutada. Así se podrá ordenar que el robot gire a la izquierda con mayor o menor velocidad de rotación por ejemplo. En cuanto al texto, deberá estar escrito sobre carteles de un color que destaque en el entorno. El texto podrá consistir en operaciones aritméticas sencillas, órdenes de control para la navegación del robot, o sencillas preguntas sobre datos que el sistema posea en su base de datos. Una vez analizado, el sistema dará respuesta según la naturaleza del texto de entrada. En caso de tratarse de una operación aritmética, se deberá dar la solución del cálculo. Si se trata de una orden de navegación, se deberá considerar como un comando de movimiento para el robot que deberá comportarse en consecuencia. Y por último, si se trata de una pregunta, se deberá dar una respuesta coherente a la semántica y formulación sintáctica de ésta.

1.2.5. Simulación del comportamiento del robot según los comandos procesados

Para poder comprobar el correcto funcionamiento del sistema de reconocimiento de gestos, se deberá utilizar un sistema que visualice de forma sencilla e intuitiva una simulación del comportamiento que debería presentar el robot en respuesta a las órdenes recibidas. Esta funcionalidad será útil tanto para realizar pruebas como para contar con una salida auxiliar que corrobore el comportamiento de un robot físico que se conecte al sistema. De esta manera se podrá ir verificando que cada orden procesada es interpretada de forma similar por el robot físico en concordancia con lo visualizado en la simulación.

1.2.6. Creación y conexión al sistema de reconocimiento de un robot

Se deberá construir un robot mecánico con capacidades motrices básicas (avanzar, retroceder, y girar en ambos sentidos) que se conectará al sistema de reconocimiento de gestos y actuará en consecuencia.

1.3. Estado del arte

1.3.1. Punto de partida

Se trata de un proyecto que surge por mutuo acuerdo entre el profesor supervisor y el grupo de alumnos en el ámbito de las normas que regulan la asignatura de Sistemas Informáticos.La idea comienza a fraguarse por la inquietud que nos suscita el mundo de la robótica alimentada por la gran cantidad de noticias en los medios de comunicación sobre robóticas, así como la difusión de eventos sobre robótica tales como el Hispabot. Madurada la idea , el procesa comienza con las reuniones pertinentes con el profesor del sector de robótica, que nos introduce en la materia, proporcionándonos principalmente la documentación y referencias pertinentes. Existe en la literatua una gran cantidad de trabajos relacionados, si bien dado que el objetivo principal del proyecto no es la investigación, nos centramos en las referencias básicas que aparecen en la sección bibliografía.

1.3.2. Conocimientos previos

En el desarrollo del proyecto ha sido necesario el uso e integración de conocimientos adquiridos previamente en asignaturas impartidas a lo largo

de la carrera de Ingeniera Informática Superior. De esta forma el proyecto se ha fundamentado en conocimientos previos de asignaturas como Inteligencia Artificial, Robótica, Informática Gráfica, Ingenieria del Software, Aprendizaje Automático, Programación Lógica, Procesadores de Lenguaje, y los distintos laboratorios de Tecnología de Computadores.

1.3.3. Visión general de los campos de estudio relacionados con el proyecto

Vamos a exponer brevemente una visión general de los principales campos de estudio que han constituido la base para el desarrollo de este proyecto. Se expondrá el contexto histórico y evolución de cada materia destacando los hitos más significativos así como una previsión de las futuras tendencias.

Entornos 3D

Introducción: En los últimos tiempos se ha producido un avance increíble en el desarrollo de aplicaciones que hacen uso de la tecnología gráfica 3D para representar entornos interactivos que puedan ser explorados por el usuario. En el año 1995 se produjo un hito en la tecnología de las tarjetas gráficas al aparecer por primera vez en el mercado placas capaces de realizar cálculos 3D de forma optimizada. Marcas como 3Dfx con su chip Voodoo fueron las pioneras en este sector. Gracias a este avance, se produjo una revolución en el campo de los videojuegos así como en otros sectores que hacían uso de los gráficos tridimensionales. A partir de ese momento, la escalada en la evolución de los chips gráficos fue constantes, habiendo alcanzado hoy en día una situación en la que la tecnología (nº de transistores, velocidad de memoria, ancho de banda, etc) de los chips gráficos está a la par e incluso supera a la de los chips principales de un computador(CPU). Sin embargo, esta incremento en la capacidad de cálculo no hubiera sido suficiente para alcanzar los niveles actuales de realismo en las aplicaciones 3D si no hubiera sido por el desarrollo en paralelo de nuevas soluciones algorítmicas que aprovecharan este nuevo potencial emergente. De esta manera se han realizado innumerables avances en las técnicas de computación gráfica en campos tan dispares como la representación de terrenos virtuales o las técnicas de animación de modelos. Veamos a continuación una breve enumeración de estas nuevas técnicas.

Técnicas principales: Terrenos virtuales: El objetivo de todo algoritmo para representar terrenos es la minimización de la cantidad de triángulos(entidad básica en el procesamiento de geometría en 3d) que se deben procesar para mostrar dicho terreno con una calidad y fluidez adecuada. Con

este precepto han surgido algoritmos como el CLOD, ROAM, Geometrical MipMapping, VDPM, etc. También es común el uso de técnicas de partición del espacio geométrico tales como el uso de Octrees, Quadtress, BSP y KD-Trees que permiten optimizar el cómputo global de los terrenos así como acelerar los cálculos de colisiones, etc.

Cielo y Atmósfera: La representación de la bóveda celeste y de sus elementos (nubes, sol, estrellas, etc) suele realizarse en la mayor parte de los casos utilizando skybox (cubo que engloba el escenario en su interior y que posee una serie de imágenes en cada una de sus caras que representan el cielo),o sky-domes (esferoides truncados sobre los que se texturiza como en el caso anterior una imagen del cielo). Estas técnicas se suelen complementar con el uso de capas de texturas animadas para dar dinamismo a las nubes, luces glow para representar el sol o estrellas, e incluso sistemas de partículas avanzados para conseguir resultados más sofisticados como nubes volumétricas y otros efectos atmosféricos como lluvia, niebla, nieve, etc.

Animaciones: Los sistemas de animación de entidades 3D ha evolucionado bastante en los últimos años. Antaño las entidades se animaban mediante una técnica denominada keyframe. Básicamente consistía en grabar a lo largo del tiempo una serie de posiciones (keys) del elemento 3D de forma que posteriormente se interpolaban para generar una animación. Posteriormente apareció el uso de la animación por huesos o esqueleto. En este caso, a la entidad 3d poseía un esqueleto al cúal iba asociada la piel del modelo. Esta técnica incrementó sustancialmente el realismo de las animaciones ya que ahora el movimiento producía deformaciones coherentes con la morfología de la entidad 3D. Por último, se complementó la animación por esqueleto con otras soluciones como el uso de cinemática inversa o aplicación de modelos físicos(ragdoll) para realizar animaciones que se adaptarán al entorno virtual en cada situación.

Sombreado: Aunque la iluminación de modelos tridimensionales es algo solventado prácticamente desde el nacimiento de la infografía, la capacidad de calcular las sombras producidas por un objeto ha sido un problema arduo que aún hoy en día es campo de estudio e innovación. A lo largo de los último años se han generalizado dos técnicas principales. La primera se denomina Shadow Mapping y consiste en generar en tiempo real una imagen que contiene las sombras de una escena, para posteriormente proyectar dicha imagen sobre los elementos que son ocluidos por otros respecto a la fuente de luz. La otra técnica se denomina Stencil Shadow y hace uso del denominado Stencil Buffer(característica incorporada en las tarjetas gráficas de nueva generación) para calcular las sombras mediante la extrusión de la geometría del objeto que proyecta las sombras. Este último algoritmo fue perfeccionado para solucionar ciertas limitaciones mediante una aproximación ideada por

John Carmack(presidente de IDSoftware) denominada Z-fail Stencil Shadow.

Sirva la breve descripción anterior como ejemplo de la enorme cantidad de innovaciones que se han producido en el campo de la computación gráfica en los últimos tiempos. Nos encontramos inmersos en una etapa de expansión constante que la que cada paso supera con creces los éxitos logrados hasta la fecha. Es por tanto necesario preguntarnos acerca de las futuras tendencias del mundo de la computación 3D.

Tendencias futuras: Desde hace poco tiempo se han instaurado en el mercado nuevas tarjetas graficas con Píxel Shaders. Esta nueva tecnología permite el proceso de las imágenes a nivel de píxel(unidad básica que compone cualquier imagen digital). De esta forma, se está abriendo un nuevo abanico de técnicas para elaborar efectos especiales e incrementar de forma asombrosa el foto realismo de las imágenes tridimensionales. Efectos como iluminación por píxel, normal mapping, parallax mapping, HDR(High Dymic Range Iluminance), Bloom, Blur, Fressnell Refraction & Reflexion, y un largo etcétera que empiezan a aparecer en las nuevas aplicaciones y que a buen seguro colmarán todas las que aparezcan en el futuro.

Redes neuronales

Introducción: Desde la aparición de las computadoras digitales en los años 40 uno de las investigaciones recurrentes ha sido la implementación de sistemas que intentaran simular los procesos cognitivos de los seres humanos. Se ha tratado de crear modelos para reproducir el comportamiento de las neuronas, ya sea de forma individual o en forma de red interconectada por la denominada sinapsis neuronal. Según avanzaba el estudio en el propio campo de la neurología y se iban ampliando los conocimientos sobre las funciones y estructura del cerebro humano, estos modelos simulados fueron evolucionando, aunque no siempre para ajustarse de forma precisa a lo que los nuevos estudios biológicos desvelaban. En muchos casos se ha optado por simplificaciones o modificaciones respecto al modelo biológico para conseguir una arquitectura práctica a nivel de computación.

Comienzos: La nueva tecnología de computación desarrollada a mediados del siglo veinte posibilitó a varios estudiosos la posibilidad de plasmar los conocimientos sobre el proceso de pensamiento humano en modelos de computación. Entre ellos destaca Frank Rossenblant que implementó el Perceptron. Sin embargo pronto se constató la limitada capacidad del sistema, lo que llevó a que se estableciera un gran pesimismo en este campo de estudio en los años venideros.

Esquemas actuales: A finales de los años 80 hubo un resurgimiento de las redes neuronales, aunque esta vez se primó la aplicación práctica de esta tecnología por encima de la consecución de un modelo que simulara el comportamiento humano. Se establecieron una serie de arquitecturas para las redes neuronales atendiendo a aspectos como el método de aprendizaje o de entrenamiento. De esta forma existen redes de tipo Hamming o Hopfield(peso fijo), de aprendizaje competitivo o de Mapa de Características(entrenamiento no supervisado), y basadas en decisiones, perceptron, ADALINE, modelos temporales dinámicos, etc(entrenamiento supervisado). Gracias a estas nuevas aproximaciones, las redes neuronales se han convertido en una herramienta de uso práctico en multitud de aplicaciones que deben tomar decisiones a partir de cálculos no deterministas como pueden ser el reconocimiento de imágenes o patrones, predicciones bursátiles, diseño de sistemas complejos, etc.

Futuro: Parece clara que la tendencia actual de buscar el pragmatismo por encima de la fidelidad en la construcción de las redes neuronales se mantendrá en un futuro. Si bien es cierto que el aumento de la capacidad de computación y la previsible revolución que supondría la introducción de la computación cuántica pude hacer que en años venideros se retome el objetivo de modelar el complejo cerebro humano utilizando redes neuronales artificiales.

Robótica

Introducción: El origen de la robótica se difumina a lo largo de la historia debido a que se trata de un área donde se combinan la ingeniería industrial, electrónica, informática, biología, etc, siendo por tanto complejo determinar el momento histórico cuando nace como tal esta disciplina. Sin embargo podemos asumir que la aplicación práctica y comercial de los robots se hizo una realidad a lo largo del siglo veinte, gracias entre otros factores a la revolución electrónica de la miniaturización y el avance derivado del asentamiento de la informática.

Evolución: El objetivo de los robots ha sido inicialmente el de desempeñar de forma automáticas tareas complejas para liberar al hombre de la carga que estas suponían. Siguiendo esta premisa, el carácter práctico de la robótica ha ido colmando los procesos industriales de autómatas que han elevado lo que supuso en su momento la revolución industrial a unas cotas de eficiencia y precisión que constituyen el pilar fundamental de los procesos de producción a gran escala. Este tipo de robots se caracterizan por realizar tareas

relativamente simples de una manera sistemática y precisa. Normalmente su capacidad sensorial y motriz están limitadas para optimizar el desempeño de sus tareas. Sin embargo, en los últimos tiempos otro tipo de robots más sofisticados han empezado a ser utilizados satisfactoriamente en tareas mucho más complicadas como puede ser la exploración espacial, apoyo logístico militar, etc. Estas nuevas máquinas se enfrentan a retos mucho más arduos tales como desenvolverse de forma autónoma en entornos no controlados. Estas nuevas tareas han impulsado el desarrollo de las capacidades de los robots para analizar y "comprender" su entorno, detectar obstáculos, moverse por terrenos no acondicionados, etc.

Tendencias Futuras: Los trabajos más recientes hacen prever el establecimiento de tres tendencias principales en la robótica de aquí a unos años.

El primero, la consolidación de la robótica industrial, cada vez más sofisticada y eficiente. Sirva como ejemplo el proyecto que se está desarrollando en la universidad de Southern California consistente en un enorme sistema robot capaz de construir una casa en una única jornada de trabajo. De forma autónoma será capaz de recoger el material de construcción depositado en la zona de trabajo y en un breve plazo de tiempo levantar muros y paredes siguiendo rigurosamente los planos diseñados por los arquitectos.

En segundo lugar, seguirán evolucionando los robots con capacidad para desempeñar tareas en diferentes tipos de entornos, especialmente en aquellos que constituyen un mayor peligro o incomodidad para los seres humanos. Así asistiremos al desarrollo de nuevos exploradores espaciales, aeronaves sin tripulación, robots con capacidad para combatir y asistir a tropas del ejército, etc. Sin embargo, es probable que en breve plazo este tipo de autómatas empiecen a entrar en el ámbito doméstico en forma de aspiradores autónomos, limpiadores de cristales, etc. Dentro de este grupo parece que se está estableciendo nuevas perspectivas según las cuales la solución no pasará por tener un único y sofisticado aparato, si no que tareas en principio complejas, serán realizadas por multitud de pequeños y simples robots que trabajarán en equipo. Este camino está llevando a la emulación de comportamientos biológicos de tipo enjambre o colonia como el que se da en algunas especies de insectos. Pongamos por ejemplo el trabajo de los ingenieros de la Universidad Libre de Bruselas, quienes han desarrollado un pequeño robot que simula el comportamiento de una cucaracha. En las pruebas previas, esta pequeña máquina consiguió infiltrarse en una colonia de estos insectos. En un futuro el objetivo será introducir varios ejemplares que consigan destacar como líderes en la jerarquía de la colonia para poder determinar el comportamiento de ésta, haciendo por ejemplo que migren de hábitat abandonando

un lugar de uso humano.

Por último podemos suponer un amplio desarrollo de los robots en su sentido más romántico y más presente en la literatura de ciencia ficción. Hablamos de la consecución de robots de formas humanoides y con capacidad de interactuar de forma "inteligente" y "emotiva" con las personas en su día a día. En este sentido varios grupos de investigación están trabajando desde hace tiempo en un programa a largo plazo que concluya con la fabricación de este tipo de entidades. Así hay que destacar el robot Asimo de Honda, probablemente el autómata con la biomecánica más sofisticada desarrollada hasta la actualidad, o los proyectos de universidades como el MIT con Kismet o la de Carnegie Mellon, cuyos objetivos se centran en desarrollar interfaces sensibles a las emociones humanas para futuras entidades robóticas.

OCR (Optical Character Recognition)

Introducción: El problema del reconocimiento óptico de caracteres se trató por primera vez por parte de la NSA (National Security Agency) estadounidense a principio de los años cincuenta. La labor de desencriptar códigos y mensajes cifrados en aquellos años hizo que investigadores como David Sheppard fueran instados a desarrollar un método para reconocer letras de forma automática para su posterior análisis criptográfico. De esta experiencia previa, surgió la empresa IMR que en 1955 desarrolló el primer sistema comercial OCR. Una vez traspasado el campo de seguridad gubernamental, la tecnología OCR se implantó en los servicios postales a partir de la década de los años sesenta para mejorar los procesos de clasificación del correo.

Retos de los sistemas OCR: Cuando tratamos con sistemas del tipo OCR hay que distinguir entre caracteres escritos de forma regular (mediante máquinas de escribir, imprenta, ordenador) o los escritos a mano.

En el primer caso, el reto de analizar un texto de forma efectiva parece haberse alcanzado con bastante éxito, aunque todavía quedan ciertos tipos de codificaciones (caracteres no latinos) que presentan ciertos problemas a la hora de ser analizados correctamente por este tipo de sistemas.

En el caso de la escritura a mano, aunque se están consiguiendo grandes avances, aún queda margen para seguir investigando y optimizando los procesos de aprendizaje de los sistemas y la adaptación a las diferentes escrituras de forma individual.

Futuro: El futuro de esa tecnología parece discurrir en dirección a la consecución de robustos sistemas que permitan, ya no sólo el análisis de un texto, si no la reconstrucción de aquellos cuyas características de conservación y

antigüedad han hecho que se deterioren provocando que su legibilidad sea harto compleja.

Capítulo 2

Diseño y arquitectura

2.1. Introducción

En este capítulo vamos a dar una idea global de la arquitectura de *Reconocimiento visual de instrucciones*. Tiene dos apartados importantes: la arquitectura modular, y la conexión de estos módulos.

2.2. Arquitectura de la aplicación. Módulos en tubería

Al diseñar la aplicación escogimos implementar un sistema altamente modular para conseguir un alto grado de independencia en el desarrollo y de fácil ampliación. Por esta razón hemos invertido parte del trabajo en desarrollar una plataforma propia de enlace de módulos, en la que la una parte de la aplicación (lo que hemos denominado *pipeline* o *tubería*), se encargue de realizar el trabajo mecánico, que, básicamente, se compone de:

- Conectar los módulos mediante puertos independientes con diferente información por puerto, pudiendo crear conexiones *1 a 1*, *n a 1*, *1 a n*, y *n a n*.
- Iniciar y cerrar los módulos, creando y liberando la memoria necesaria y llamando a las funciones pertinentes de cada módulo.
- Gestionar un reloj de ciclos de ejecución, transmitiendo la acción por el grafo que forma la arquitectura de módulos.
- Control de proyectos de aplicación dinámicos, mediante definición de los mismos en **XML**. De esta forma, diferentes archivos de configuración

de proyecto puden crear aplicaciones totalmente distintas sin tener que reprogramar nada.

- Manejo de errores mediante retrollamadas a funciones definidas por el usuario.

El *pipeline* es multiplataforma y funciona con módulos compilados desde *cualquier lenguaje estándar* como bibliotecas dinámicas. Esto dota a la aplicación de un marco muy amplio de uso en muchos ámbitos de desarrollo, no solamente el propio de la visión por computador o la robótica, sino en cualquier aplicación que necesite una arquitectura modular, ya sea secuencial o paralela.

2.2.1. Diagrama general

A continuación representamos, en la figura 2.1, un esquema directo de la aplicación:

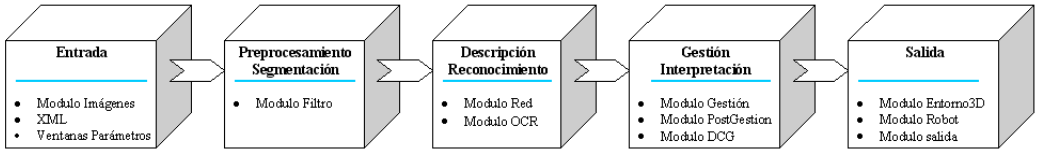


Figura 2.1: Diagrama de la arquitectura

En este esquema, como vemos, tenemos las siguientes etapas:

- **Entrada**: esta etapa está formada por un lado por la captura o generación de imágenes, y por otro, por el XML que define la aplicación (realmente, el hecho de que la aplicación comience como una captura es debido al archivo de proyecto). Además, la entrada del proyecto tiene como parte importante las ventanas de los parámetros de los filtros. Aquí, por tanto, se genera toda la información que va a ser procesada más tarde. Los módulos correspondientes son:
  - Módulo de imágenes.
  - Módulo de ventana de parámetros.
  - Entrada XML.
- **Preprocesamiento y segmentacion**: esta parte la componen los filtros, que procesan la imagen y localizan los objetos de interés. También

incluiríamos la parte de mostrado de imágenes. Preparamos, pues, los datos para reconocer objetos. Los módulos correspondientes son:

  - Módulo de filtros.
  - Módulo de ventana de imágenes.

- **Descripción y reconocimiento**: Son los módulos del OCR, y la red neuronal, que describen los objetos con su base de datos o sus pesos, y los reconocen. Pasada esta etapa, ya tenemos la información en un formato que podemos procesar. Los módulos correspondientes son:

  - Módulo de OCR.
  - Módulo de redes neuronales.

- **Gestión e interpretación**: forman la etapa los módulos de gestión, post-gestión, procesamiento de texto reconocido (gestión de la salida, y buscar sentido a la salida para responder a ella con una respuesta o acción). Aquí filtramos los ruidos generados, y juntamos las señales que se han generado en paralelo.

  - Módulo de gestión de mensajes.
  - Módulo de post-gestión.
  - Módulo de respuesta de texto.

- **Salida**: Entorno 3D, control del robot y modulo de salida (muestran la salida haciendo actuar al robot real o simulado). Por fin, mostramos la salida real de todo el proceso de reconocimiento. Los módulos son:

  - Módulo de salida de texto.
  - Módulo de robot.
  - Módulo de entorno 3D.

2.3. Diagrama exhaustivo de pipeline de "Reconocimiento visual de instrucciones"

En la figura 2.2 hemos esquematizado todo el proceso que siguen los datos de nuestra aplicación hasta llegar a una salida visible por el usuario. Los datos comienzan en la interfaz de imágenes (puede ser una imagen de cámara, un vídeo, una imagen fija...), y descienden por el grafo de módulos hasta el **robot** o el **entorno 3D**. Asimismo, tenemos también de entrada las

ventanas de parámetros, que dotan a los filtros de los valores necesarios *en tiempo real*.

Tras el filtrado pertinente de cada imagen, se las lleva a un módulo de proceso (redes neuronales para los guantes, y algoritmo de *OCR* para el texto), y, paralelamente, a ventanas de visualización, para depuración y comprobación de resultados. Se filtran las señales erróneas y, para el módulo de texto, se envía la información a un módulo de DCG[1] que genera una salida como respuesta "inteligente". Cuando la información ya ha sido extraída de cada imagen, sólo queda unificarla con el resto de datos, para dar una salida coherente.

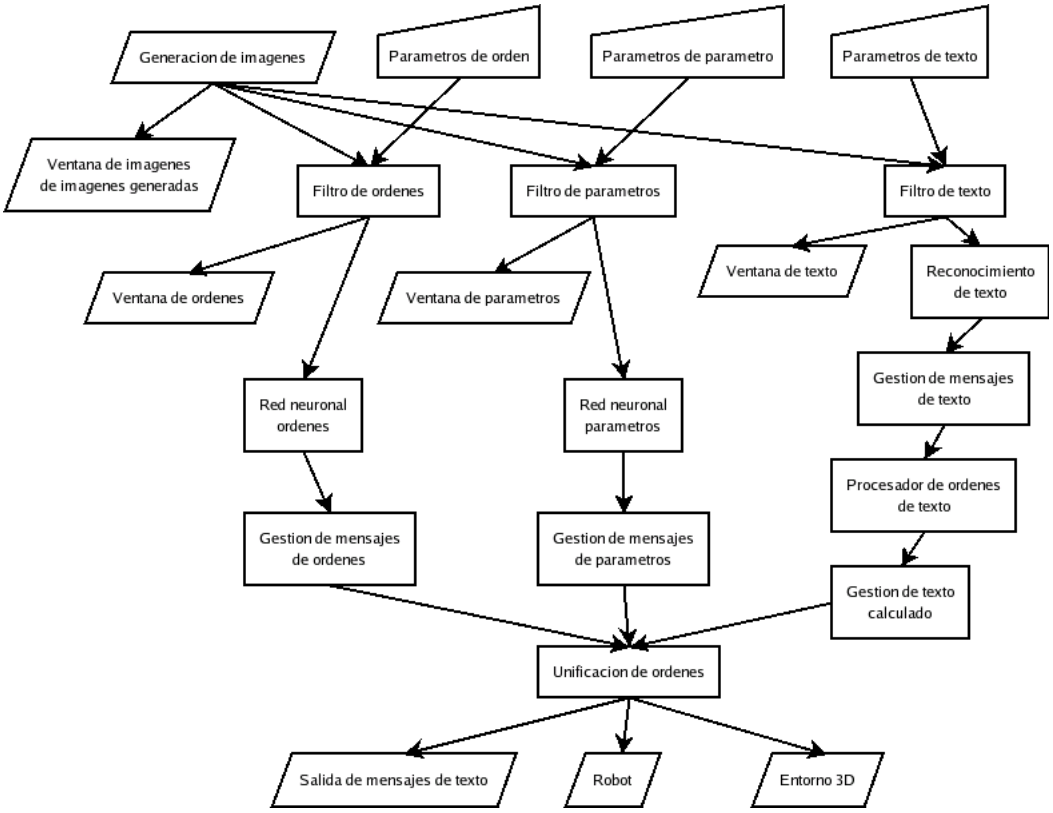


Figura 2.2: Diagrama de tubería

[1]Definite clause grammar

2.4. Diagramas UML

La aplicación no ha sido sintácticamente programada orientada a objetos, sólo algunos módulos. Por tanto, sólo podemos dar unos pocos diagramas. Son el diagrama UML que de forma esquemática muestra las relaciones anteriormente descritas entre las principales clases que componen la apliación del entorno 3D en la figura 2.3, y el diagrama de la clase que, en Win32, usa una cámara web en la figura 2.4.

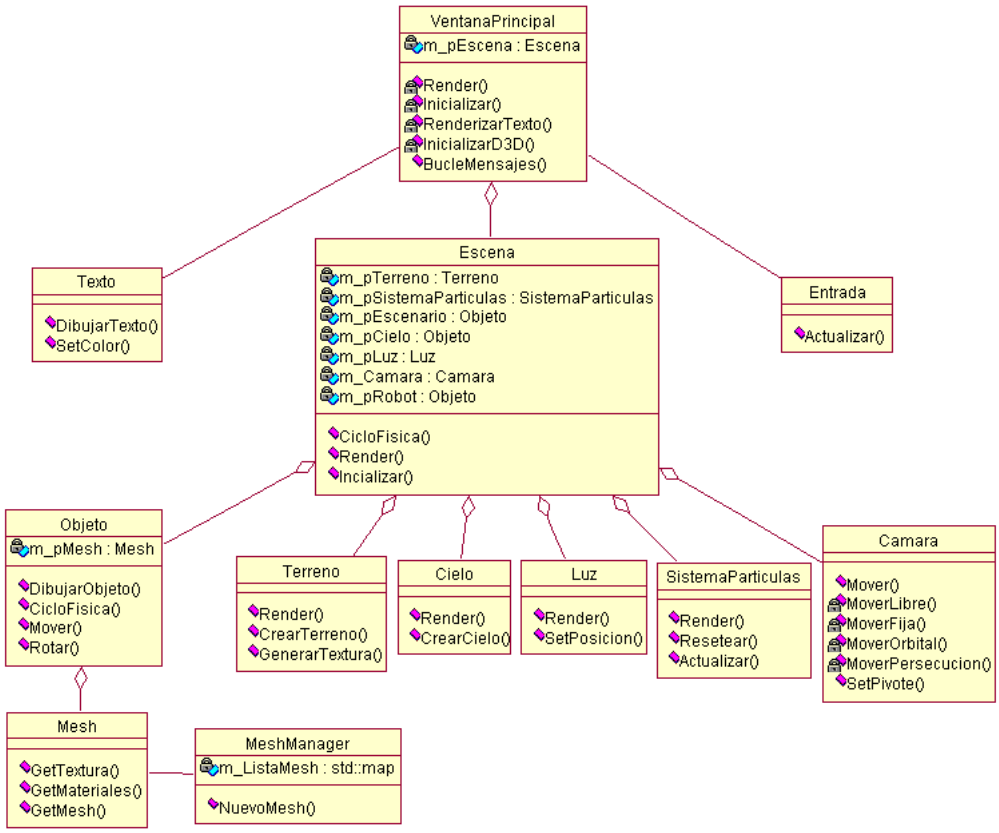


Figura 2.3: Diagrama de clases UML del entorno 3D

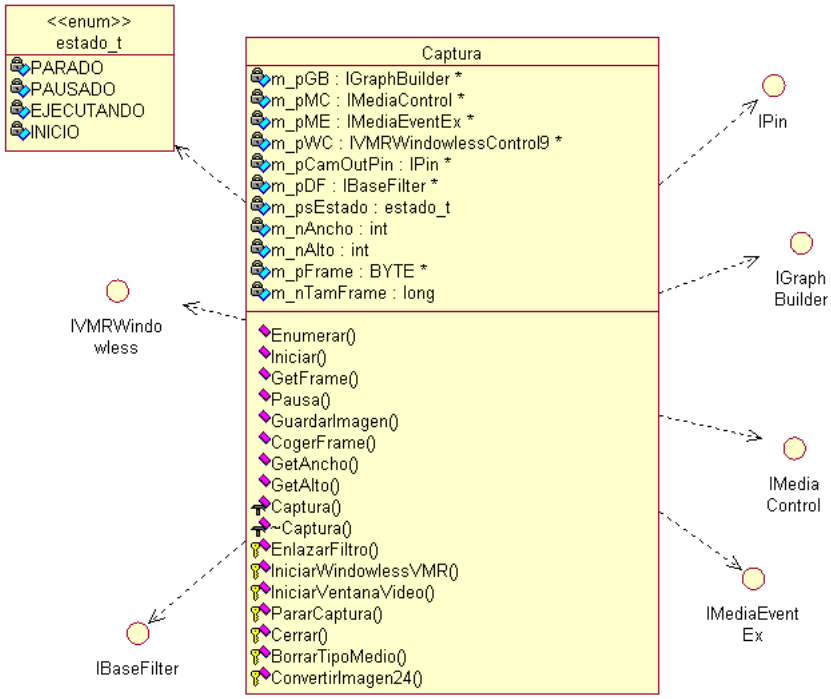


Figura 2.4: Diagrama de clases UML de la cámara

Capítulo 3

Resultados del proyecto

3.1. Pruebas

3.1.1. Pruebas de arquitectura

Pruebas de desarrollo

Uno de los mayores retos del proyecto ha sido conseguir una arquitectura sólida que satisficiera nuestras necesidades de conexión. La evolución de la generación de la aplicación ha llevado implícitas las pruebas diarias del éxito del diseño y la implementación de nuestro *pipeline*. Por eso detallamos los cambios del mismo, como resultado de nuestros experimentos.

La primera visión del diseño global era mucho más simple que la que finalmente hemos acabado usando: los módulos sólo se conectaban en forma de árbol. Pronto tuvimos que abandonar este enfoque, pues los requisitos de conexión de módulos se mostraron más complejos de lo que estimamos en un principio.

Así pues, pensamos en conectar los módulos *1 a 1* en forma lineal. Los primeros resultados fueron satisfactorios: los módulos se comunicaban una vez que la implementación del diseño dejó de tener errores. La implementación en este punto comenzaba a ser sólida, y pronto ampliamos el diseño para que los módulos tuvieran conexiones de *1 a N*. Con esto conseguimos, por ejemplo, ver las imágenes que generaban los filtros sin tener que cambiar en absoluto los módulos que operaban. El diseño comenzaba a dar sus frutos, empezábamos a ahorrar horas de trabajos y a reutilizar fuertemente el código desarrollado, ya que un módulo de "ventana de imágenes" podía, instanciándose varias veces, presentar diferentes imágenes.

El diseño ya comenzaba a ser realmente sólido, sin embargo, nos vimos en la necesidad de que un módulo aceptase varias entradas. Por tanto, añadimos

esa funcionalidad. En esta fase del diseño completamos casi toda la aplicación. Las pruebas y la corrección de errores fueron paralelas y terminaron por dar con un conjunto muy fiable, con conexiones *N a N*.

En última instancia, nos dimos cuenta de que el sistema de puertos no era del todo completo. Un módulo sólo se podía conectar a otro por un puerto. Esto nos presentaba el inconveniente en el módulo de cálculo de respuesta, que debía ofrecer la salida de la orden y el parámetro de forma independiente. Finalmente, pues, añadimos más potencia a los puertos, dotando a la estructura de una completitud amplia y sólida.

Como ejemplo global de pruebas y de la utilidad de la arquitectura de *pipeline*, podemos reseñar el del módulo de "gestión". Poco valorado al principio, pensamos que iba a ser un simple trámite de la salida global. Sin embargo, finalmente el diagrama total tiene 4 instancias del módulo, para las cuales no hemos tenido que reprogramar nada, sólo variar el archivo de proyecto XML.

Eficiencia

El *pipeline* no es un ejemplo de velocidad de proceso. Las pruebas que hemos realizado nos han permitido, en un ordenador con un microprocesador *Intel Pentium IV* con una frecuencia de reloj de 3,0 GHz, alcanzar velocidades de ciclo de 200 milisegundos. Para una aplicación del ámbito docente como es la que presentamos, el resultado es desde luego más que suficiente, pero no deja de ser un tiempo de ejecución lento para requisitos como, por ejemplo, de tiempo real.

Comprobación de los módulos

A continuación detallamos las pruebas del aspecto arquitectónico de los módulos:

- **Generación imágenes y ventana de imágenes**: El resultado esperado de este módulo era la generación de imágenes desde diversas fuentes, en un formato unificado. Para esto, usamos principalmente la ventana de imágenes, comprobando que las imágenes correspondiesen a lo esperado. Para ejemplos de resultados, puede remitirse a la sección 4.6.

- **Filtros**: La serie de transformaciones que sufre la imagen en el módulo de filtro (gestos o carteles) es el resultado de un largo periodo de investigación sobre imágenes de prueba.

La elección de una transformación u otra ha estado dirigida siempre con el propósito de conseguir un análisis posterior mucho mas simple y fiable.

Todos los filtros que fuimos creando que no presentaban valor añadido han sido eliminados, quedando así los mínimos necesarios para facilitar la extracción de información de la imagen en el posterior análisis (red neuronal, OCR).

La elección del tamaño de las mascaras utilizadas o la creación de varios filtros dentro de los mismos bucles se ha realizado siempre con la intención de que la fase de filtro sea lo mas rápida posible, no impidiendo que la aplicación funcione en tiempo real.

Tiempo aproximado del filtro de gestos: 5 milisegundos.

Tiempo aproximado del filtro de carteles: 6.5 milisegundos.

- **Parámetros**: El módulo de generación de parámetros tuvo algunos inconvenientes. En un principio, hicimos un programa con el código base de lo que iba a ser el módulo, consistente en una ventana que generaba estructuras de datos con los valores elegidos. El funcionamiento del programa fue exitoso, cosa que vimos imprimiendo por pantalla el contenido de dichas estructuras. Cuando integramos el módulo (ya programado como tal) en la aplicación, tuvimos algunos problemas, pues los mensajes no llegaban bien al módulo de filtros. Las pruebas nos llevaron a la conclusión de que fue un fallo de arquitectura, con lo que tuvimos que remodelar el diseño del núcleo del pipeline para que admitiese más tipos de conexiones. Tras esto, el resultado fue satisfactorio.

- **OCR**: Ha sido uno de los principales retos de este proyecto. Al principio pensamos en utilizar un OCR ya implementado, pero tenía el problema de que no se ajustaba completamente a nuestras necesidades. Probamos a implementarlo nosotros con algoritmos ya realizados, como descriptores de regiones, pero esos sistemas resultaban ser demasiado lentos cuando el numero de caracteres a reconocer dentro de la imagen aumentaba y como siempre el tiempo ha sido un factor que ha corrido en nuestra contra. Llegamos a plantearnos usar la red neurnal tambien en este campo, pero comprobamos que su entrenamiento era muy costoso, además que no cumplia con las exigencias como por ejemplo que el tamaño de los caracteres no importe. Desarrollamos por tanto nuestro propio algoritmo descriptor reconocedor de caracteres basado en descripción de fronteras. La cantidad de información que este método

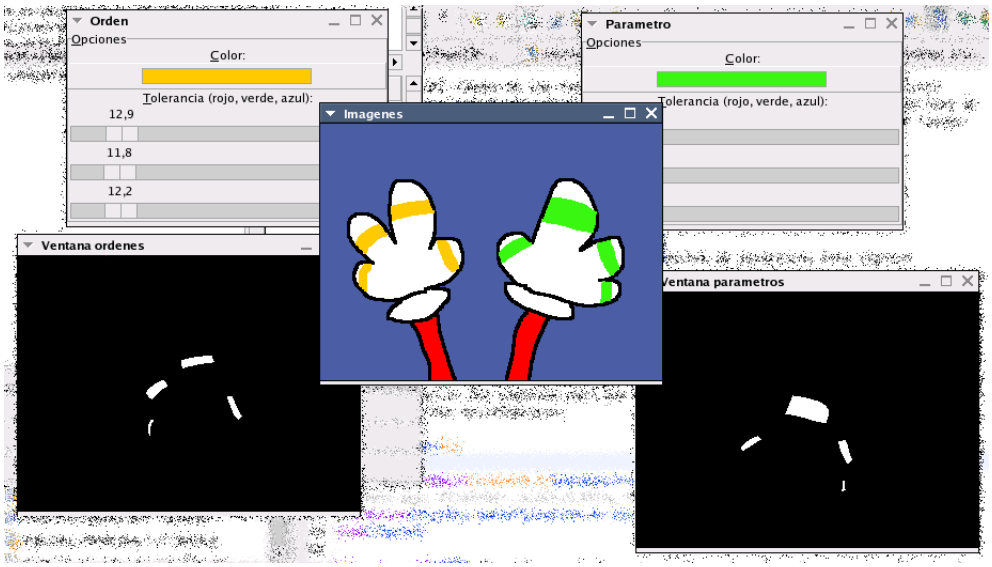


Figura 3.1: Pruebas satisfactorias de los parámetros

necesita para describir la frontera de un objeto es muy pequeña lo que acelero todo el proceso. Si queremos que sea muy preciso y que nunca se equivoque reconociendo un objeto aumentamos la cantidad de información que debe tratar, esto aumenta el tiempo ligeramente, así que en vez de hacer eso decidimos usar a la salida del OCR un diccionario que comprobara si las palabras que salían tenían o no sentido, si no lo tenían las corregía.

- **Gestión de mensajes**: La gestión de mensajes fue probada a través de su funcionamiento, y mostrando la salida por consola. La comprobación de la corrección la hemos realizado imprimiendo por la salida estándar el estado del grafo de mensajes en todo momento.

- **Post-gestión de órdenes**: El módulo de gestión total de la tubería de órdenes ha tenido pruebas triviales, debido a su sencillez, simplemente, hemos certificado mediante el uso que las órdenes llegaban bien a los módulos de salida.

- **Proceso de texto**: El proceso de texto ha sido implementado en Prolog, por lo que las pruebas han sido realizadas de una manera externa a la aplicación, con el intérprete de SWI-Prolog. Gracias a este método, el desarrollo fue más rápido, ya que la comprobación conun intérprete es muy ágil. Cuando funcionó por separado, la integración en la aplicación principal no causó ningún problema, y funcionó tal y como lo habíamos

previsto, con lo que no hizo falta realizar más pruebas que la pura comprobación del funcionamiento en ejecuciones normales. Podemos ver un ejemplo en 3.1.1.

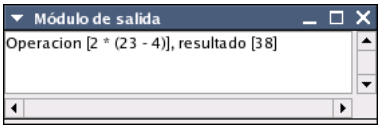


Figura 3.2: Proceso de texto

- **Robot**: El desarrollo del robot tuvo un trabajo paralelo. Al principio, las pruebas fueron paralelas a la construcción de la estructura, y controlábamos los motores mediante un control de corriente contínua. Las pruebas nos llevaron a la conclusión de que había que remodelar los árboles de engranajes y los cambios de par, pues en un principio no suministraban la suficiente potencia como para mover todo el peso de la estructura. Tuvimos que cambiar esto, y fue un cambio bastante grande. Una vez que el robot se movía con el control remoto, probamos a crear el circuito que iba a hacer de capa entre el puerto paralelo y los motores. Conectamos el circuito, y sobre la placa las tensiones funcionaban bien. Lo ensamblamos al robot, y, aunque en un principio funcionaba bien, pronto dejó de hacerlo. Tras depurar, vimos que había un fallo en un punto de la placa (nos costó bastante averiguarlo), y, una vez corregido esto, el robot comenzó a funcionar de una manera muy estable.

- **Red neuronal**: Las pruebas sobre la red fueron realizadas antes de empezar el proyecto. Se realizaron pruebas sobre cientos de fotos para el reconocimiento de ciertos patrones, para ello implementamos un pequeño programa de entrenamiento donde contabilizamos el índice de aprendizaje sobre el conjunto de fotos de entrenamiento, el índice de fallos que cometía sobre otro conjunto de prueba y sobre otro de validación. Se contabilizaba tanto el índice de aciertos como de error, modificando el factor de aprendizaje, el conjunto de entrenamiento y el numero de iteraciones.

  Concretamente para el reconocimiento de los 4 gestos que se hacen con la mano al robot se necesitaron 148 fotos de entrenamiento, 49 fotos de validación, 30 de prueba y 20 vueltas de aprendizaje, esto equivale en un Pentium4 a 3 horas de aprendizaje ajustando los pesos de la red.

  Porcentaje de aciertos en entrenamiento: 89 Error medio: 0,0141046521582562

Porcentaje de aciertos en validación: 93 Error medio: 0,00799933215976971

Porcentaje de aciertos en prueba: 100 Error medio: 0,00645778571591585v

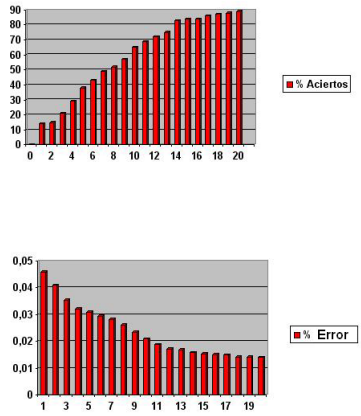


Figura 3.3: La imagen de la izquierda muestra como aumenta el aprendizaje cuanto mas se enseña a la red. Aumentando el nº de aciertos. Y la de la derecha muestra como a medida que aprende comete menos errores reconociendo figuras.

Una vez guardados los pesos en un archivo solo hay que cargarlos en una nueva red cada vez que se quieran utilizar y la retropropagación para el reconocimiento de futuros patrones es casi instantánea, que es lo que realmente importa. La eficiencia de la red reconociendo patrones es mejorada por el uso de los filtros previos.

- **Entorno 3D**: Inicialmente la simulación 3D se probó exahustivamente de forma aislada para cetificar el correcto funcionamiento de cada uno de los elementos especificados(movimiento del robot, seguimiento de la cámara, iluminación, etc). En dichas pruebas se utilizó una entrada directa por teclado gestionada internamente por la aplicación para controlar la navegación del robot a través del escenario. A la hora de realizar la integración con el resto de módulos, simplemente se sustituyó el control por teclado interno por los comandos recibidos a través de los puertos de conexión resultantes de los análisis previos de los gestos del usuario. Una vez establecida la conexión del módulo con el pipe, sólo hubo que probar que cada uno de los comandos analizados producían el comportamiento deseado en la simulación.Además, como

elemento redundante se añadió una salida en texto que mostrara al usuario el comando en ejecución en un instante determinado.

Figura 3.4: Entorno 3D

- **Salida de texto**: Para las pruebas de la salida de texto simplemente hemos comprobado que el texto que mandábamos al módulo salía por la ventana, añadimos un ejemplo en la figura 3.1.1.

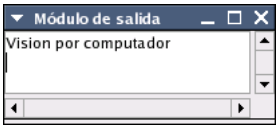


Figura 3.5: Salida de texto

- **Joystick**: La comprobación del módulo de joystick se ha ido haciendo a medida que la integración avanzaba. Al ser un módulo que se ha desarrollado en la fase final de la aplicación, la estructura de la misma ya estaba bastante sólida, y el funcionamiento del módulo ha sido casi inmediato.

3.2. Trabajo futuro

Dada la extensión del proyecto y que concierne a varios ámbitos de desarrollo, vamos a dividir esta sección según los apartados del mismo:

3.2.1. La aplicación y los módulos

Los módulos de la aplicación son la parte más interesante del trabajo posterior al que puede dar lugar este proyecto. Cada uno es independiente

de los demás, y sólo "conocen" entre sí las interfaces de entrada/salida que los definen. Por tanto, cualquiera de estos módulos puede ser incluido en un proyecto de visión por computador (en un principio, algunos módulos tienen una funcionalidad más general). Además, la aplicación que hemos creado puede sera ampliada añadiendo más módulos (para crear un sistema más inteligente, y enlazar esos nuevos módulos con la salida ya existente), o sustituyendo algunos, por ejemplo.

3.2.2. Pipeline

El pipeline consiste en una librería en C que conecta módulos y pasa información entre ellos. De este modo, sólo es necesario crear dichos módulos, conectarlos (escribiendo un archivo en *XML*) y establecer sus argumentos (en el *XML* también). Por tanto, esta pieza de código es perfectamente reusable en cualquier aplicación que necesite una arquitectura modular de elementos independientes que necesiten ser conectados entre sí.

Por otro lado, podría tener interés la creación, como proyecto futuro, de un editor visual de dichos esquemas de conexión de módulos, que generase y leyese documentos en XML para el pipeline. Esto supondría una gran ventaja para la creación rápida de aplicaciones y la prueba *in situ* de los proyectos.

Además, tendría gran utilidad una reimplemtación de la especificación del pipeline, de tal modo que la velocidad fuese mayor.

3.2.3. Módulos

Cada módulo es una caja negra que realiza una función de visión aumentado la capacidad de visión de la aplicación. Nosotros hemos desarrollado una serie de modulo capaces de reconocer gestos manuales o mensajes dentro de carteles.

Posibles proyectos futuros pueden realizar diversos módulos mejorando la interpretación visual del robot. Por ejemplo módulos que detecten:

- Zonas luminosas o tipo de absorción de la luz.
- Objetos móviles y tipo de movimiento.
- Forma y volumen de objetos.
- Textura y colores de superficies.
- Etc.

Cuanto mayor sea la capacidad de recoger características de la imagen de entrada, mayor será la capacidad de interpretación de esta por parte del robot. Si en un futuro se desarrollaran estos módulos, el robot sería capaz por ejemplo de detectar un objeto con una iluminación, un volumen, un color y textura determinada que coincidiría con la descripción que tiene almacenada de como es una *mesa*. Sabría por tanto que dentro de la imagen existe una mesa, lo mismo pasaría con el resto de objetos del entorno. Sabiendo los objetos que existen en la imagen puede interpretar donde esta y como es el lugar.

Si en un futuro este proyecto sigue desarrollándose podría servir como descriptor automático de imágenes. O quizás como ayuda para invidentes.

3.2.4. Robot

Mediante la construcción del robot hemos demostrado la simpleza de la interacción entre software y hardware muy cercano al usuario. Los módulos y las rutinas que hemos desarrollado pueden servir bien como ejemplo o base para nuevos proyectos, o como módulos ya escritos que pueden ser usados mediante el *pipeline*.

Una posible ampliación muy atractiva y visual sería añadir más funcionalidad al robot, ampliando el circuito y añadiendo motores, de forma que tuviese, por ejemplo, un brazo que pudiese coger cosas.

3.2.5. Entorno 3D

A continuación se enumeran una serie de ampliaciones y mejoras que se podrían introducir en el entorno 3D en futuras expansiones del proyecto:

Detección de colisiones entre el robot y el entorno

Se debería implementar un sistema para calcular en cada instante la posición del punto de contacto del robot con el terreno sobre el que se encuentra. De esta forma el robot podría navegar a través de entornos que contasen con suelos accidentados. Además de debería calcular posibles colisiones con distintos elementos como paredes, obstáculos y cualquier otro tipo de entidad física que se encuentre dentro de la simulación.

Para implementar la opción de que el robot se desplace sobre una superficie abrupta, se sugiere implementar un sistema que a partir de las coordenadas XZ del robot, se obtenga la correspondiente altura del terreno en ese punto, es decir la coordenada Y, la cual servirá para situar al robot a la

altura adecuada. Este cálculo es trivial y consiste en una simple interpolación entre los vértices que componen el triángulo en cuyo interior se encuentra el punto XZ proyectado sobre el plano homónimo. Para realizar estos cálculos es necesario acceder a la información geométrica del terreno. Dicha información es accesible de una forma simple a través de la clase Terreno que se ha implementado.

En el caso de las colisiones con otros elementos del entorno lo más sencillo sería implementar un sistema de colisiones jerárquico. Este sistema consiste en ir realizando diferentes test de colisión que, de forma gradual, fueran refinando la precisión del cálculo de la colisión. De esta forma, se empezaría por realizar un test de colisión entre las bounding box (cajas imaginarias que engloban el volumen de una entidad 3D) del robot con los diferentes objetos estáticos del entorno. Si no existe colisión a este nivel, podemos asegurar que no hay colisión y por tanto terminar en este punto el cálculo. Si se produce colisión entre diferentes bounding box, debemos asegurarnos de que verdaderamente la geometría contenida en las cajas están en contacto. Para ello se pasaría a un segundo test de grano mucho más fino, en el que se realizarían comprobaciones de colisión entre los triángulos de cada una de las entidades 3D implicadas en la colisión. Este último cálculo suele ser bastante costos en cuanto a tiempo de computación, por eso es importante descartar en una primera pasada todas las situaciones que no requieran un cálculo tan preciso. De nuevo, la clase Objeto proporcionada, da acceso a todos los datos de la geometría necesarios para realizar estos cálculos. Además, el propio lenguaje gráfico empleado (Direct3D), proporciona a través de su librería auxiliar D3DX multitud de funciones que facilitan la implementación del sistema de colisiones aquí propuesto.

Añadiremos una última cuestión respecto al tema de las colisiones. Si se quisiera representar un entorno extremadamente complejo, con una gran cantidad de geometría 3D, sería necesario añadir un nivel más a la jerarquía de colisiones previamente expuesta. Sería adecuado realizar en primera instancia un test previo en el cuál se detectaría en que sector del escenario se encuentra el robot, pudiendo descartar en esta primera pasada todos aquellos elementos que se encontrarán fuera de dicho sector. Luego se pasaría a realizar el algoritmo ya explicado únicamente con los objetos que comparten sector con el robot. Para implementar este sistema, habría que crear una estructura de partición del entorno. Se pueden optar por distintas aproximaciones como el uso de árboles BSP, Octrees, Quadtrees, etc. Sobre estas estructuras existe multitud de documentación al ser algoritmos de uso común en computación gráfica.

Creación de rutas de navegación

Para demostrar la funcionalidad del sistema de visión por computador como sistema de control de un robot, se podrían establecer diferentes rutas en el entorno 3D, que el robot debe seguir guiado por el usuario. De esta forma, el usuario debería guiar al robot a lo largo de una serie de puntos de control o waypoints para completar un recorrido. Además se podría incrementar la dificultad de dicho reto imponiendo reglas como recorrer la ruta en un tiempo determinado o a una velocidad lineal o de giro mínima.

La implementación de esta ampliación sería trivial una vez implementado el sistema de colisiones previamente propuesto. Así, simplemente habría que disponer una serie de objetos que se corresponderían con los puntos de navegación e ir comprobando si el robot colisiona con ellos en el orden adecuado. Obviamente habría que inhibir cualquier respuesta motriz a dicha colisión, ya que estos objetos no debería impedir el movimiento del robot.

Modificación de los modelos (Robot, Escenario o Terreno)

Si se quiere cambiar el aspecto de la simulación, incorporando nuevos modelos para los objetos 3D, simplemente habría que convertir los nuevos modelos desde su formato origen (normalmente formatos de programas de modelado 3D como 3Dstudio o Maya) al formato .X que es el que se usa en la aplicación. Dicha conversión se puede realizar a partir de los plugins que incluye el SDK de DirectX. Para modificar el terreno, únicamente hay que crear un nuevo mapa de altura (en formato crudo o RAW) y utilizar la pequeña aplicación auxiliar desarrollada para generar terrenos que configura el tamaño y grado de desnivel de la geometría así como las diferentes capas de texturas que se usarán en función de la altura.

Mejoras en el apartado gráfico

Para incrementar el aspecto visual de la aplicación se pueden introducir en el futuro nuevos efectos gráficos. Por ejemplo, se podría introducir el uso de texturas HDR que optimicen el comportamiento visual de la iluminación y los reflejos de los objetos 3D. Se podría incluir también la novedosa técnica PRT(precomputed radiance transfer) que permite implementar la técnica de iluminación global por radiosidad en tiempo real (esta característica se ha incorporado a la nueva versión de DirectX aparecida en Abril de 2005). Por supuesto, el uso de shaders para desarrollar nuevos efectos sería también muy apropiado, teniendo en cuenta además lo fácil de su programación gracias a las facilidades de Direct3D para integrar esta tecnología.

Cualquier otro tipo de modificación será fácilmente implementada ya que el diseño de la aplicación y su modularidad permiten una sencilla integración de nuevos componentes. Para más detalles consultar el diseño en el Anexo correspondiente al Entorno 3D.

3.3. Conclusiones

Desde que comenzamos y hasta la finalización completa del proyecto, hemos pasado por varias etapas. En una primera fase, afrontamos la aplicación como algo ligeramente borroso, no demasiado definido. Poco a poco fue tomando forma, proceso en el cual fuimos haciéndonos conscientes de algunos puntos clave en el desarrollo de un proyecto como éste. Los más importantes que podemos esquematizar como conclusiones, son los que detallamos a continuación:

El proceso de desarrollo no ha de ser tomado como un bloque global que atacar directamente

Esto sólo lleva a un código confuso y muy acoplado, que da muy poca oportunidad a la expansión o a la reutilización del mismo, incluso dentro del proyecto. Hemos tenido la suerte de hacernos cargo de esta circunstancia en una fase poco avanzada del proyecto, e invertir un esfuerzo extra en organizar las cosas, dividir el trabajo a conciencia, con sentido, y asignando a cada cual las partes en las que más puede desarrollar sus habilidades. Asimismo, todo el tiempo invertido en realizar una buena arquitectura de trabajo, nos ha dado alas para ampliar la aplicación en la medida en que las necesidades del proceso fueron ampliándose.

El cumpliminento de los requisitos del proyecto ha de ser el objetivo primario

Con esto queremos señalar que, a pesar de que todo el proyecto se daba, como idea y como realidad, de una manera muy importante a la experimentación y a las pruebas, nuestra principal prioridad era cumplir con los requisitos primeros que estaban impuestos como fin del proceso de desarrollo. De esta manera, conseguimos poner punto y final al proceso de implementación bastante antes de la fecha límite, pudiéndonos dedicar al desarrollo de una memoria concreta, explicativa, y que abarcara todos los aspectos que hemos considerado importantes, así como a perfeccionar esos detalles que siempre quedan como **objetivos secundarios**, pero que tanto realzan el contenido final de la aplicación.

Es posible llevar a cabo las ideas que han conducido a la motivación de la realización de un proyecto

Decidimos trabajar en la visión por computador y en robótica porque teníamos ideas que habíamos madurado durante algún tiempo, y queríamos llevarlas a cabo. Tras la finalización del trabajo, nos hemos percatado de que teniendo un método serio de trabajo y aprendizaje, siendo consecuentes con los plazos de entrega que nosotros mismos nos impusimos, y trabajando con tesón, es posible conseguir plasmar en un proyecto de software (y ligeramente, hardware), como es el nuestro, aquellas ideas que habíamos tenido. Eso sí, perfeccionándolas y eliminando de las mismas todas aquellas partes que eran imposibles en el marco de un año de trabajo.

La visión por computador es un campo en el que queda mucho por hacer

Si bien es verdad que no hemos llegado ni mucho menos a lo más profundo, innovador e interesante de este campo, sí es cierto que podemos decir, tras haber estado un año experimentando y trabajando, que la visión por computador tiene mucho por delante aún. A nuestro nivel nos hemos dado cuenta, por ejemplo, de lo muy sensible que es el filtrado de imágenes al tipo de luz que incida sobre los objeto. Del reconocimiento de gestos de una imagen fija previamente pensada al reconocimiento en tiempo real hay un abismo en la efectividad de resultados. La visión por computador, según estos ejemplos y otros muchos que hemos encontrado es, a nuestro parecer, una ciencia que aún tiene que desarrollarse mucho.

3.4. Agradecimientos

- Gracias al profesor Dr. D. José Antonio López Orozco, del Departamento de Arquitectura de Computadores y Automática, por su ayuda en la construcción del robot.
- Gracias a Juan Rodríguez, compañero y alumno en la Facultad de Informática, por su ayuda con el reconocedor en Prolog.

Bibliografía

[1] Gonzalo Pajares, Jesús M. de la Cruz "Visión por Computador: Imágenes Digitales y Aplicaciones" Ra-Ma.

[2] Gonzalo Pajares, Jesús M. de la Cruz, José M. Molina, Juan Cuadrado, Alejandro López "Imágenes Digitales" Ra-Ma.

[3] Tom M. Mitchell "Machine Learning" McGraw-Hill, 1997.

[4] Ian H. Witten, Eibe Frank "Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations" Morgan Kaufmann, 1999.

[5] Russell, S. y Norvig, P. "Artificial Intelligence: A Modern Approach" Prentice Hall, 2004.

[6] Rich, E. y Knight, K. "Artificial Intelligence" McGraw-Hill, 1994.

[7] Frank D.Luna "Introduction to 3D game programming with DirectX 9.0" Wordware Publishing, Inc

[8] Trent Pollack "Focus On 3D Terrain Programming" Muska y Lipman/Premier-Trade

[9] Julio Sanchez, Maria P.Canton "DirectX 3D Graphics Programming Bible" IDG Books Worldwide,Inc.

[10] Estephan Zerbst "3D Game Engine Design" Premier Press.

[11] Aníbal Ollero "Robótica: Manipuladores y robots móviles" Marcombo

[12] Edwin Wise "Applied Robotics" Prompt Publications

[13] Phillip John McKerrow "Introduction to Robotics" Addison-Wesley

[14] K.S. Fu, R.C. González, C. S. G. Lee "Robótica: control, detección, visión e inteligencia" McGraw-Hill

Parte II

Detalles de los módulos

Capítulo 4

Módulo de generación de imágenes

4.1. Introducción

Este módulo tiene como objetivo la creación y modificación de un búfer de colores (una imagen en formato plano) para la alimentación de la tubería de visión. Obtiene, de diferentes fuentes, imagenes codificadas de diferentes maneras, y genera, en un formato unificado, una imagen sin comprimir.

4.2. Detalles

- **Entrada**: Este módulo no tiene entradas del pipeline. Sólo los datos de la imagene externa. Como argumentos, la función que debe realizar, y el tamaño y los bits de la imagen que genera.
- **Salida**: Una estructura de datos como la que definimos en 4.3.
- **Descripción**: Módulo para la captación de imágenes de cámara, archivos, vídeo o generadas automáticamente, que alimenta al pipeline.

4.3. Imágenes generadas

La funcionalidad principal de este módulo es, como hemos comentado antes, la de generar imágenes de un solo formato, independientemente del origen del que se obtengan. Con este procedimiento pretendemos establecer la base del árbol de módulos de nuestro proyecto. Previamente al diseño y la implementación del módulo, establecimos cuáles eran los requisitos que

debía cumplir el formato de imágenes que íbamos a usar. En primer lugar, las imágenes debían ser en color. Para esto, debíamos ser capaces de manejar la profundidad de cada punto de la pantalla. Además de esto, queríamos que las dimensiones de las imágenes (ancho y alto) fuesen flexibles, y, a priori, ser capaces de manejar cualquier tamaño (aunque posteriormente esto no ha sido realmente posible, pues la cámara web sólo admite un conjunto de dimensiones determinado).

Estos dos requisitos, más la tendencia que hemos intentado mantener de mantener el proyecto lo más simple posible (filosofía *KISS*[1]), nos han llevado a definir nuestro formato de imágenes interno de la siguiente forma:

- **Alto**: Un entero que determina el alto de la imagen.
- **Ancho**: Un entero que determina el ancho de la imagen.
- **Bytes**: Un entero que determina el número de bytes por punto.
- **Buffer**: Un puntero (en la implementación de C, realmente implementa un vector) de número de 8 bits sin signo (1 byte, corresponde al formato `unsigned char` en C), que contiene la información de colores de la imagen.

Esta ha sido toda la información por imagen que nos ha sido necesaria. El hecho de disponer de un búfer lineal nos ha capacitado para escribir implentaciones muy rápidas del recorrido de las mismas, principalmente por la potencia de los punteros del lenguaje C.

4.4. Arquitectura y funcionamiento del módulo

El módulo sigue las interfaces de comunicación con el *pipeline*, de modo que crea los búferes en la función de iniciar, a la vez que, según se haya instanciado a través de la configuración del XML que define el proyecto, abre la comunicación con las bibliotecas pertinentes, en función de los argumentos.

En el ciclo de imágenes se procede según sea el funcionamiento. En el caso de que las imágenes cambien cada ciclo (no pasa cuando son imágenes de un color fijo o cargadas de archivo), se obtiene del recurso indicado el búfer en el formato que ofrezca la biblioteca, y se transforma al formato de salida común, como se ha comentado en la sección anterior. Una vez que se ha hecho esto,

[1] Keep it simple, stupid.

se "deposita" en el puerto de salida la imagen resultante, habiendo ya dado uniformidad a las diferentes entradas, haciendo que el *pipeline* no dependa de las fuentes generadores de imágenes a más bajo nivel.

En la función de cerrar simplemente libera los recursos.

4.5. Bibliotecas

Para la generación resultados desde diferentes fuentes, el módulo hace uso de una serie de librerías; son las siguientes:

- **DirectX**: Las bibliotecas *DirectX* nos han provisto de la interfaz necesaria en su módulo *DirectShow*, para la adquisición de imágenes de cámara.
- **SANE**: Sane (Scanner Access Now Easy) es una biblioteca originariamente para interfaces con escáneres, pero ampliada para cualquier dispositivo de imágenes. A través de esta biblioteca accedemos a la cámara.
- **Xine**: Xine-lib tiene como objetivo la reproducción de archivos de vídeo en varios formatos (los más usuales, también puede aceptar formatos nuevos a través de *plugins*). A través de Xine cargamos un vídeo en el módulo de imágenes y lo reproducimos.
- **Gdk**: Usamos Gdk para cargar fácilmente archivos de imágenes, y usarlos así como fuentes sencillas de imágnenes cuyo resultado se conoce.
- **Código propio**: También hemos implementado, para pruebas principalmente, las siguientes funcionalidades del módulo:
  - **Colores planos**: Pintamos todo el búfer de un color. Es muy útil para comprobar el funcionamiento de los filtros.
  - **Imágenes aleatorias**: Rellena la matriz de colores con colores al azar, de esta manera vemos cómo los filtros aceptan o dejan de aceptar los valores.

4.6. Ejemplos

Estos son ejemplos del funcionamiento del módulo en una ejecución normal:

Figura 4.1: Color fijo

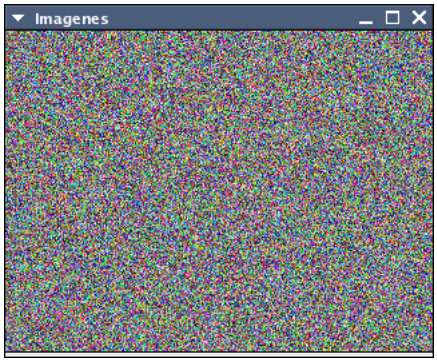


Figura 4.2: Imagen aleatoria

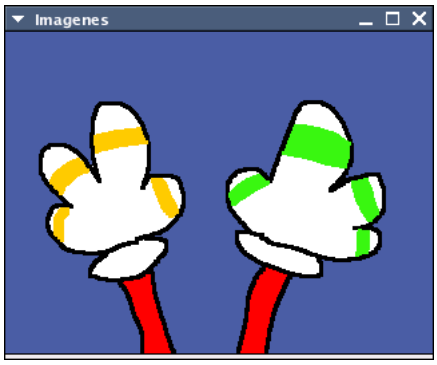


Figura 4.3: Imagen de archivo

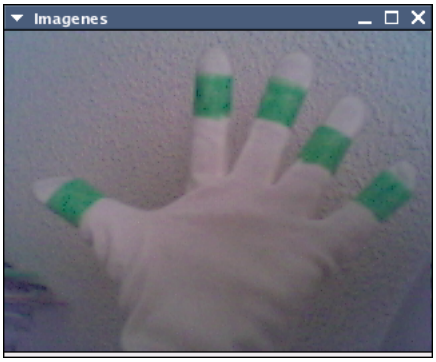


Figura 4.4: Cámara

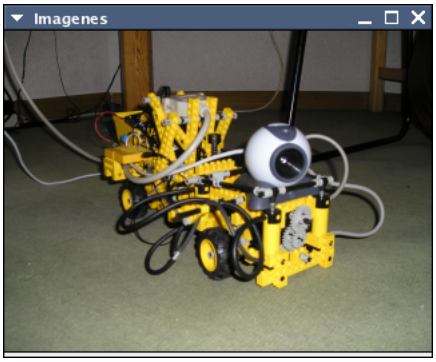


Figura 4.5: Vídeo

Capítulo 5

Módulo de Filtro

5.1. Introducción

La visión artificial es el proceso sensorial más complejo de todos. Las tareas en visón por computador se pueden enumerar en:

1. Visión de bajo nivel. (Tareas automáticas)
   - Captación. Obtención de la imagen.
   - Preprocesamiento. Incluye técnicas como reducción de ruido y realce de detalles.
2. Visión de nivel medio. (Etiquetar objetos)
   - Segmentación. Localización de los objetos de interés.
   - Descripción. Obtención de características: tamaño, formas, etc.
3. Visión de alto nivel. (Emular inteligencia)
   - Reconocimiento. Identificación de objetos: tornillos, puertas, etc.
   - Interpretación. Significado de un conjunto de objetos.

El objetivo de los filtros utilizados en nuestro proyecto entra en el ámbito del preprocesamiento y segmentación de imágenes.

Ejemplos de la fase de preprocesamiento son el suavizado, el realce, detección de bordes, detección de umbral, etc.

El procesamiento de una imagen puede ser visto como una transformación de una imagen en otra imagen, es decir, a partir de una imagen, se obtiene otra imagen modificada. Desde el punto de vista de visión artificial, el único

propósito del procesamiento de imágenes es conseguir mas adelante un análisis de estas mas simple y mas fiable. Por consiguiente, el procesamiento de imágenes debe facilitar la extracción de información para un posterior análisis, de manera que la escena pueda ser interpretada de alguna manera.

Por este motivo aplicamos a la imagen capturada por la webcam una serie de filtros, para simplificar la imagen, hasta el punto de eliminar la información que no nos interesa y realzar la información importante para el análisis posterior de la imagen, en este caso el reconocimiento de gestos y carteles.

La capacidad visual del robot, dependerá de los módulos de visión que estén activos. De momento solo hay módulos implementados y activos que permiten al robot recibir ordenes con gestos hechos con unos guantes o también recibir información procedente de carteles. Los carteles no solo pueden darle ordenes, si no hacerle una pregunta de la cual tenga conocimiento o hacerle realizar una operación aritmético-lógica.

Por tanto los únicos medios para comunicarse con el robot son guantes y carteles especiales, de momento.

Son especiales por su color. La razón de utilizar colores especiales ha sido crear en las imágenes capturadas, regiones de color con unos rangos de intensidades en los tres colores, mas separadas del resto de intensidades del histograma de la imagen. Así podremos aislar esta región, la del color especial. En el caso de los guantes, es la posición de los dedos la que indica la orden, es en los dedos donde esta el color especial, así que si solo nos quedamos con las regiones de este color y el resto lo despreciamos, estaremos simplificando muchísimo la imagen para un posterior análisis de esta. Lo mismo pasaría con los carteles, desechamos toda la imagen que no forme parte del cartel y dentro del cartel nos quedamos solo con la frase.

Los módulos posteriores a los filtros son módulos de análisis que deben de recibir la información lo mas clara posible, en el caso de los gestos se utiliza una red neuronal, la cual tiene que ser entrenada con imágenes muy simplificadas para que el entrenamiento tenga efecto y que las imágenes que reciba una vez entrenada, sean filtradas de la misma manera, para generar imágenes iguales que con las que fue entrenada, para poder reconocerlas. Respecto a los carteles el siguiente modulo es un OCR, muy sensible a ruidos, por tanto hay que asegurar que el filtro es efectivo, para que la salida de este no sea incoherente.

5.2. Detalles

- **Entrada**: Una estructura de datos como la que definimos en 4.3. Procedente del módulo de generación de imagenes.
- **Salida**: Misma estructura de datos. Su contenido ha cambiado debido al filtrado.
- **Descripción**: Este módulo es el preprocesamiento y la segmentación de la imagen de entrada, forma parte de la vision de bajo y medio nivel. Realiza una serie de operaciones para generar una imagen de salida binarizada, con los objetos de interes localizados y centrados, para su posterior analisis en el siguiente módulo.

5.3. Filtro de gestos

Como ya hemos dicho este filtro sirve como preprocesamiento y segmentación de la imagen. Ya que el suavizado reduce los ruidos y la extracción de regiones de color localiza los objetos de interés, con el objetivo de pasarle una información mucho mas comprensible y simplificada a la red neuronal. Los gestos al robot se realizan con la ayuda de 2 guantes, uno para la mano izquierda que dará las ordenes y otro para la derecha que indicará los parámetros de dichas ordenes.

Cada guante tiene en la punta de los dedos unos marcadores de color especial, un color que no se encuentre formando parte del entorno (colores muy llamativos con una textura que no generé brillos o sombras).

Se determinaron una conjunto de ordenes, las justas para que un objeto pueda describir cualquier trayectoria sobre una superficie plana. Concluimos que estas podrían ser únicamente: avanzar y girar. Es necesario decirle la distancia que tiene que avanzar en cada momento, pero como eso no era simple, se introdujo la orden parar, así mientras se mueve el robot tu decides cuando ha recorrido la distancia oportuna y detenerlo con una orden. También se distinguió en la orden girar, entre girar a la izquierda y girar a la derecha. Con esto tenemos 4 tipos de ordenes distintas para dar al robot. Pero aun el robot necesita mas información sobre estas ordenes, como por ejemplo en la orden de giro, con cuantos grados tiene que realizarlo o en la orden de avanzar a cuanta velocidad debe moverse. Siguiendo con el objetivo de la simplicidad en vez de añadir mas gestos diferentes a la misma mano, se utilizo la otra mano, es decir, una mano indicaría las ordenes al robot y otra los parámetros según el tipo de orden.

Los parámetros son o de velocidad o de grados de giro, la velocidad puede ser nula, medio baja, medio alta o alta y los ángulos de giro pueden ser 0º, 45º, 90º o 180º, es decir, cuatro parámetros en ambos casos, se utilizan los mismos símbolos para velocidad como para giro, por tanto solo existen 4 gestos diferentes que se puedan dar con la mano derecha para expresar los parámetros al robot.

La coincidencia del numero de gestos utilizados para ordenes y el numero de gestos utilizados para parámetros, simplificara la implementación de la red. Y el hecho de usar los mismos gestos para representar las ordenes con la mano izquierda y los parámetros con la mano derecha, simplificara también el entrenamiento de la red.

Los gestos elegidos son diferentes posiciones de los marcadores del guante, lo mas claro posible, para que después del filtrado la red no tenga problema para diferenciar unos símbolos de otros.

Los marcadores son las únicas áreas de la imagen que no se eliminaran de la imagen. Estas regiones pasarán a ser blancas y el resto negro. Por tanto repito los gestos tienen que ser los suficientemente distintos unos de otros, para que una vez filtrados, esas zonas blancas puedan diferenciarse a simple vista y saber a que orden se están refiriendo.

Es en ejecución cuando se decide a través de una ventana proporcionada por el pipeline el color de la imagen a filtrar, así que el color de los marcadores del guante no tienen porque ser fijos, se pueden determinar en cada momento. Eso si, los colores de ambos guantes deben de ser distintos y especificarse que color será el que representa a las ordenes y cual representara a los parámetros.

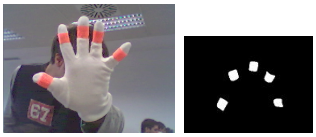


Figura 5.1: La imagen de la izquierda muestra la imagen que le llega al filtro. La de la derecha muestra la imagen una vez filtrada.

Las fases de este filtro son:

- Suavizado. Por promediado del entorno.
- Extracción de regiones por el color.
- Centrado de imagen.

5.3.1. Suavizado de la imagen

Difumina la imagen. El suavizado es una transformación de vecindad, donde el valor del nuevo píxel depende de los valores de los píxeles que le rodean. Nuestro método de filtro es un suavizado por el promediado del entorno de vecindad.

Se utiliza para la eliminación de ruidos y otros efectos debidos a la cuantización o a perturbaciones. La razón de haber utilizado un suavizado al principio fue para crear un difuminado de la imagen y que futuros filtrados sean mas uniformes.

Cuando no se aplica, la extracción de regiones posterior no es muy fiable, ya que a causa de la iluminación o de la textura del material utilizado en el color especial, hay zonas del objeto de interés que no tienen el mismo color pudiendo pasar por ejemplo de ser un rojo casi blanco a un rojo casi negro, esta amplitud de color es inadmisible para la extracción de colores, ya que esos píxeles serían considerados fuera de rango y por tanto como elementos del entorno y no como elementos de interés. Esto repercutiría en la red neuronal posterior, la cual tiene que asignar pesos según el valor de los píxeles de entrada, si no podemos determinar unos valores fiables en las imágenes de entrada no se podrá entrenar de forma fiable la red ni poder asegurar un comportamiento seguro en el futuro.

Las desventajas de este método son que desdibuja contornos y detalles de forma, pero en nuestro caso concreto esto no tiene relevancia.

$$\mathbf{g(x,\ y)} = (\tfrac{1}{K}) * \Sigma f(n, m)$$

(sumatorio de 0 a K, siendo K el numero total de puntos de la vecindad)

Método:

Se utiliza una máscara de convolución $5x5 \rightarrow wi = 1/25$

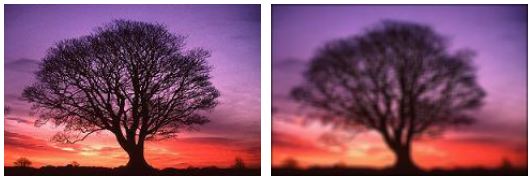


Figura 5.2: Imagen original e imagen suavizada.

5.3.2. Extracción de regiones de color

Como hemos dicho los objetos de interés son las puntas de los dedos, así que tenemos que aislarlas del resto de la fotografía. La forma es delimitar estas regiones y darlas toda la importancia respecto al resto de la imagen.

Queremos que el filtro convierta una imagen capturada por la webcam en una imagen blanca y negra, donde las puntas de los guantes quedaran en blanco y el resto de la imagen en negro. Así la red solamente será entrenada para recibir imágenes con regiones blancas y negras, si hay una sola región de un cierto tamaño implicaría que solo hemos enseñado un dedo del guante, lo que se correspondería con el gesto de avanzar.

Esto es a lo que llamamos segmentación, ya que estamos localizando los objetos de interés.

Por tanto nuestro objetivo es binarizar la imagen basándonos en el hecho de que los píxeles de una determinada región presentan una distribución de intensidad similar, por tanto, a partir del histograma de los niveles en los tres colores, determinamos cual es la zona de dicho histograma y por tanto la región de la imagen.

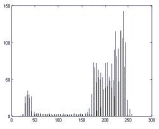


Figura 5.3: Ejemplo del histograma de intensidades de una imagen.

Basándonos en el modelo de color RGB, se pueden extraer de la imagen aquellas regiones en las que predomine una determinada componente de color.

El método consiste en elegir un determinado predicado y determinar en toda la imagen los píxeles que cumplen dicho predicado. Esos píxeles los marcamos en blanco y el resto en negro, de esta forma obtenemos una imagen binaria.

Es difícil determinar cual es el umbral optimo para poder llevar acabo una binarización adecuada. Además debemos tener en cuenta que la iluminación que habrá de unas ocasiones a otras será distinta, esto influye en la manera en que la cámara percibe los colores del entorno, por ejemplo, si hay poca luz los colores serán mas oscuros y lo contrario si hubiera mucha luz, por tanto no hemos podido determinar un umbral fijo porque este será dependiente del entorno.

La interfaz del pipeline genera para cada modulo de filtro una pequeña ventana que permite la elección de un color de las imágenes que están entrando en ese momento por la webCam. De esta manera nos estamos asegurando de seleccionar y fijar el color exacto en esas condiciones del entorno.

Debido a que las imágenes son a color, al seleccionar un color del entorno, se estarán fijando automáticamente 3 umbrales, uno para el rojo, otro para el verde y otro para el azul.

Si las tres componentes del píxel (x,y) están dentro del rango seleccionado, entonces las 3 componentes tomaran el valor blanco (255) y si estan fuera de rango tomaran el valor negro (0). Binarizando así la imagen.

Como elegir la tolerancia de este rango. Cuanto mas amplio sea el rango mas cantidad de colores entraran dentro de este. Aunque elijamos el color exacto del entorno que queramos filtrar, si no fijamos bien la tolerancia del rango, la imagen no binarizará las regiones correctas.

Figura 5.4: Extracción de una región de color de una imagen con tolerancia baja, media y alta.

5.3.3. Centrado de la imagen

Los gestos realizados con los guantes nunca son capturados por la webcam en la misma posición. Nunca estarán totalmente centrados, si no ligeramente o totalmente desplazados mas a la derecho o mas a la izquierda, arriba o abajo. Esto es de vital importancia para la red, ya que entrena y reconoce en relación a los valores de los píxeles de la imagen, si aprende que el símbolo de avanzar es una región blanca sobre un fondo negro situada siempre en el centro de la imagen, cuando este en fase de reconocimiento y la webcam capture un gesto desplazado, la red lo considerara como gesto no reconocido. Otra opción podría ser entrenar la red para que reconociese el mismo gesto en cualquier posición, pero eso no podría nunca servir en el entrenamiento, y que los pesos no terminarían nunca de fijarse, ya que el píxel (x,y) si esta blanco para unas imágenes se considerara como ejemplo de entrenamiento positivo, para otras se considerara negativo y los pesos no podrán ajustarse. Por tanto hay que intentar conseguir que las regiones de interés extraídas estén siempre situadas mas o menos en la misma zona de la imagen. Para

eso decidimos que la mejor forma de hacer esto era centrar la imagen según el centro de masas del conjunto de píxeles de interés.

Todos los píxeles cuyo color esté dentro de este rango, formaran un conjunto. El conjunto de las coordenadas cartesianas de estos píxeles dentro de la imagen.

Vamos a llamar centro de masas (c. m.), a la media de las coordenadas de todos los píxeles que forman el conjunto, de esta manera el centro de masas será la coordenada de un píxel que puede o no pertenecer al conjunto, pero que representa el centro de la mayor concentración de elementos de este. La coordenada x será la media aritmética de todas las coordenadas x de este conjunto y lo mismo con la y.

$X = \frac{\Sigma x_i}{k}$, desde i=1..k, siendo k el cardinal del conjunto y $\forall x_i \epsilon conjunto$.
$Y = \frac{\Sigma y_i}{k}$, desde i=1..k, siendo k el cardinal del conjunto y $\forall y_i \epsilon conjunto$.

El objetivo es centrar el centro de masas dentro de la imagen. Así estaremos centrando la región de interés. El centrado implica un desplazamiento de todos los píxeles de la imagen.

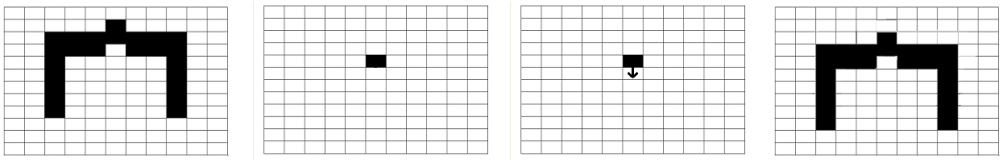


Figura 5.5: Imagen en blanco y negro. El color elegido será el negro. La 2º imagen muestra cual sería la coordenada que representa el centro de masas del conjunto de píxeles negros. La 3º cual seria el desplazamiento. Y la 4º el resultado del centrado.

Si la posición del c. m. no esta en el centro de la imagen. La distancia que hay que desplazar la imagen es el modulo entre el punto central de la imagen y el punto del centro de masas. Por tanto movemos todos los píxeles de la imagen n posiciones verticalmente y m posiciones horizontalmente para cuadrar el c. m. con el centro de la imagen. En este proceso hay píxeles de la imagen original que se pierden y otros que se crean y no tienen un valor concreto, estos serán creados con el color del entorno, ya que se supone que no son de interés.

Así siempre las regiones blancas que representan los gestos aparecen centrados en la imagen sobre un fondo negros, totalmente preparados para ser pasados a la entrada de la red neuronal.

Curiosidad del centrado

El centrado tiene otra utilidad. Debido a que necesita hacer el calculo de cuantos píxeles cumplen la propiedad de estar dentro del rango de color, si resulta que no hay en toda la imagen ninguno que la cumpla, el método no devuelve la imagen centrada, si no que devuelve NULL. Tal y como esta implementado el pipeline si un modulo saca como estructura de datos un puntero a NULL, las siguientes operaciones que se realizarían sobre esta estructura dejan de hacerse, esto aumenta la velocidad del programa si no hay gestos o carteles frente al robot, ya que disminuye en gran cantidad el numero de instrucciones realizadas. Se podría decir que el centrado es un detector de objetos de interés que mantiene al pipeline en *stand by* mientras que no se detecten objetos frente a la cámara.

5.4. Filtro de carteles

Los carteles son señales al igual que los gestos percibidos visualmente por el robot. Llevan impresos mensajes, ya sean ordenes, operaciones aritméticas o preguntas sobre datos que el robot posea en su base de datos. Este debe ser capaz de desechar toda la información de la imagen excepto este mensaje.

Es aquí donde se nos planteó la duda de como resolver este problema, como enseñar al robot a desechar todos los píxeles de la imagen excepto aquellos píxeles negros que forman parte de la región de las letras que forman el mensaje. Hubo distintas soluciones al problema, pero la que mejor ha funcionado es la de crear unos carteles de un color especial, es decir, de un color que no suela encontrarse en el entorno y el mensaje de este impreso en letras negras.

Por tanto ya existe una característica que diferencia a los píxeles del mensaje del resto, y es que son lo únicos píxeles negros rodeados en todas sus direcciones por píxeles con la intensidad propia del color especial. En resumen este filtro lo que hace es binarizar la imagen, el objetivo perseguido es parecido al del filtro de los guantes, sen intenta binarizar la imagen dejando a un color lo importante y en otro color lo que no nos interesa. La diferencia es que el procesamiento llevado a cabo en este filtro es mas complejo que en el de los guantes, requiere pasar por mas fases de procesamiento.

Así pues este filtro convertirá imágenes que contengan un cartel con un mensaje X en una imagen blanca con el mensaje negro centrado, horizontal y con el menor numero de ruidos, también asegura no filtrar la imagen si no se percibe ningún cartel o si este no esta al 100x100 dentro del campo de visión.

Todo esto son medidas de seguridad para facilitar el futuro análisis del mensaje por parte del OCR, evitando posibles fallos de este y que sea lo mas fiable posible.

Estos son ejemplos del filtro. Las imágenes sin cartel no serán procesadas ni pasadas al OCR, para evitar fallos y operaciones innecesarias.

Figura 5.6: 1º Simulación de la captura de un cartel. 2º resultado de filtrar la imagen.

Ejemplos de mensajes dentro de un cartel:

- Ordenes y parámetros: Avanzar MediaAlta, Parar, Girar 90, ...
- Operaciones aritmético-lógicas: (150mod15)/5, (true and false), ...
- Preguntas a su base de datos: Nombres creadores, ...

Las fases de este filtro serían estas:

if(not BORDES and CENTRAR and ESQUINAS)
ROTAR
ESQUINAS
EXTACCION DE REGIONES
OPERACIÓN DE CIERRE
end if;

5.4.1. Bordes

En este proyecto hay un factor que siempre tenemos en contra y es el tiempo. Hay que recordar que la captura, procesamiento y análisis de las imágenes se hacen en tiempo real, si la captura de imágenes se hace cada X milisegundos hay que asegurarse que todas las operaciones que se deban de realizar tardan menos que ese intervalo. Por tanto los filtros no solo tienen el deber de procesar la imagen si no también de mejorar el rendimiento del programa.

Por ello son utilizados como sensores de detección de ciertos objetos, en este caso de carteles.

Esta fase es una función que recibe una imagen de entrada y detecta si la imagen esta completamente dentro del campo de visión o solamente en parte, si solo ha sido capturado parte del cartel, el mensaje puede que no haya salido entero por tanto carecería de sentido y no seria valido. Si el cartel ha sido capturado en su 100x100 devuelve cierto si no falso. Al devolver falso el resto de operaciones de filtro sobre el cartel se dejan de hacer y se pasa NULL al modulo de OCR para que tampoco realice ninguna operación, mejorando por tanto la eficiencia del programa, ya que solamente realizara operaciones cuando detecte carteles enteros.

La idea para implementar este función es simple, solo hay que buscar píxeles cuya intensidad este dentro del rango de color especial del cartel si existe alguna región de estos píxeles en el borde de la imagen capturada implica que el cartel no ha sido capturado en su totalidad.

5.4.2. Centrar

Esto significa centrar el cartel dentro de la imagen. La imagen quedara desplazada lo necesario para que el cartel se encuentre justo en el medio y por tanto también lo estará el mensaje. Esto no aporta valor añadido al posterior análisis por parte del OCR, sirve para asegurar la integridad del mensaje en las siguientes fases del filtro, además sirve como al igual que en el filtro de los gesto y como la función anterior de BORDES, como un sensor. El anterior detectaba si el cartel estaba entero y este si existe cartel dentro de la imagen.

El centrado en su implementación cuenta el numero de píxeles de color especial, es decir, del cartel. Haciendo una media aritmética de sus posiciones, por esto si se da el caso en que el numero de píxeles de este color detectados es igual a cero implica que no hay cartel y por tanto que las siguientes operaciones que se realicen sobre la imagen no tiene sentido al igual que la ejecución del OCR, por tanto como la función BORDES, CENTRAR devuelve NULL si no hay cartel en la imagen, acelerando la ejecución del programa y si resulta que hay cartel, entonces devuelve la imagen centrada.

El método utilizado es el mismo que en centrado del filtro de gestos.

5.4.3. Esquinas

Esta función hace un barrido de la imagen sabiendo ya que existe un cartel en ella, con el objetivo de encontrar la cuatro esquinas de este. Las esquinas son coordenadas cartesianas, cuyo conocimiento es de gran utilidad

para realizar una posible rotación del cartel, para la extracción de regiones y como sensor.

- Respecto a la rotación:

  Con saber la posición de al menos dos esquinas contiguas de las cuatro del cartel podemos conocer cuantos grados esta inclinado el cartel, dentro de la imagen. Ya que dos puntos forman un vector, solo hay que calcular el ángulo que forma este vector con algún eje de coordenadas y sabremos cuanto esta inclinado el cartel, para su posterior rotación.

- Respecto a la extracción de regiones:

  Conocer las esquinas es conocer los limites de lo que nos interesa y de lo que no, ya sabemos que todo lo que este fuera de las esquinas es desechable y lo que queda dentro es necesario procesarlo.

- Respecto al sensor:

  Conocer al menos tres esquinas, es conocer tres coordenadas y por tanto eso nos permite crear dos vectores. Un vector es un lateral del cartel y el otro la base, sabiendo esto si el ángulo que existe entre esos vectores es un ángulo recto, significa que lo que se esta detectando es un cartel, si no cumple esta propiedad es que se ha detectado un objeto del color buscado, pero no es un cartel. Esto evita pasarle al OCR posible información sin sentido que podría generar fallos en el programa principal.

5.4.4. Rotar

Como ya hemos dicho el análisis siguiente a este procesamiento es llevado a cabo por el OCR, este es un modulo implementado de tal forma que no es sensible al tamaño y en cierta medida al formato de los caracteres, pero si que es sensible cuando estos se encuentran rotados. Muchas veces el robot detectara y procesara carteles que no están completamente horizontales, lo que provocaría fallos en el OCR, por tanto es necesario rotar la imagen lo necesario para que el cartel quede horizontal. Con la función anterior de las esquinas ya sabemos cuantos grados esta girado el cartel, solo hay que pasarle a esta función como parámetros la imagen y los grados a girar.

Esta transformación implica un cambio en la disposición y distribución de los píxeles respecto de un sistema de coordenadas.

Para estas tres transformaciones elementales, es necesario hacer una interpolación a la imagen resultante. A efectos de notación, la imagen original

tendrá coordenadas (i, j) y la imagen resultante coordenadas (x, y). Estas coordenadas están asociadas a un determinado sistema de referencia, que es preciso establecer con su origen y su convenio de ejes.

Vamos ha utilizar coordenadas homogéneas ya que permiten realizar la rotación mediante el uso de matrices.

La transformación de rotación consiste en que para cada píxel de la imagen original se calcula su correspondiente píxel de la imagen destino. Una vez sabemos la posición del píxel le asignamos a este la misma intensidad que el de la original. Este sistema puede producir una imagen destino rotada con muchos píxeles en blanco, es decir, píxeles que no se asocian con ninguno de la original siguiendo la matriz de transformación vista. La única forma de asignar un valor de intensidad a todos los píxeles de la imagen destino, es hacer la transformación inversa, donde en vez de hallar el píxel destino (x, y) a partir de un píxel (i, j), hallamos el píxel (i, j) a partir del (x, y). Ahora podemos recorrer toda la imagen destino viendo cual es el píxel origen asociado a él y asignarle así su intensidad, sin que queden píxeles sin color asignado. Ahora puede que varios píxeles destino compartan el mismo píxel origen, pero eso es menos grave que ver una imagen rotada con casi un tercio de espacios en blanco.

La rotación en sentido opuesto viene dada por la siguiente transformación:
$i = x.cos\theta - y.sen\theta$
$j = -x.sen\theta + y.cos\theta$
(i, j) es un píxel de la imagen origen y (x, y) uno de la imagen destino.

Los píxeles de la imagen resultante que tras el calculo se les asigna un píxel de la imagen original que se sale de rango, se le asigna una intensidad correspondiente con la del entorno. En el ejemplo anterior se les asigno el color negro.

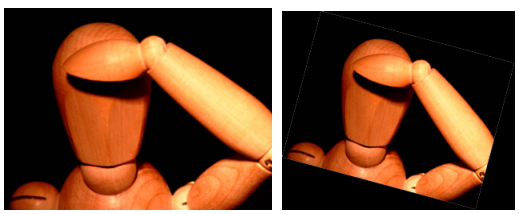


Figura 5.7: Ejemplo del funcionamiento del algoritmo de rotación.

5.4.5. Extracción de regiones

Esta parte es similar a la utilizada en el filtro de gestos. Solo que aquí no nos interesan las regiones de color especial, si no ciertas regiones rodeadas de este color especial, que en este caso serán los caracteres del mensaje.

La extracción de regiones binariza la imagen convirtiendo a un color la región buscada y a otro el resto de la imagen, las regiones del mensaje pasaran a negro y el resto a blanco, como en el ejemplo anterior.

La función ESQUINAS se realizara otra vez después de la rotación (si esta ha sido necesaria) ya que la posición de estas dentro de la imagen habrá cambiado. Como ya hemos dicho antes, saber las coordenadas de estas esquinas nos informa de que área de la imagen es necesario filtrar y cual directamente es considerada como región no extraíble y puesta directamente a color blanco, en este caso será toda aquella región que quede fuera del cartel.

La extracción de regiones propiamente dicha se realizara por tanto dentro del área de la imagen perteneciente al interior del cartel. Las regiones que se extraerán serán aquellos píxeles que cumplan la propiedad de ser oscuros rodeados en todas sus direcciones por color especial.

5.4.6. Operacion morfológica

Después de todas las operaciones anteriores realizadas sobre la imagen los caracteres del cartel puede que hayan quedado dañados, es decir, como si se hubieran erosionado, perdiendo suavidad en sus bordes o incluso dejar un mismo carácter separado en regiones distintas.

Nuestro cerebro tiene la necesidad de encontrar sentido a lo que ve y a unificar figuras inacabadas, pero esta cualidad no la tiene el OCR, si este detecta regiones separadas las tratara como distintos caracteres y buscara el carácter mas aproximado a estas, lo cual sería un error. Es necesario por tanto juntar regiones que hayan quedado separadas y rellenar huecos de los caracteres del mensaje para asegurar por tanto una mejor comprensión del mensaje.

La operación morfológica realizada sobre la imagen es una operación de cierre para rellenar los caracteres.

Las operaciones morfológicas simplifican las imágenes y preservan las formas principales de los objetos. La morfología puede utilizarse para suavizar bordes de una región, separar regiones que el proceso de segmentación presenta unidas o como en nuestro caso unir regiones que fueron separadas durante la segmentación.

Una transformación morfológica θ viene dada por una relación de la imagen (conjunto de puntos X) con otro pequeño conjunto de puntos B, llamado

elemento estructural. B se expresa con respecto a un origen local O (llamado punto representativo o punto de vista director).

La transformación morfológica $\theta(X)$ aplicada a la imagen X significa que el elemento estructural B se desplaza por toda la imagen. Suponiendo que B se posiciona sobre algún punto de la imagen, el píxel de la imagen correspondiente al punto representativo O de B se denomina píxel actual. El resultado de la relación entre la imagen X y el elemento estructural B en la posición actual se almacena en el píxel actual de la imagen.

Dilatación

La dilatación es una operación isotrópica ya que se comporta igual en todas las direcciones y expansiona el objeto un píxel. Esta operación en ocasiones se denomina rellenado o crecimiento. La dilatación con un elemento estructural 3x3 isótropo puede interpretarse como una dilatación que cambia todos los píxeles de fondo que son vecinos al objeto. Gráficamente la dilatación se realiza como sigue: se va recorriendo la imagen por ejemplo de izquierda a derecha y de arriba abajo y, donde nos encontremos un 1, situamos el origen del elemento estructural; en esa posición se realiza la unión del elemento estructural con la parte de la imagen sobre la que se solapa dicho elemento, marcando todos los píxeles de esta parte de la imagen con los valores del elemento B.

Erosión

La erosión re realiza de forma parecida a la dilatación solo que cuando recorriendo la imagen original, nos encontramos un píxel con valor 1, aplicamos el elemento estructural no para asignarle sus valores a la imagen destino, si no como comparación. Si todos los píxeles del elemento B coinciden con los valores de los píxeles de la región de la imagen entonces ponemos el valor de píxel de la imagen destino con valor 1, si no con valor 0. Se puede apreciar la desaparición de muchos contornos existentes en la imagen original. La erosión es denominada como reducción, utilizada para simplificar la estructura de los objetos, ya que los objetos con anchos pequeños desaparecen , por tanto, objetos complicados pueden descomponerse en otros mas simples.

Apertura y Cierre

LA erosión y la dilatación son transformaciones no invertibles. Si una imagen es erosionada y luego dilatada, la imagen original no se recupera. En efecto, el resultado es una imagen mas simplificada y menos detallada

Figura 5.8: Original. Binarizada segun un umbral. Y ejemplo de cierre sobre esa imagen.

que la original. La erosión seguida de una dilatación crea una transformación morfológica denominada apertura. La dilatación seguida de una erosión crea una transformación llamada cierre. La apertura y el cierre con un elemento estructural isótropo se utiliza para eliminar detalles específicos de la imagen mas pequeños que el elemento estructural. La forma global de los objetos no se distorsiona. El cierre conecta objetos que están próximos entre sí, y rellena pequeños huecos y suaviza el contorno del objeto rellenando pequeños valles, mientras que la apertura produce el efecto contrario. Los conceptos de pequeño y próximo están relacionados con la forma del elemento estructural.

5.5. Código

Ver HTML: documentación del módulo en filtro_gestos.c.

Capítulo 6

Módulo de red neuronal

6.1. Introducción

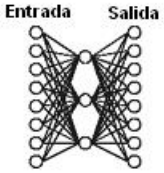
Las Redes Neuronales Artificiales (ANNs de Artificial Neural Networks) fueron originalmente una simulación abstracta de los sistemas nerviosos biológicos, formados por un conjunto de unidades llamadas neuronas o nodos conectadas unas con otras. Estas conexiones tienen una gran semejanza con las dendritas y los axones en los sistemas nerviosos biológicos.

Una ANN es un gran numero de conmutadores interconectados, donde a las conexiones se les asigna un peso. Este peso será fijado durante la fase de aprendizaje. El procesamiento de información es paralelo distribuido.

En que tipos de problemas se utilizan:

- En problemas donde los ejemplos se describen con un gran numero de atributos.
- Puede haber ruido en los datos.
- Cuando no se conoce la forma de la función objetivo. Problemas que no tienen un algoritmo especifico para su solución, o cuyo algoritmo es demasiado complejo para ser encontrado.
- Cuando es permisible un tiempo de entrenamiento largo.
- Ejemplos: Reconocimiento del habla, clasificación de imágenes, predicciones financieras, etc.

Dicho esto la elección de una red neuronal como medio de resolución del problema de reconocimiento de patrones sería en este caso la más acertada. Es



Entrada		Valores Ocultos				Salida
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

Figura 6.1: Ejemplo esquemático de una ANN.

muy útil cuando no se conoce la función objetivo y se estima que los datos de entrada llegan con cierto porcentaje de ruido. La decisión se tomó meses antes de empezar el proyecto. Para asegurarnos de que era viable implementamos la red dentro de un pequeño programa, que a partir de imágenes de entrada devolvía cierto si detectaba una imagen de una persona con una guante blanco y falso si no había guante.

6.2. Detalles

- **Entrada**: Una estructura de datos como la que definimos en 4.3 con la imagen filtrada para ordenes o parametros. Procedente del módulo de filtro.

- **Salida**: Una cadena de texto que contiene el parametro o la orden que debe realizar el robot.

- **Descripción**: Encargado de la descripción y reconocimiento de patrones, forma la visión de medio y alto nivel. Red multicapa de sigmoides con pesos ajustados por entrenamiento. Cada neurona de la capa de entrada recibe un pixel de la imagen, por retropropagación de esta entrada calcula los valores de las neuronas de salida. Cada una de estas neuronas lleva asociada una cadena de texto, como la retropropagación solo activará una neurona de salida, será su cadena asociada, la salida del modulo. (La descripción de patrones se encuentra en un archivo de pesos).

6.3. Descripción técnica

Hemos utilizado una red multicapa, ya que permiten representar superficies de decisión no lineales. Debido a esto no se puede utilizar unidades lineales ya que solo permitirían representar funciones lineales. Tampoco pudimos utilizar perceptrones porque su función de salida es discontinua, no derivable y por lo tanto no se le puede aplicar el descenso del gradiente. Necesitábamos una unidad que diese como salida una función no lineal y que fuese derivable con respecto a las entradas.

Por eso utilizamos el sigmoide como unidad.

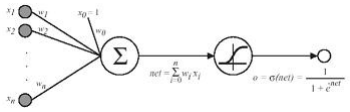


Figura 6.2: Unidad Sigmoide.

La función sigma $\sigma(x) = \frac{1}{(1+e^{-x})}$ es no lineal y derivable $\frac{d\sigma(x)}{dx} = \sigma(x).(1 - \sigma(x))$.

El descenso de gradiente se puede utilizar para entrenar:

- Una unidad sigmoide.
- Una red multicapa de unidades sigmoides. Retropropagación.

La retropropagación permite aprender los pesos para una red multicapa con un numero de unidades e interconexiones dado. Consideramos una red con múltiples unidades de salida, de forma que el error es la suma de los errores sobre todas las salidas. El espacio de hipótesis viene dado por los valores posibles para los pesos de todas las unidades de red. Utilizamos el descenso de gradiente para encontrar una hipótesis que minimice el error, el descenso utilizado es incremental, es decir, los pesos se actualizan después de considerar cada ejemplo de entrenamiento en lugar de esperar a considerarlos todos. Es mas improbable que el descenso incremental caiga en un mínimo local.

La secuencia de pasos utilizada en nuestro entrenamiento viene a ser esta:

- Crear la red.
- Se inicializan los pesos de la red con valores pequeños y aleatorios entre -0.05 y 0.05

- Para una media de 30 iteraciones se realiza lo siguiente: De una lista de imágenes de entrenamiento se va cogiendo una a una y se cargan en la capa de entrada de la red, para esa imagen se calculan las capas, luego según un objetivo se calculo el error cometido en la capa oculta y en la salida y respecto a estos errores se reajustan los pesos de las interconexiones.

- Se realiza un prueba y una validación con imágenes distintas a las del entrenamiento, para ver el porcentaje de error total cometido, si es aceptable se guarda en un archivo los pesos de la red entrenada, para que en la fase de reconocimiento de imágenes solo haya que realizar el calculo de capas.

Para cada unidad de salida k, se calcula su termino de error: $\delta_k = O_k.(1 - O_k).(t_k - O_k)$

Para cada unidad oculta h, se calcula su termino de error: $\delta_h = O_h.(1 - O_h).\Sigma(W_{kh}.\delta_k)$, siendo k las salidas.

Se actualiza cada peso de la red $W_{ji} = W_{ji} + \Delta(W_{ji})$ donde $\Delta(w_{ji}) = \eta.\delta_j.x_{ji} + \alpha.\Delta w_{ji}.(n-1)$.

(t_k es la salida dada por ejemplo de entrenamiento para la unidad k. o_t es la salida generada por la red.)

La actualización de los pesos en la iteración n depende de la actualización en n-1. El 2º termino de la ecuación representa la cantidad de movimiento. Un símil físico seria una pelota que cae por la superficie de error, la cantidad de movimiento hace que la pelota tienda a mantener la misma dirección, con esto se intenta evitar que la pelota pare en un mínimo local o que se pare en un llano.

Respecto el descenso del gradiente en el sigmoide: Para el descenso incremental consideramos el cambio en los pesos inducido por cada ejemplo de entrenamiento $\Delta w_{ji} = \eta \frac{\eta E_d}{\eta w_{ji}}$ donde el error sobre un ejemplo de entrenamiento viene dado por la suma de los errores en cada unidad de salida $E_d(w) \equiv \frac{1}{2}.\Sigma(t_k - o_k)^2$, siendo k las salidas

La salida viene dada por $net = \Sigma w_i.x_i$, i=0..n y $o = \sigma(net) = \frac{1}{1+e^{-net}}$, aplicando la regla de la cadena $\frac{\eta E_d}{\eta w_{ji}} = \frac{\eta E_d}{\eta net_j}.\frac{\eta net_j}{\eta w_{ji}} = \frac{\eta E_d}{\eta net_j}.x_{ji}$. Para calcular $\frac{\eta E_d}{\eta net_j}$ distinguimos el caso de las unidades de salida y las ocultas.

Error en las unidades de salida $\frac{\eta E_d}{\eta net_j} = -(t_j - o_j).o_j.(1 - o_j)$. Error en la unidades ocultas $\frac{\eta E_d}{\eta net_j} = o_j.(1 - o_j).\Sigma(-\delta_k.w_{kj})$.

Para valores pequeños de los pesos (al principio del proceso) la red presenta una función casi lineal donde es menos probable encontrar mínimos

locales. Cuando la función es mas compleja (un punto mas avanzado del proceso) es de esperar que nos hayamos acercado tanto al mínimo global que los mínimo locales sean aceptables. Para garantizar que alcanzamos el mínimo global utilizamos heurísticas:

- Añadiendo cantidad de movimiento.
- Utilizando descenso incremental.
- Entrenar distintas redes con los mismos ejemplos, pero con distintos valores iniciales en los pesos.

La capacidad expresiva de este tipo de red es bastante alta ya que:

- Cualquier función booleana se puede representar con una red de dos capas.
- Cualquier función continua se puede aproximar con un error arbitrariamente pequeño por una red de dos capas.
- Cualquier función se puede aproximar con un error arbitrariamente pequeño por una red de tres capas (las dos ocultas de sigmoides y la de salida de unidades lineales).

6.4. Diseño

Esta formada por una capa de entrada, una de salida y una sola capa oculta. Si imaginásemos la red como una caja negra, esta tendría que recibir como entrada una imagen y sacar como salida una cadena de texto explicativa de algún atributo de esa imagen.

En nuestro caso la entrada serán siempre imágenes del mismo tamaño 320x240 píxeles, por ello la capa de entrada consta de 76800 unidades. En realidad la entrada es un unsigned char* que representa la imagen con valores de 255 o 0, es decir, blanco o negro, recuerdo que las imágenes que le llegan a la red son imágenes que previamente han pasado por el modulo de filtro a si que llegan ya binarizadas. Estas entradas serán normalizadas entre 1 y 0, para que las entradas estén en el mismo rango que las unidades de la capa oculta y de salida.

La capa oculta debe tener tan pocas unidades como sea posible. Medidas experimentales demuestran que el hecho de aumentar el número de unidades ocultas proporciona mejoras poco significativas en la precisión, pero requieren

mucho mas tiempo de entrenamiento. Nosotros hemos optado por utilizar 15 unidades.

Como ya sabemos al robot se le controla con 2 manos, los gestos de la mano izquierda le indican las ordenes y los de las derecha los parámetros, con el objetivo de simplificar el diseño los gestos de ordenes y de parámetros son los mismos, pero significan cosas distintas. Tanto para ordenes como para parámetros hay 5 tipos de gestos. Como recordatorio las ordenes eran: parar, avanzar, girar a la izquierda y girar a la derecha. Y los parámetros eran nula, medio baja, medio alta, alta si la orden actual es la de avanzar, donde los parámetros indican la velocidad a la que debe hacerlo o 0º, 45º, 90º, 180º si la orden actual es una de giro. La 5º gesto tanto para ordenes como para parámetros es el "gesto no reconocido". Por tanto dado que hay 5 tipos de gestos ha reconocer, la red tendrá que sacar 5 posibles salidas. Al principio optamos por una capa de salida de una sola unidad. El valor oscila entre 0 y 1 así que por ejemplo si esta unidad valía 0.2 significaba que había reconocido el 2º gesto, si valía 0.8 había reconocido el 4º gesto. Luego se cambio al diseño actual que es una capa de salida de 5 unidades, esto hace a la red mucho mas fiable, se podía decir que la salida de la red antes era analógica y ahora es digital, ya que todas las salidas tendrán valores menores de 0.5 excepto una que será mayor, solo hay que asociar la unidad de la salida que se ha puesto en alta con una cadena de texto. Esta asociación se hace mediante un script en Lua, así es más modificable ya que si se quiere cambiar el texto de salida no hay que recompilar el proyecto, solo cambiar un archivo de texto.

La organización de red por capas es estándar, la salida de cada unidad alimenta a todas las unidades de la siguiente capa.

La tasa de aprendizaje utilizada ha sido 0.3. La mas alta posible para reducir el tiempo de aprendizaje sin disminuir la precisión.

El descenso de gradiente es incremental para reducir el riesgo de quedarnos en mínimos locales.

Los pesos de la unidades de salida y oculta son inicializados con pequeños valores aleatorios entre 0.05 y -0.05.

6.5. Entrenamiento

La mayor parte del código utilizado en la red esta dirigido al entrenamiento. Por eso decidimos hacer un programa aparte que contiene el código de entrenamiento de la red y luego el código que está presente en el proyecto que solo contiene el necesario para crear una red, calcular los valores de las

capas a partir del valor de la capa de entrada y generar la cadena de texto de salida. Así el código del proyecto queda mas sencillo para leer.

El proceso de entrenamiento empieza con la sesión fotográfica, es necesario hacer mas de un centenar de fotos para obtener un entrenamiento medianamente fiable. Nosotros para el entrenamiento de ordenes sacamos 185 fotos, consiste en sacar fotos dándole ordenes al robot correctas, erróneas o simplemente no dándoselas. Todas estas fotos han de ser filtradas del mismo modo que lo haría el modulo de filtro del proyecto, la razón de hacer un filtrado previo es poder pasar a la red imágenes muy simples, también deben de ser tomadas en unas condiciones de iluminación similares a las que tendrá el entorno por el que circule el robot. No es lo mismo hacer aprender a la red a reconocer un gesto perdido en un mar de píxeles de miles de colores a reconocer un conjunto de píxeles blancos centrados sobre un fondo negro. Los objetivos mas perseguidos en este proyecto es la eficiencia y en este caso la fiabilidad en el reconocimiento.

Las fotos son nombradas con un formato determinado, por ejemplo, "_orden_parada_21.bmp" esto significa que la foto contiene la orden de parada y "_no_gesto_51.bmp" indica que la foto no representa ninguna orden para el robot. Este formato es utilizado en el entrenamiento para que la red sepa ir reajustando los pesos según el nombre explicativo de la foto.

Figura 6.3: _no_gesto_36.png

Todas las fotos no son utilizadas para el entrenamiento. Se hacen 3 listas de fotos que se utilizaran para el entrenamiento, la prueba y la validación. Estas 2 ultimas sirven para comprobar el buen funcionamiento de la red entrenada.

El objetivo del programa de entrenamiento es crear una red, entrenarla y salvar la estructura y pesos de red en un archivo.

El entrenamiento consiste en :

- Recorrer la lista de imágenes de entrenamiento una por una.
- Cargar la imagen en la imagen en la capa de entrada, cada valor de píxel se asocia a una unidad de la capa.

- Según el nombre de la foto, ejemplo "_orden_parada_21.bmp", se cambia el objetivo, esto sirve para calcular el error cometido.
- Cambiado el objetivo, se calcula el valor de la capas respecto a la capa de entrada, se calcula el error cometido en las capas oculta y salida y se reajustan los pesos, para disminuir el error.
- Esta lista es recorrida un numero finito de iteraciones. Las condiciones de parada pueden ser varias. La nuestra es simplemente un numero concreto, en este caso fueron 30 iteraciones. Por tanto los pesos fueron ajustados 30x(numero de fotos de la lista) veces.

Ya tendríamos así unos pesos que representan una aproximación a la función buscada.

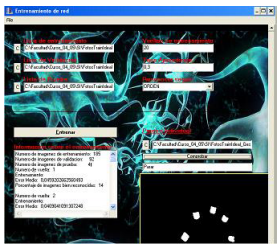


Figura 6.4: Captura de nuestro programa de entrenamiento de redes.

Estos fueron datos de un entrenamiento de la red:

- Datos entrada:
  - 148 fotos de entrenamiento, 49 fotos de validación y 30 de prueba.
  - Índice de aprendizaje: 0.3
  - 20 iteraciones
- Datos salida:
  - Porcentaje de aciertos en entrenamiento: 89 Error medio: 0,0141046521582562
  - Porcentaje de aciertos en validación: 93 Error medio: 0,00799933215976971
  - Porcentaje de aciertos en prueba: 100 Error medio: 0,00645778571591585

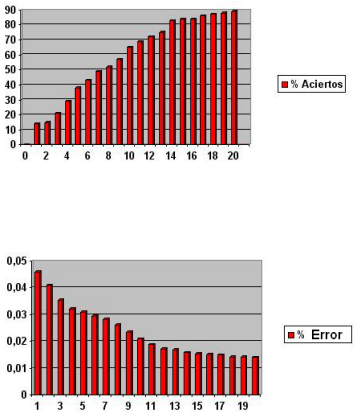


Figura 6.5: La imagen de la izquierda muestra como aumenta el aprendizaje cuanto mas se enseña a la red. Aumentando el nº de aciertos. Y la de la derecha muestra como a medida que aprende comete menos errores reconociendo figuras.

6.6. Red Neuronal en el proyecto

Como cada modulo del pipeline, el modulo de red tiene un pequeño numero de funciones fijas utilizadas para ser llamadas desde el pipeline. Tres de ellas son "red_iniciar", "red_cerrar" y "red_ciclo". Iniciar crea la red y carga el archivo creado por el programa de entrenamiento, por ejemplo el "orden_net". La función cerrar libera toda la memoria. Y la función ciclo lo único que hace es recibir un char* que representa la imagen, cargar estos valores normalizados en la capa de entrada y calcular el valor de las capas oculta y de salida según los pesos, que solamente tarda aproximadamente 0.10 segundos. Luego como ya dijimos solamente una de las cinco unidades de la capa de salida tendrá un valor superior a 0.5, esto es equivalente a que se ha puesto en ALTA y el modulo sacará como salida la cadena de texto asociada a esa unidad. Cadena modificable desde un script junto con el nombre del archivo de la red entrenada. Todo lo que sea modificable en un futuro por posibles mejoras son parámetros que van escritos en scripts.

Estas son 5 imágenes filtradas de ejemplo, cada una representa una orden o un parámetro.

Recordatorio:

- 1 dedo: Orden: Avanzar, Parámetro: Medio baja o 45º

- 2 dedos: Orden: Girar derecha, Parámetro: Medio alta o 90º
- 3 dedos: Orden: Girar Izquierda, Parámetro: Alta o 180º
- 5 dedos: Orden: Parar , Parámetro: Nula o 0º

Figura 6.6: Ejemplos de imagenes de entrada en la red.

6.7. Evolución

El primer sistema utilizado para resolver el problema de reconocimiento de gestos, fue la implementación de dos redes distintas una para ordenes y otra para parámetros, debido a que su estructura era distinta.

Se fueron modificando los filtros con el objetivo de facilitar el aprendizaje a la red.

La segunda elección fue utilizar el mismo numero de gestos tanto en ordenes como en parámetros así se pudo conseguir la misma implementación para ambas.

Luego se redujo considerablemente el código, eliminando la parte de entrenamiento del proyecto. El código de entrenamiento pasaría a ser un programa a parte que generará archivos de redes entrenadas. En este punto había 2 archivos uno para cada red.

Y por ultimo visto que los gestos de las ordenes eran reconocidos con mucha mas facilidad que los asignados a los parámetros, que fallaban constantemente, decidimos que los gestos de los parámetros fuesen iguales a los de las ordenes, solo que en vez de hacer gestos con la izquierda se hacen con la derecha. De esta manera ambas redes cargan el mismo archivo y son igual de fiables. Solo se diferencian por la cadena de texto que devuelven, pero eso va por scripts.

6.8. Código

Ver HTML: documentación del módulo en red.c y en red_neuronal.c.

Capítulo 7

Módulo de OCR

7.1. Introducción

Reconocimiento Óptico de Caracteres, Optical Character Recognition.

Es el modulo encargado de deducir el mensaje que pueda haber en una imagen en formato bmp y convertirlo en un mensaje formado por caracteres ASCII.

Como entrada recibiría una imagen de este estilo: Y como salida de-

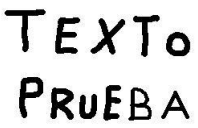


Figura 7.1: Imagen recibida desde modulo filtro.

volvería la cadena de texto: “texto prueba”.

La cámara, cada intervalo de tiempo hace capturas de su entorno, si dentro de este existe algún cartel, la imagen será filtrada y se pasara al modulo OCR para que la analice, la salida de este modulo que como ya hemos dicho es una cadena de texto que se pasará a un modulo DCG Gramática de Cláusulas Definidas la cual realizará sobre el mensaje un análisis sintáctico-semántico, después el robot generará la respuesta correspondiente. En caso de que sea una orden u operación aritmético-lógica la realizara y en caso de que se una pregunta y conozca la respuesta, la responderá.

El lenguaje natural es una forma de comunicación imprecisa y ambigua que se apoya en el conocimiento compartido por los que se comunican. Además el lenguaje natural está en continua expansión y permite expresar una misma idea de muchas formas. Gran dificultad debido a que el lenguaje es algo vivo: expansión, modificación, etc. Con ambigüedad léxica, sintáctica, semántica y referencial. Es por lo que el tratamiento del lenguaje natural suele requerir de 4 fases: Un análisis morfo-léxico, otro sintáctico, otro semántico y otro pragmático. El primero esta implícito dentro de este modulo, los dos siguientes se encuentran dentro del modulo de la DCG.

Es decir, no solo se analiza la imagen para sacar el mensaje del cartel, si no que también se hace un análisis del mensaje.

Como vemos en el ejemplo, la imagen esta binarizada formada por regiones negras que son los caracteres a reconocer, es decir, a asociarles un carácter ASCII, y por regiones blancas que forman el fondo. Son las regiones negras por tanto las áreas de la imagen en las que nos debemos centrar.

Se puede decir que el análisis no se realiza sobre la imagen global si no sobre un conjunto de pequeñas subimagenes de esta, donde se encuentran los caracteres. Para cada subimagen se hace un reconocimiento de patrones. La región negra de la subimagen se puede describir según su frontera y la distancia relativa desde de puntos concretos de esta hasta puntos de los limites de la subimagen, esta definición es un patrón. Gracias a una base de datos que guarda los patrones de cada carácter y su valor ASCII, se puede establecer una asociación entre una región con un valor ASCII gracias a una aproximación de patrones, con aproximación me refiero a que se elige el que mas se acerque con la descripción de su frontera a algún patrón de la base de datos.

Como resultado el OCR nos da una cadena de caracteres que a veces tiene caracteres erróneos debido al ruido del filtrado u otras razones. La cuestión es que existe la posibilidad de que las palabras que salgan de este modulo no existan y carezcan de sentido, por eso se realiza posteriormente al reconocimiento de patrones una corrección ortográfica llevada a cabo por un analizador morfo-léxico.

7.2. Detalles

- **Entrada**: Una estructura de datos como la que definimos en 4.3 con la imagen filtrada para carteles. Procedente del módulo de filtro.
- **Salida**: Una cadena de texto que representa el mensaje contenido en la imagen de entrada. (Una orden, una operación aritmético-lógica o una

pregunta.)

- **Descripción**: Encargado de la descripción y reconocimiento de patrones, forma la visión de medio y alto nivel. Utiliza un método propio de descripción de fronteras de los objetos. Se basa en discriminar al objeto aprovechando las relaciones geométricas inherentes a la forma del objeto(caracter). No importa el tamaño, posición u orientación de los objetos. (La descripción de patrones se encuentra en un archivo de vectores).

7.3. Diseño

La actividad del OCR pasa por 3 fases:

1. Enmarcación de los caracteres de la imagen.
2. Reconocimiento de patrones.
3. Análisis morfo-léxico.

7.3.1. Enmarcación de los caracteres

El análisis de la imagen se podría decir que sigue un esquema parecido al método divide y vencerás, ya que no intenta estudiar toda la imagen a la vez devolviendo la cadena de caracteres que contiene, si no que divide el problema de complejidad muy alta en subproblemas mas pequeños de igual tamaño y que no se solapen entre si, es decir, dividen la imagen en pequeñas subimagenes mas simples.

El análisis se realizara para cada una de las subimagenes. Estas ocupan un cierto área dentro de la imagen global, para ello hay que calcular en que región se encuentra cada carácter y almacenar este área, para su posterior análisis como una imagen independiente, imagen que solo contendrá un carácter en negro sobre un fondo blanco. Posteriormente a cada subimagen se le asociará el carácter ASCII que mas se aproxime a la morfología de las fronteras que forman su región negra.

Una vez hemos resuelto los subproblemas y tenemos los caracteres, solo queda unirlos formando palabras y estas formando frases, para ello será necesario hacer un calculo de las distancias entre las áreas, que nos darán la información de si existen espacios en blanco entre estos caracteres o no.

El método de enmarcado es muy sencillo. Se realiza un barrido de arriba a abajo explorando todos los píxeles de cada fila como un solo bloque. Vamos a llamar fila negra aquella que contenga uno o mas píxeles negros, es

Figura 7.2: Ejemplo de una imagen de entrada al módulo.

decir, existen caracteres que intersecan con la fila, y llamaremos fila blanca a aquella cuyos píxeles son todos blancos, es decir, que no hay caracteres que intersequen a esta. Como muestra la fotografía, las imágenes ya llegan binarizadas donde los caracteres son negros y el fondo es blanco, por eso si hacemos un barrido de arriba hacia abajo iríamos encontrando al principio filas blancas hasta encontrar una primera fila negra (el caso de que no haya ninguna fila negra en toda la imagen no se puede dar ya que el filtro cuando no hay caracteres, no pasa la imagen al OCR y este no hace nada) entonces sabríamos que esta es la coordenada Y de la parte superior de la primera línea de caracteres. Debido a que el mensaje fue rotado en el filtro para que fuese horizontal, lo suyo es que a partir de ahora todas las líneas sean negras hasta encontrar una blanca que representara la coordenada Y de la parte inferior de la primera línea de caracteres, por tanto ya tendríamos delimitados los puntos superior e inferior de los caracteres de la primera línea, continuamos con el mismo método para el resto de la imagen. Vamos almacenando las coordenadas Y que nos vamos encontrando superior e inferior para cada línea.

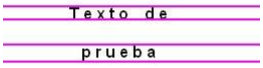


Figura 7.3: 1º delimitar las lineas del texto.

Ahora nos queda por saber cuales son las fronteras laterales para cada carácter. El esquema anterior, para hallar los laterales no es útil cuando el mensaje ocupa mas de una línea, ya que en una línea puede haber un carácter en la posición X donde en otra línea en esa posición X hay un

espacio, por tanto consideraría que en las dos líneas hay un carácter y en futuro se estudiara una región que contiene un espacio creyendo que dentro hay un carácter.

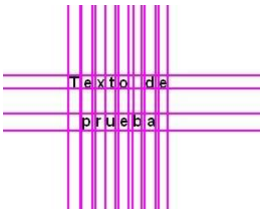


Figura 7.4: Este sistema fallaría.

Es necesario realizar un barrido vertical de izquierda a derecha no para toda la imagen si no entre la coordenada Y superior y la inferior da cada una de las líneas de caracteres antes calculada. Así iremos hallando las coordenadas X laterales izquierda y derecha para cada carácter.

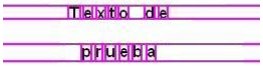


Figura 7.5: Sistema correcto.

Ya tenemos los datos mas importantes, ahora para definir un área rectangular solo es necesario guardar 2 coordenadas, por ejemplo la superior izquierda y la inferior derecha, por eso para cada carácter utilizando los valores ya calculados le asociados 2 coordenadas que definirán su área. Esta lista de pares de coordenadas será utilizada en la próxima función de estudio de la morfología del carácter para que sepa que regiones de la imagen debe analizar.

Aun nos quedaría un pequeño problema y es que dentro de una fila hay caracteres mas altos que otros, por ejemplo la 'o' es mas baja que la 'L', y para el posterior análisis nos interesa que las fronteras para cada carácter estén totalmente ajustadas a lo que realmente ocupa. Hay que reajustar las

coordenadas de cada carácter acercándolas aun mas a este, este reajustamiento solo afecta a las coordenadas Y que son las que pueden estar desajustadas, el método para ello es igual al utilizado para calcular los limites laterales. Ahora a partir de los limites laterales reajustamos para cada carácter las fronteras superiores e inferiores.

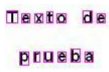


Figura 7.6: Regiones de cada letra finalmente delimitadas.

Ya sabemos que regiones ocupa cada carácter dentro de la imagen, sabemos que distancia existe entre cada región. Esta información nos permite deducir si esa distancia es un espacio entre palabras o entre letras. Esto es muy útil ya que cuando a cada región le asociemos un carácter ASCII tendremos una lista de caracteres, pero no sabremos que palabras forman. Por ejemplo, si sabemos que entre el carácter 6 y el 7 existe un espacio grande entre sus regiones esto será que el carácter 6 y el 7 pertenecen a palabras distintas.

Para esto hacemos la media aritmética de los espacios entre regiones contiguas sin contar las contiguas que están en distintas líneas, aquellos espacios entre regiones que estén por encima de la media serán espacios en blanco en el mensaje. Los espacios se representan como una cadena de booleanos de longitud igual al numero de regiones menos uno, que valdrá falso si entre dos regiones no se considera que exista espacio y cierto en caso contrario.

7.3.2. Reconocimiento de patrones

Descripción

Un reconocedor de patrones se puede considerar como un descriptor. Consiste en extraer características de un objeto para reconocerle.

Los descriptores se pueden clasificar en:

- Descriptores de frontera. (Códigos de cadena, números de contorno, signaturas, ...)

- Descriptores de región. (Descriptores globales, esqueleto de una región, textura, momentos invariantes, ...)

- Descriptores de estructuras tridimensionales.

El algoritmo utilizado en el proyecto ha sido desarrollado por nosotros, se puede incluir dentro del área de descriptores de frontera. Ya que se basa en discriminar al objeto aprovechando las relaciones geométricas inherentes a la forma del objeto. Definimos por tanto a un carácter por su contorno.

Figura 7.7: Caracter y su contorno.

El funcionamiento es el siguiente:

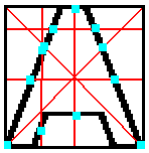


Figura 7.8: Rectas que cortan al caracter en 12 puntos.

- Ajustamos un carácter dentro de un marco.

- Trazamos una serie de rectas que cortan al carácter por varios puntos de su frontera. Estos puntos serán los que definirán al carácter. Según el dibujo si trazamos 6 rectas conseguiremos 12 puntos para definir al carácter.

  Los puntos tienen que ser independientes de la posición que ocupa el carácter dentro de la imagen en ese momento, también no tiene que importar el tamaño ya que un cartel podrá estar unas veces mas lejos que otras y una A siempre tendrá que seguir siendo una A independientemente de la distancia. También en cierta medida el algoritmo tiene que ser permisivo con distintos tipos de formato.

- Por eso, lo que se calcula es la distancia del segmento que se forma entre ese punto y el marco. No guardamos la distancia exacta, sino la proporción de la longitud del segmento con la de toda su recta. De esta forma ya no importara el tamaño de las letras, ya que aunque una A se mas pequeña q otra la proporción seguirá siendo la misma.

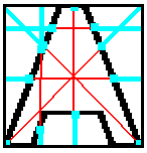


Figura 7.9: Los 12 segmentos que definen al caracter.

- Son entonces no 12 puntos lo que generamos sino 12 valores que indican la distancia relativa que hay desde esos puntos al marco.
- Para cada carácter o digito calculamos sus 12 valores, y los guardamos en una base de datos junto con su identificador (carácter ASCII).

  Así tendremos en un archivo la descripción para cada carácter en forma de 12 valores.

  Ejemplo para 3 letras:

  A 0.000000 30.882353 20.744681 20.744681 42.647059 26.470588 31.914894 31.914894 42.647059 26.470588 9.574468 9.574468

  B 0.000000 0.000000 0.000000 13.815789 0.000000 0.000000 0.000000 5.921053 2.941176 2.941176 0.000000 0.000000

  C 0.000000 0.000000 0.000000 85.227273 5.714286 5.714286 5.113636 5.113636 4.285714 4.285714 5.113636 1.704545

Reconocimiento

El reconocimiento es un proceso de etiquetado. se ocupa de identificar cada objeto segmentado de la escena y asignarle una etiqueta.

- Como todo modulo, el modulo OCR tiene su función Iniciar, esta función siempre cargara el archivo de descripciones en memoria para continuas consultas.
- En cada ciclo cada vez que le llegue una imagen filtrada, el método anterior llamado enmarcamiento de caracteres nos pasara en cada ciclo, una lista de N áreas o subimagenes q tendremos q tratar.

- El OCR irá letra por letra (subimágenes) calculando sus 12 valores y comparándolos con los valores de las letras de la base de datos.
- Aquel carácter de la base de datos cuyos valores sean mas parecidos, es decir, generen menor error, será el que asignemos a esa subimagen. Es decir a cada subimagen se le asocia un carácter ASCII.
- Así con toda la lista de N subimagenes lo q generara una lista de N caracteres ASCII. Es decir una cadena de texto.
- La función anterior de enmarcamiento como dijimos infería por la distancia entre áreas si había o no espacio entre letras, esta información se la pasa al reconocedor que la utiliza sobre la cadena de texto, poniendo o no espacios en blanco.
- Así ya tenemos una cadena de texto formada por palabras. Palabras cuyas letras serán las que menos error han cometido según el método utilizado.

Por tanto este método asegura el reconocimiento de caracteres a distinto tamaño y en formatos distintos (no cursiva).

La orientación del mensaje se resuelve dentro del modulo de filtro anterior, con la función de rotación.

7.3.3. Análisis morfoLexico

Muchas veces a causa del ruido del filtrado o por el formato de las letras, el OCR no asocia el carácter ASCII correcto a la subimagen, por ejemplo puede confundir la letra 'N' con la 'H', ya que en algunos casos aunque lo que hayamos querido expresar en la imagen era una 'N' la 'H' genera menos error.

Por este motivo ha sido necesario crear una ultima fase, que asegure la detección y corrección de errores en el mensaje reconocido. Ya que a la salida de este modulo el mensaje ha de ser claro, para que robot pueda tratarlo ya sea como orden o como otra operación.

Para este ejemplo:

Frase reconocida por el OCR: "TEXTO P8D58A"

Frase corregida por el analizador morfo-léxico: "texto prueba"

Esta ultima fase la forma una serie de funciones que componen el análizador morfo-léxico del modulo OCR. Todo análisis morfo-léxico se base en:

- Diccionarios (lexicones)

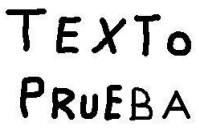


Figura 7.10: Imagen recibida desde modulo filtro.

- Reglas morfológicas

Ambas cosas son interdependientes. Si en el diccionario sólo guardamos lexemas, necesitaremos muchas reglas morfológicas.

Si guardáramos todas las formas de las palabras en el diccionario, no necesitaríamos reglas morfológicas. Esto es lo que hemos seguido con nuestro diccionario, un diccionario con todas las formas verbales, etc.

Dificultades con los diccionarios:

- Polisemia: palabra con varios significados. Ej.: banco
- Homonimia: palabras distintas con la misma grafía. Ej.: divorciado: nombre, adjetivo y verbo

Si fuésemos a hacer un tratamiento total del lenguaje natural del mensaje sería necesario que el diccionario fuese mucho mas completo, es decir, que guarde no solo las palabras si no mas información del tipo:

- Categoría sintáctica: preposiciones, conjunciones, nombre, adjetivo, verbo, etc.
- Concordancia. Género, número, persona, caso.
- Preposiciones que admite un verbo, tipos de complementos, etc.
- Información morfológica (patrón de formación de la palabra).
- Información semántica. Concepto correspondiente, palabras sinónimas.

Los diccionarios se suelen organizar utilizando relaciones de herencia múltiple, tanto de tipo gramatical como conceptual.

Se implementan con tablas hash, tries o árboles B.

Nuestro diccionario no es tan complejo ya que solo actúa como verificador ortográfico.

Se organiza por bloques de palabras, cada bloque contiene palabras de la misma longitud, por tanto si la palabra mas larga en español contiene 23 letras, el diccionario posee 23 bloques, con palabras desde longitud 1 a longitud 23. Cada bloque es una tabla hash.

La longitud de la palabra es algo en lo que se suelen cometer errores en el OCR, por tanto ya directamente buscamos la palabra de misma longitud, en un bloque determinado del diccionario. Si esta palabra no existe en la tabla hash solo se buscan palabras similares en un conjunto mucho mas restringido de palabras, que son las de su misma longitud.

Como ya hemos dicho el análisis en el modulo OCR no es demasiado amplio, esta parte entraría en el ámbito del modulo DCG. El OCR hace un análisis morfo-léxico sencillo donde solo comprueba que las palabras del mensaje existen dentro del diccionario. Y en el caso de que no exista una palabra actúa de corrector ortográfico buscando la palabra del diccionario del mismo tamaño en letras que mas se aproxime a la palabra de la imagen, es decir, que según el método de reconocimiento del OCR menos valor de error genere.

El error se calcula viendo los valores de los puntos que describen a cada carácter de la palabra del diccionario en la base de datos del OCR, comparando la diferencia del valor de estos puntos con los obtenidos de su correspondiente carácter asociado de la imagen, se va viendo cual de las palabras que realmente existen en el diccionario genera menos error. Recuerdo que la palabra que menos error genera según este método sería una secuencia de caracteres que no forman ninguna palabra existente. Por eso buscamos una palabra de mayor error, pero que exista.

Este método tiene un gran porcentaje de acierto entre el mensaje que representa la imagen y la que saca como salida el analizador.

Es por tanto el diccionario quien en ultima fase, pone la cadena de caracteres a la salida del modulo. Asegurando por tanto que todas las palabras que salgan de este modulo, son palabras que existen en el diccionario de español y que mas se corresponden con el mensaje del cartel.

7.4. Código

Ver HTML: documentación del módulo en ocr.c y en ocr_code.c.

Capítulo 8

Módulo de gestión de mensajes

8.1. Introducción

Este módulo de la aplicación tiene como misión recoger todos los mensajes que han podido ser generados por módulos de proceso anteriores, y filtrarlos de tal modo que la salida de la línea de ejecución esté dotada de cierta coherencia con el resultado deseado. Para ello, admite las entradas en forma de cadenas de texto, y elige cuáles de ellas son las que deberían dar realmente una salida a los módulos posteriores.

8.2. Detalles

- **Entrada**: Cualquier cadena de texto, y parámetros para controlar la tolerancia.
- **Salida**: Una cadena de texto, la que más se asemeja a la que realmente debería ser generada.
- **Descripción**: Módulo que, a través de la parametrización, guarda una tabla con las entradas, y, en función de las cantidades de información, presenta la mejor salida.

8.3. Arquitectura y funcionamiento del módulo

El módulo trabaja con una *tabla hash* inicialmente vacía. Cuando recibe señales procedentes del módulo procesador, realiza una de las dos opciones siguientes:

1. **El elemento no estaba en la tabla**: Se añade a la tabla y se suma una unidad.

2. **El elemento ya estaba en la tabla**: Se suma una unidad al número de llegadas consecutivas de ese elemento.

Tras este primer paso, se procede a la "debilitación" de las otras señales. Con esto queremos decir que reducimos el índice de refuerzo asociado a todas las señales que no fueran la que hemos escogido para que impere, tras una serie de ciclos que parametrizamos a través de los argumentos del módulo, la señal más importante (la que más veces seguidas ha llegado), que consideramos como la real que debería ser transmitida.

De esta forma, siempre mantenemos en la tabla todas las señales que van llegando, y creamos un tipo de diagrama de estados borroso, en el que el estado principal es decidido mediante los valores que el módulo va asignando a cada registro de la tabla.

Como utilidad añadida, este módulo es uno de los más genéricos del *pipeline*. El hecho de que los estados posibles se vayan creando de forma dinámica, y que sean sólo diferenciados por una cadena de texto, ha hecho posible que el módulo sea usado para gestionar diferentes líneas, sin tener que modificar ni una línea de código. La única parte que hay que personalizar son los parámetros de la instanciación del módulo, para que el comportamiento sea lo mejor posible.

Capítulo 9

Procesamiento de texto reconocido

9.1. Introducción

Una vez que el OCR ha reconocido una cadena de texto y la ha validado, envía la señal a este módulo, que se encarga de procesarlo, y crear una nueva salida en el formato propio del pipeline para el entorno 3D y el robot.

9.2. Detalles

- **Entrada**: Una cadena ya preparada del OCR.
- **Salida**: Un resultado en forma de cadena de respuesta o de orden para el robot.
- **Descripción**: Módulo que, a partir de una cadena ya reconocida, calcula un resultado matemático, o da órdenes de conducta a un robot.

9.3. Implementación

Para implementarlo, hemos usado un programa en **Prolog**, que está formado en su mayor parte por una DCG [1]. Este programa recibe la entrada del módulo, y, tras procesarla, crea una salida, que puede ser una cadena de respuesta, el resultado de una operación aritmética (soporta paréntesis y anidamiento de expresiones) o una orden para la salida.

[1] Definite clause grammar

9.4. Ampliabilidad

La idea de crear el módulo como antes hemos explicado ha sido propiciada por la idea de que la "inteligencia del robot" fuese algo ampliable. Con sólo modificar este archivo, se puede dotar a la máquina de más información (ampliando el diccionario), o de más capacidad de proceso (creando más reglas de reconocimiento en la DCG).

Capítulo 10

Módulo de control del robot

10.1. Introducción

El módulo de control de robot ha sido creado con el fin de tener una pieza de software capaz de controlar nuestro robot de una forma sencilla y transparente para los módulos que generan la información de salida. A este módulo le llega una estructura de datos que contiene la orden y el parámetro, y el robot se encarga de moverse en función de esa información. A continuación detallamos cómo lo hace.

10.2. Detalles

- **Entrada**: Una estructura de datos que contiene una cadena de órdenes y otra de parámetros. Un script de comportamiento y la configuración del puerto como argumentos.
- **Salida**: El movimiento del robot.
- **Descripción**: Módulo que, a través de un guión de comportamiento y de unas órdenes, mueve un robot por el puerto paralelo.

10.3. Arquitectura del módulo

El módulo en sí tiene una estructura muy simple: consiste en un conjunto de funciones que son exportadas a un *script* programado en el lenguaje **Lua**, y que funcionan principalmente haciendo llamadas a una librería que hemos creado, y que se encarga de controlar el puerto paralelo.

10.3.1. Scripts

La verdadera programación de la función del robot la hemos delegado en un *script*. Este método de programación nos permite más flexibilidad a la hora de controlar el comportamiento de la máquina.

En el script del robot implementamos algunas funciones determinadas que necesita el módulo de control, y las que nosotros deseemos. En este código definimos de una manera muy simple qué tiene que hacer el robot (realmente el puerto paralelo) cuando le llegan las órdenes de los otros módulos.

10.3.2. Biblioteca de control del puerto paralelo

Hemos desarrollado una biblioteca que provee un interfaz lo más simple posible de control de los pines del puerto paralelo, y que además es **multiplataforma**. A través de ella se puede controlar los pines del cable, con llamadas simples, en las cuales sólo hay que especificar que pin se quiere usar, y si se quiere poner a *alta* o a *baja*. De este modo nos hemos abstraído, en la implementación del módulo en sí, de las pequeñas dificultades que puede ocasionar interactuar directamente con la entrada/salida del ordenador.

Para la implementación de esta biblioteca hemos usado a su vez dos bibliotecas externas: **parapin** para la implementación en GNU/Linux, y la DLL **inpout32.dll** para el uso en plataformas Win32.

10.4. Construcción del robot

Para construir la estructura física del robot, hemos usado piezas de *Lego Technics*. Este material es barato y razonablemente consistente para soportar su propio peso, el de la cámara, el circuito, y el cable paralelo. Además, destaca principalemente por su versatilidad de uso y su capacidad de reconstrucción. El ensamblado de las piezas es inmediato (hay que tener en cuenta que se vende como juguete para niños a partir de 12 años), y los errores de estructura se pueden subsanar con mucha facilidad.

Sin embargo, hemos escogido este material por la facilidad que hemos encontrado para generar máquinas móviles. Los conjuntos de piezas de *Lego Technics* suelen venir acompañados por estructuras más complejas de construcción, como motores y brazos hidráulicos, elementos que hemos usado para nuestro robot.

El circuito ha sido conectado sobre una placa pequeña de entrenador, a pesar de su fiabilidad relativa, es rápida de montar y de depurar. Hemos usado un chip **L293B**, que sirve para control de motores bidireccionales de

corriente contínua, y alimentamos el circuito con una pila de 9 voltios. El circuito que usa el robot es el de la figura 10.1.

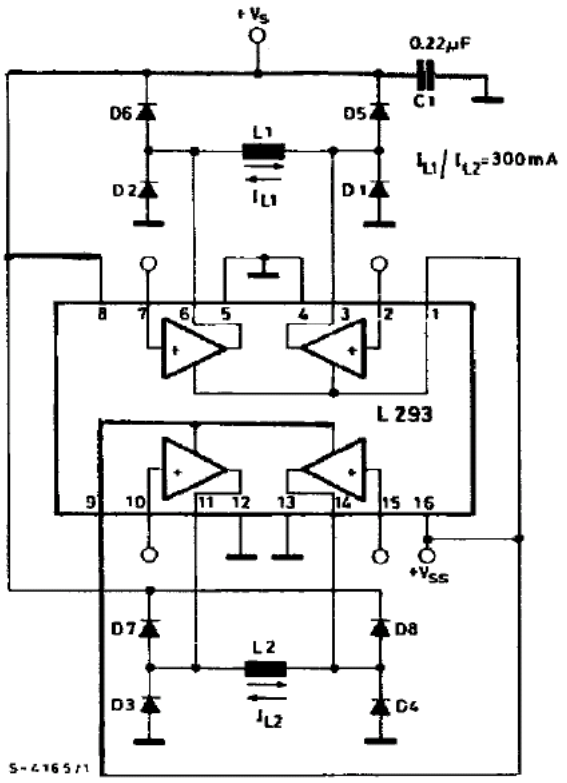


Figura 10.1: Cirtuito de control robot con chip *L293B*

10.5. Fotos

A continuación mostramos algunas fotos del robot que hemos construido:

Figura 10.2: Foto inferior del robot

Figura 10.3: Foto lateral del robot

Capítulo 11

Módulo de Entorno 3D

11.1. Introducción

El entorno virtual 3D constituye un interface visual con el usuario que puede observar la simulación de las evoluciones de un robot que se desplaza por un escenario siguiendo las órdenes procesadas por el sistema de visión por computador.

Con esta aplicación, es posible evaluar y testar las funcionalidades que se han implementado sin tener que llegar a la integración del sistema en una entidad robótica real.

11.2. Detalles

- **Entrada**: Una estructura de datos que contiene una cadena de órdenes y otra de parámetros.
- **Salida**: Movimiento del robot virtual.
- **Descripción**: Módulo que a partir de unas órdenes representa el desplazamiento de un robot virtual a través de un escenario en tres dimensiones.

11.3. Especificación

Para cumplir con su cometido, la aplicación debe poseer las siguientes características funcionales:

11.3.1. Representación tridimensional de un entorno/escenario

Teniendo en cuenta que una de las principales motivaciones para elaborar un sistema de órdenes mediante visión artificial es la de implementar un dispositivo de navegación que permita a un autómata desplazarse por su entorno, es fundamental simular gráficamente un escenario sobre el cuál el robot virtual pueda desenvolverse. Esta representación debe ser lo más inmersiva posible para que la precisión de la simulación sea adecuada.

11.3.2. Representación de un robot capaz de desplazarse por su entorno

Debe existir una entidad (en este caso un modelo 3D) dentro de la simulación que represente la ubicación, orientación y desplazamientos resultantes de las distintas órdenes procesadas por el sistema de control. Aunque en principio no es relevante, se ha intentando que esta entidad cumpla con ciertos criterios de diseño que empaticen con los sistemas motrices más comunes en autómatas (uso de ruedas, orugas, etc).

11.3.3. Sistema de control

Las diferentes órdenes que son procesadas por el sistema de visión artificial, son analizadas para establecer las nuevas propiedades de posicionamiento, dirección, etc de la entidad que representa al robot. Esta interacción se realiza mediante el paso de mensajes distribuidos por el pipe que se encarga de la interconexión entre los diferentes módulos. Además, es necesario que de forma autónoma la aplicación pueda procesar diferentes comandos emitidos por el usuario mediante el teclado y ratón para poder controlar otros aspectos secundarios de la simulación, como son el posicionamiento de la cámara, activación de diferentes efectos gráficos, etc.

11.3.4. Sistema de cámaras interactivas

Para poder visualizar de forma óptima todos los componentes de la simulación, es necesario poseer un sistema de cámaras que permita seguir cómodamente las evoluciones del robot por el escenario, así como permitir al usuario adaptar la posición de las distintas cámaras para obtener el ángulo de visión más relevante en cada momento.

11.4. Implementación

La consecución de las especificaciones previamente expuestas se ha conseguido mediante la implementación de un amplio conjunto de características relacionadas principalmente con la programación gráfica 3D. A continuación se describen detalladamente:

11.4.1. Uso de DirectX 9.0

La implementación de los gráficos tridimensionales se ha realizada sobre la librería gráfica Direct3D incluida en DirectX 9.0. Se ha optado por este estándar en vez del uso también muy extendido de la librería OpenGl por razones académicas, en el sentido de que previamente habíamos tenido experiencia con OpenGl en otros proyectos y esta se presentaba como una oportunidad idónea para investigar nuevas tecnologías. Cabe señalar que ambas librerías poseen similar potencialidad por lo que la elección por razones funcionales no era especialmente relevante.

11.4.2. Modelos 3D creados sobre 3dStudio Max 6.0

Se ha empleado la herramienta de modelado 3dStudio Max para la elaboración de los modelos tridimensionales tanto del escenario como del robot. Para su posterior integración con la aplicación, se ha utilizado un conversor que compatibiliza el formato usado por 3dStudio con el usado por Direct3D (archivos .X). En cuanto al apartado de las texturas, se ha utilizado Adobe PhotoShop para su realización.

11.4.3. Terreno generado a partir de un mapa de altura

Como parte del escenario, se ha incluido un terreno que es generado de forma procedural a partir de un mapa de altura. Una vez calculado la malla 3D de dicho terreno, se genera de forma procedural la textura a partir de diferentes imágenes que se interpolan siguiendo como criterio la altitud del terreno. De esta forma, según lo alto o bajo que esté el terreno, este presentará el aspecto de diferentes materiales (hierba para las zonas bajas, roca en las cumbres de las montañas, etc). Por último se utiliza un algoritmo primitivo de sombreado que ilumina la textura resultante de forma coherente respecto a la posición de la luz en el escenario (en este caso del sol).

11.4.4. Iluminación dinámica

Para otorgar una mayor sensación de integración de la entidad que se desplaza con el escenario, se ha utilizado iluminación dinámica. De esta forma el robot es iluminado correctamente según su posición actual, incrementando además el realismo en la percepción de los materiales al visualizarse efectos de brillo, luz difusa, variación cromática,etc.

11.4.5. Iluminación estática. LightMaps

Los elementos estáticos del escenario no requieren de iluminación dinámica ya que su posición no varía durante la ejecución de la aplicación. Para ahorrar recursos, es habitual el uso de texturas secundarias también llamadas lightmaps que codifican la información sobre la iluminación que ese objeto recibe. En este caso, se ha utilizado una única textura resultado de la fusión de la textura primaria y los lightmaps para optimizar el rendimiento.

11.4.6. Sombreado Dinámico. Stencil Buffer

De forma análoga a la iluminación dinámica, el sombreado dinámico es un efecto que permite la integración de objetos móviles en escenarios de forma muy realista. Se ha implementado un algoritmo de sombreado que se basa en el uso del Stencil Buffer. Este algoritmo es bastante costoso en cuanto a recursos de la tarjeta gráfica, por lo que sueles implementarse para ser soportado por tarjetas con aceleración 3D de última generación.

11.4.7. Luces Glow. Lens Flares

Las luces Glow son puntos lumínicos que producen un haz a su alrededor cuando se mira directamente. En este caso, se ha implementado un punto de luz que representa al sol. Además, se ha implementado un efecto conocido como Lens Flares que se produce cuando se enfoca un sistema óptico (cámara de fotos, de video,etc) sobre un foco de luz intensa. Este efecto produce una serie de reflejos residuales que se ubican de forma relativa a la luz que los origina. Se ha introducido por razones estéticas y para incrementar la inmersión en el entorno virtual.

11.4.8. Sistema de cámaras

Se han implementando varios sistemas de cámaras para ofrecer múltiples posibilidades al usuario de seguir la acción que se desarrolla en la simulación. Se dividen en:

- Cámara de seguimiento rígido: La cámara se sitúa siempre detrás del robot y sigue cada uno de sus movimientos permaneciendo siempre a una misma distancia.

- Cámara de seguimiento orbital: A diferencia de la anterior, esta cámara orbita alrededor del robot manteniendo siempre su punto de enfoque fijado en él.

- Cámara fija: Se establece una posición de la cámara donde permanece fija mientras enfoca y sigue los movimientos del robot.

- Cámara libre: Con esta cámara se puede navegar por todo el escenario así como enfocar a los puntos de mayor interés para el usuario.

11.4.9. Sistemas de partículas

De forma genérica, se ha implementado un sistema de partículas para poder introducir efectos especiales y atmosféricos como pueden ser humo, fuego, lluvia, nieve, etc. Los sistemas de partículas combinan una representación gráfica mediante billboards(polígonos especiales que mantienen siempre su orientación respecto a la cámara) y un sistema de control físico que determina el comportamiento de las partículas en cuanto a aceleración, dirección, tiempo de vida, turbulencia, etc.

11.5. Diseño

El diseño de la aplicación es bastante sencillo en cuanto a su estructura jerárquica. Existe una clase que contiene a la VentanaPrincipal de la aplicación y que constituye el núcleo de la ejecución. Esta clase contiene un objeto del tipo Escena que es el que administra el resto de entidades con sus correspondientes representaciones gráficas. Así, la escena es el contenedor para otros objetos como son el Terreno, el Cielo, SistemaPartículas, Cámara, y el resto de entidades como el escenario y el propio robot que pertenecen a la clase Objeto. Respecto a esta última clase, destacar que posee como atributo un objeto de la clase Mesh que constituye la representación gráfica de la entidad (es decir, la maya 3D). Para gestionar la carga de los diferentes modelos 3D, existe una clase estática llamada MeshManager que comprueba que no se carguen en memoria varios modelos del mismo tipo. Por tanto, la creación de objetos del tipo Mesh se realiza siempre a través de esta clase gestora y nunca directamente. Por último, existen dos clases estáticas que

se encargan de la interacción del usuario mediante el ratón y teclado (clase Entrada) y otra que gestiona la impresión de texto en pantalla (clase Texto).

Respecto al flujo de ejecución, se puede resumir en que existe un método que es llamado externamente de forma periódica. Dicho método hace que en cada ciclo se actualice la posición de cada uno de los objetos de la escena y posteriormente se renderice. Este proceso se hace de forma delegada, de forma que VentanaPrincipal llamará al método Render() de la Escena, la cuál llamará recursivamentea a los respectivos métodos Render() de cada una de las entidades que contiene.

11.6. Capturas

Se muestran a continuación una serie de capturas de pantalla del módulo en funcionamiento:

Figura 11.1: Primer plano del robot.

Figura 11.2: Seguimiento del robot con una cámara estática.

Figura 11.3: Visión general del escenario.

Capítulo 12

Otros módulos

12.1. Introducción

A continuación explicamos el funcionamiento del resto de los módulos. Los aglutinamos en esta sección, por carecer de interés la explicación detallada de los mismos, debido a su sencillez.

12.2. Módulo de post-gestión

Hemos añadido un módulo que recoja todas las señales que son generadas por las distintas ramas de proceso del *pipeline*, y las agrupe en una sola salida válida, para ofrecer de este modo, a través de sus puertos, datos a todos los módulos de salida.

12.2.1. Detalles

- **Entrada**: Cadenas que representan órdenes o parámetros. Como argumentos, las cadenas por defecto.

- **Salida**: Una estructura de tipo *orden/parámetro* para el robot o el entorno 3D.

- **Descripción**: Módulo que se encarga de recoger todo el proceso del pipeline y unificarlo en una sola estructura.

12.3. Ventana de parámetros

La ventana de parámetros es una interfaz gráfica en la que es posible elegir un color, y tres tolerancias (una para el rojo, otra para el verde y otra para el azul), y alimentar así a un módulo de filtro, de tal modo que la parametrización de los valores del mismo se pueda hacer en tiempo real.

12.3.1. Detalles

- **Entrada**: Sólo los argumentos del XML, con colores por defecto.
- **Salida**: Una estructura que define colores y tolerancias sobre los mismos, para los filtros.
- **Descripción**: Módulo que se encarga de controlar los colores del filtrado.

12.4. Ventana de imágenes

La ventana de imágenes es simplemente un módulo que muestra gráficamente, en una ventana que se redimensiona automáticamente, una imagen que sigue el formato propio de nuestra aplicación (ver sección 4.3).

La ventana de imágenes tiene como función añadida la capacidad de guardar una toma de la imagen que se este dibujando en el instante en el que se pulsa **F5**.

12.4.1. Detalles

- **Entrada**: Una imagen en el formato del pipeline, y la información del archivo que se guardará.
- **Salida**: No tiene salida, más que la gráfica, por pantalla.
- **Descripción**: Módulo que se encarga de mostrar imágenes.

12.5. Módulo de control de joystick

Hemos añadido también al pipeline un módulo que, a través de un interfaz de joystick, puede traducir sus entradas a las correspondientes órdenes y parámetros del robot y del entorno 3D. De esta forma, la depuración es sencilla y rápida.

Para la implementación hemos utilizado las bibliotecas SDL (Simple Directmedia Library).

12.5.1. Detalles

- **Entrada**: La entrada del joystick, y, como argumentos, la configuración de este dispositivo.
- **Salida**: Órdenes directas para el robot.
- **Descripción**: Módulo que traduce el control de un joystick estándar a movimientos del robot o el entorno 3D.

12.6. Módulo de salida

Este módulo es simplemente una ventana con un cuadro de texto que es capaz de imprimir en él todas las cadenas (en formato C, `char *`) que le llegan a través de sus puertos.

12.6.1. Detalles

- **Entrada**: Cualquier texto.
- **Salida**: El texto en una ventana.
- **Descripción**: Módulo auxiliar que muestra texto por pantalla.