TRABAJO FIN DE GRADO

INGENIERÍA EN INFORMÁTICA

Reconocimiento de Expresiones Faciales

por Computador

Autor
Francisco José Fajardo Toril

Directores
Miguel García Silvente
Eugenio Aguirre Molina

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, 22 de agosto de 2017

Reconocimiento de Expresiones Faciales

por Computador.

Autor
Francisco José Fajardo Toril

Directores
Miguel García Silvente
Eugenio Aguirre Molina

Reconocimiento de Expresiones Faciales: por Computador

Francisco José Fajardo Toril

Palabras clave: Reconocimiento, expresión, facial, visión, computador, aprendizaje, automático, redes, neuronales, convolucionales.

Resumen

El objetivo de este documento es mostrar al lector el proceso seguido para el desarrollo de una aplicación software capaz de resolver el problema del reconocimiento de expresiones faciales por computador, que puede ser descrito como:

Dada una imagen de entrada, obtener la etiqueta de salida que mejor identifique la expresión facial que aparece en dicha imagen.

De entre todas las técnicas posibles para la resolución del problema, he basado mi solución en el entrenamiento una red neuronal convolucional. El modelo entrenado clasifica la imagen de entrada mediante la asignación de una (o varias) etiquetas, incluyendo un porcentaje por cada una que indica cuál es la más probable de estar presente en dicha imagen. El conjunto de posibles etiquetas es:

- Afraid
- Angry
- Disgusted
- Happy
- Neutral
- Sad
- Surprised

Facial Expression Recognition: by Computer

Francisco José Fajardo Toril

Keywords: Facial, expression, recognition, computer, vision, machine, learning, convolutional, neural, network.

Abstract

This document main objective is to show the reader the process I followed to develop a software application which is able to solve the face expression recognition problem through computer. It can be described as:

To obtein the tag that better describe the facial expression that appears in the given input image.

Between all the possible techniques that allow us to solve this problem, I chose to train a convolutional neural network. Trained model classify the input image by the assignment of one or few tags, including a percentage that suggest which is the most probable tag to appear in that image. Here is the whole set of possible tags:

- Afraid
- Angry
- Disgusted
- Happy
- Neutral
- Sad
- Surprised

Yo, **Francisco José Fajardo Toril**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 76424577Q, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Francisco José Fajardo Toril

Granada a 22 de agosto de 2017.

D. **Miguel García Silvente**, Profesor del Área de Ciencias de la Computación e Inteligencia Artificial del Departamento YYYY de la Universidad de Granada.

D. **Eugenio Aguirre Molina**, Profesor del Área de Ciencias de la Computación e Inteligencia Artificial del Departamento YYYY de la Universidad de Granada..

Informan:

Que el presente trabajo, titulado ***Reconocimiento de Expresiones Faciales, por Computador***, ha sido realizado bajo su supervisión por **Francisco José Fajardo Toril**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 22 de agosto de 2017.

Los directores:

Miguel García Silvente **Eugenio Aguirre Molina**

Agradecimientos

Poner aquí agradecimientos...

Capítulo 1

Introducción

1.1. Motivación

El problema del reconocimiento de expresiones faciales es un paso adelante tras resolver el problema del reconocimiento facial en imágenes. Tal y como mencioné en el resumen puede ser descrito de la siguiente manera:

Dada una imagen de entrada, obtener la etiqueta de salida que mejor identifique la expresión facial que aparece en dicha imagen.

A priori la solución para un ser humano es trivial. Tenemos muchas maneras de adivinar el estado de ánimo de una persona, pero si tenemos que basarnos en nuestra visión, una posible solución sería: *"Simplemente observa los rasgos faciales del sujeto e intenta adivinar su estado de ánimo basándote en tu experiencia pasada con otros seres humanos."*

Imagine que deseamos desarrollar un agente basado en inteligencia artificial para que interactúe con otros seres humanos. Sería de gran utilidad que este agente basara sus acciones en función del estado de ánimo de la persona/as con las que está comunicándose para lograr de esta forma un mayor grado de comprensión o empatía. De la misma manera podríamos pensar en una aplicación que escoge la música idónea para nosotros en función de nuestro estado de ánimo en ese momento a través de una captura mediante la cámara del computador.

1.1.1. Trabajos relacionados

Como se puede ver, la solución a este problema puede ser de utilidad en diversas aplicaciones del mundo real, sin embargo, ¿Cómo podría un computador resolver este problema? Hay diversas propuestas que han tratado de resolver este problema.

Figura 1.1: Puntos de Referencia extraídos con ASM.

Facial Expression Analysis using Active Shape Model[1]

Active Shape Model(ASM)[3] es una técnica que permite, dada una imagen del objeto para el que fue entrenado, localizar un conjunto de puntos de referencia previamente definidos tal y como se puede ver en la Figura 1.1. En el caso de un rostro humano estos puntos pueden estar los principales indicadores de la expresión facial, como pueden ser las cejas, boca, ojos... En este paper [1] en concreto, se tienen en cuenta para cada cara, 68 puntos de referencia. Se procesan los puntos extraídos y se obtiene un modelo geométrico para ese rostro concreto. Acto seguido, haciendo uso de la técnica de clasificación conocida como Support Vector Machine(SVM), se clasifica ese modelo geométrico para predecir a qué expresión facial pertenece obteniendo un porcentaje de acierto del 92,1 %.

Facial expression recognition and synthesis based on an appearance model[2]

En este caso se hace uso de la técnica *Active Appearance Model (AAM)*[4], un método estadístico que se emplea para emparejar, un modelo geométrico, a una nueva forma del mismo tipo para el que fue entrenado, pero que el modelo no ha visto antes, como por ejemplo un nuevo rostro humano. Esta técnica (AAM) es una generalización de la ya antes mencionada ASM[3]. Además, lo que en este caso se intenta es que la información usada esté exenta de redundancia por lo que se aplican técnicas de aprendizaje no supervisado en una fase de preprocesamiento de la información como *Análisis de Componentes Principales (PCA)* o *Análisis de Componentes Independientes(ICA)* para reducir la dimensionalidad de la información en una fase de preprocesamiento. En este caso obtienen alrededor de un 84 % de acierto.

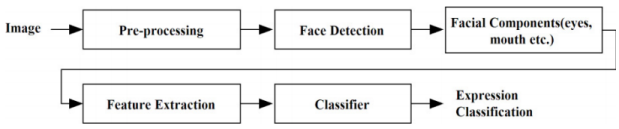

Figura 1.2: Diagrama general del proceso de reconocimiento.

1.1.2. ¿Qué se pretende conseguir con este proyecto?

Las técnicas de ASM[3] y AAM[4] son sensibles a rotaciones de la imagen, es decir, será difícil extraer los puntos de referencia en imágenes que disten mucho del concepto de imagen frontal, como por ejemplo una la fotografía de una persona mirando hacia un lado con una expresión facial determinada. Además, en la Figura 1.2 extraída del siguiente paper[5], se puede ver el proceso general que se sigue a la hora de clasificar una imagen, en nuestro caso reconocimiento de expresiones faciales.

Figura 1.3: Diagrama del proceso objetivo.

La idea principal que se quiere conseguir para resolver el problema consiste en simplificar de alguna manera dicho diagrama para que se asemeje al de la Figura 1.3 de forma que no sea necesario extraer características cada vez que se clasifique una nueva imagen. Otra de las propuestas es que la rotación de la imagen de entrada no suponga un hándicap a la hora de clasificarla. El propósito de estas medidas es:

- Ganar en robustez con imágenes rotadas o inclinadas.
- Realizar la clasificación en tiempo real simplificando el preprocesamiento de las imágenes.
- Combinar todo en una aplicación de escritorio con una interfaz simple por encima para que cualquier usuario pueda utilizarla.

1.1.3. Organización del documento

Este documento tratará de explicar el proceso seguido para el desarrollo de este trabajo. Para ello, se estructurará en los siguientes capítulos:

Portada - Información básica sobre el nombre del proyecto, la institución, el autor y los tutores.

Prefacio - Resumen, palabras clave y agradecimientos.

Introducción - Motivación del trabajo y estado del arte de este campo de investigación.

Objetivos - Se planteará un objetivo principal el cual será dividido en sub-objetivos más simples para poder llevar a cabo la tarea principal así como un resumen de las técnicas aprendidas durante la formación académica empleadas en este trabajo.

Resolución del trabajo - Este capítulo se dividirá en varias secciones, en las que se explicará el método de ingeniería del software empleado para la resolución del trabajo, así como los recursos hardware y software, la especificación de requisitos, planificación, análisis funcional e implementación de la aplicación.

Conclusión - Se hará una valoración del producto final y se valorará si se han alcanzaron o no los objetivos iniciales y en qué medida, teniendo en cuenta los puntos fuertes y débiles de la solución.

Bibliografía - Referencias y citas bibliográficas usadas.

Anexo - Pequeño manual de cómo usar la aplicación.

Capítulo 2

Objetivos

2.1. Objetivo principal

Desarrollar una aplicación de escritorio que dada una imagen detecte si aparece un rostro humano y en que en caso afirmativo, sea capaz de mostrar por pantalla, la etiqueta que mejor identifica la expresión facial que presenta dicho rostro.

Las etiquetas de las expresiones que deseamos predecir son:

- Afraid
- Angry
- Disgusted
- Happy
- Neutral
- Sad
- Surprised

Se ha de cumplir este objetivo sin olvidar qué es lo que se pretende alcanzar con este trabajo (apartado 1.1.2). Esto es, simplificar el proceso de pre-procesamiento de las imágenes y de esta forma ver si es factible un proceso de clasificación en tiempo real. Teniendo esto en mente, dividiremos el objetivo principal en distintas fases o sub-tareas.

2.2. Objetivos específicos

1. Preparación de la información con la que entrenaremos el clasificador.
   a) Obtener una base de datos.
   b) Pre-procesamiento de la información.
   c) Creación del conjunto de training, validación y test.
2. Entrenamiento del clasificador.

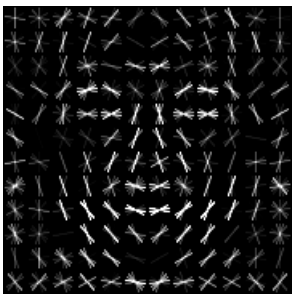

Figura 2.1: Visualización de descriptores HOG en un rostro humano [11]

a) Selección de parámetros.

b) Realización de pruebas (Vuelta al paso 2 si es necesario).

3. Desarrollo de interfaz que haga uso del clasificador.

2.3. Técnicas empleadas

Hasta ahora hemos hablado de objetivos, sin entrar demasiado en detalle acerca de qué técnicas serán las que emplearemos para llevarlos a cabo. En la siguiente sección hablaremos sobre ellas, dando una pequeña explicación de su fundamentación teórica.

2.3.1. Detector Facial

Dada una imagen, lo primero que necesitamos saber es si aparece un rostro humano o no y en función de ello hacer que nuestra aplicación, actúe de una manera u otra. De esto es de lo que se encarga precisamente un detector facial.
Para llevar a cabo esta tarea, el detector facial que he usado ha sido entrenado para aprender mediante un algoritmo basado en *SVM*[8] los descriptores de características de una batería de imágenes de rostros humanos previamente marcados y localizados manualmente (Si pudiésemos automatizar el proceso de marcado no necesitaríamos un detector facial). Los descriptores de características aprendidos por el detector son los conocidos como *Histogram of Oriented Gradients (HOG)*[6] de los que hablaremos a continuación.

¿Qué es un descriptor de características?

Es una representación de un conjunto de píxeles, ya sea de una imagen completa o de un trozo de esta. Su misión es simplificar la información que representa una imagen de forma que el descriptor represente la información más útil de la imagen eliminando aquella que no es representativa.

Histogram of Oriented Gradients (HOG)

Hay muchos descriptores de características distintos, pero los que usa nuestro detector son conocidos como descriptores HOG. Estos cuentan las ocurrencias de orientaciones de gradiente en zonas localizadas de una imagen. El gradiente nos indica por cada píxel la magnitud del mayor cambio de intensidad posible y su dirección de oscuro a claro. Podemos observar un ejemplo de esto en la Figura 2.1.
Usando un algoritmo[7] basado en la técnica de aprendizaje automático conocida como *Support Vector Machine*[8], el detector es capaz de aprender la composición de los descriptores de características HOG de un gran número de caras distintas.
Así, dada una imagen, se calculará su descriptor de características y si se encuentra en ella una ventana que contenga descriptores que se asemejen a los aprendidos, habremos detectado satisfactoriamente una cara.

2.3.2. Redes Neuronales Convolucionales

Tal y como mencioné en la motivación de este trabajo, sin entrar demasiado en detalles, que una de las restricciones de este proyecto era minimizar la fase de pre-procesamiento (apartado 1.1.2) antes de clasificar una imagen.

Figura 2.2: Esquema general de detección haciendo uso de algoritmos basados en Machine Learning [12]

Como podemos ver en la Figura 2.2 haciendo uso de algoritmos de *Machine Learning* debemos extraer, de la imagen de entrada, una serie de características que la identifiquen, dárselas al algoritmo escogido (previamente entrenado usando un conjunto con esas características) el cual, nos proporcionará una salida en función de lo que el clasificador haya aprendido. Esto supone que los algoritmos clásicos basados en *Machine Learning* son fuertemente dependientes de la representación de las características que se escoja para entrenarlos[9].

Los algoritmos de *Deep Learning* sin embargo nos permiten omitir esa fase de extracción de características, porque han sido diseñados para extraer y abs-

Figura 2.3: Detección usando algoritmos basados en Deep Learning [12]

traer su propia representación de la información a través de la composición de varias transformaciones no lineales[9]. En nuestro caso, como el problema trata de clasificar imágenes y viendo los buenos resultados obtenidos en los últimos años en problemas de visión por computador[13], el algoritmo de *Deep Learning* escogido será el conocido como Redes Neuronales Convolucionales o CNNs. De esta manera, el flujo que se seguiría para realizar la clasificación de una imagen usando CNNs será similar al de la Figura 2.3. Podemos abstraer las CNNs como una caja negra que recibe una imagen de entrada, sin ningún tipo de preprocesamiento previo, y proporciona una etiqueta de salida.

Composición

Dicho esto, podemos dar una pequeña visión de qué hay dentro de esa caja negra. La arquitectura de una CNN se define como un conjunto de capas conectadas entre si. Hay varios tipos de capas[15] y cada una de ellas realiza una operación específica sobre los datos de entrada que recibe y proporciona unos datos de salida, los cuales que vuelven a ser los datos de entrada para la siguiente capa tal y como podemos observar en la Figura 2.4.

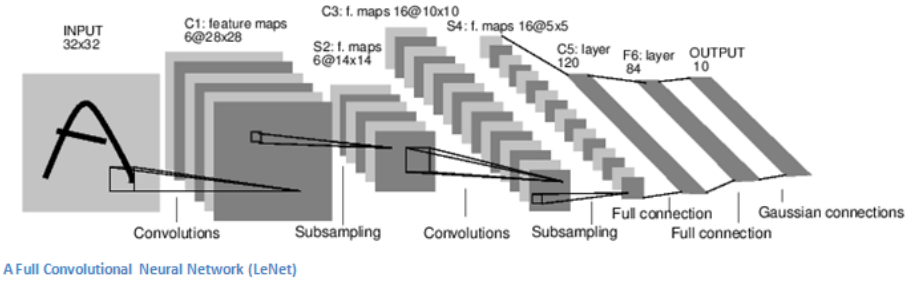

Figura 2.4: Arquitectura de Red: LeNet[13]

Funcionamiento

Ahora que conocemos de qué se compone una CNN vamos a ver a grandes rasgos cómo funciona una red neuronal convolucional. Cada una de las capas de las mencionadas en la sección anterior (2.3.2) se compone a su vez de

una serie de neuronas, las cuales tienen unos parámetros, que irán se irán ajustando (aprendiendo) en la fase de entrenamiento de la red, conocidos como pesos. Las neuronas de una capa, estarán solo conectadas con una pequeña región de la capa anterior. De esta forma, a medida que la red va aprendiendo, capas más avanzadas abstraen la información en mayor medida y aprenden patrones más complejos.
Dada una imagen de entrada, las neuronas de la red se van activando en función de los patrones aprendidos, y en la última capa calcula la puntuación de que la imagen pertenezca a cada una de las clases para las que la red fue entrenada, dando de esta forma, una etiqueta de salida. Hay muchos tipos de capas que pueden formar la arquitectura de red de una CNN. A continuación veremos un resumen de las más importantes y por norma general, las más usadas.

Capas de entrada [15] - Tal y como su nombre indica, recogen los datos n-dimensionales para la red. Suele ser la primera capa de nuestra arquitectura de red. En nuestro caso, los datos de entrada son las imágenes, que serían elementos tridimensionales teniendo de esta manera un vector del tipo $[ancho \times alto \times profundidad]$. La profundidad puede ser 1 o 3 dependiendo de si la imagen es en color (3 canales RGB) o en escala de grises (1 canal).

Capas de convolución [15] - Calcula la salida de neuronas que están conectadas a zonas locales de la imagen de entrada. El valor de cada neurona de salida es el producto escalar entre los pesos de las neuronas de la capa anterior y esa pequeña área local de la imagen. Podemos ver una representación gráfica en la Figura 2.5

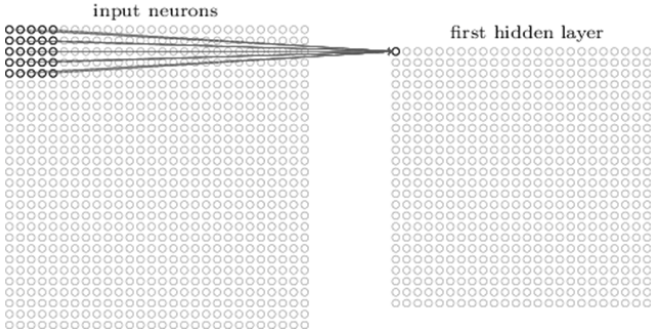

Figura 2.5: Filtro 5×5 convolucionando sobre un volumen de entrada[13].

Capas no lineales o Capas de activación[10, 14] - Por convenio, tras una capa de convolución se suele aplicar una capa de activación con el objetivo de añadir *no-linealidad* a un sistema que ha estado haciendo operaciones lineales durante las capas de convolución. Un ejemplo de

función no lineal, y una de las más usadas en capas no lineales es:

$$f(x) = max(0, x)$$

También conocida como *Rectified Linear Units* o función *ReLu* que transforma únicamente los valores negativos en cero.

Capas de submuestreo - Reduce el tamaño de la imagen a la mitad por norma general. Se pueden aplicar diferentes técnicas para realizar esta operación. Una de las más conocidas se denomina *maxpooling* y podemos ver como funciona en la Figura 2.6. Esta transformación actúa en las dimensiones espaciales de la imagen (Anchura y Altura).

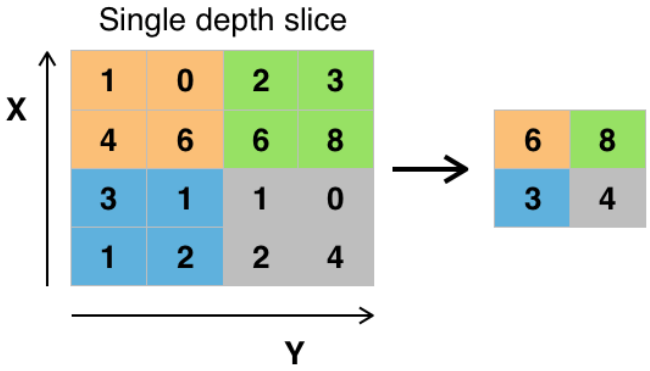

Figura 2.6: Reajuste de una imagen 4×4 a una imagen 2×2 usando *max-pooling*[14].

Capa totalmente conectada[15] - Esta capa calculará la puntuación de cada clase obteniendo una estructura de salida del tamaño $[1 \times 1 \times n]$ donde n es el número de clases de salida. En nuestro caso $n = 7$ (2.1).

Este ha sido un pequeño resumen acerca de lo que las redes neuronales convolucionales ofrecen. Es un tema muy extenso el cual no se puede cubrir de forma completa en este documento, además de no ser el objetivo de este. Si se desea saber más acerca de este tema recomiendo el siguiente libro [16].

Capítulo 3

Resolución del trabajo

Dado que el sistema que queremos construir está sujeto a experimentar, como método de ingeniería del software he empleado la técnica de modelo de prototipos rápido o también conocida como modelado de prototipado rápido[17].

3.1. Recursos

En esta sección enumeraremos los recursos que han sido utilizados para el desarrollo de este trabajo. Esto incluye recursos humanos como el autor y tutores, hardware y software utilizados así como las bases de datos.

3.1.1. Recursos Humanos

- **Autor**: Francisco José Fajardo Toril.
- **Tutor A**: Miguel García Silvente.
- **Tutor B**: Eugenio Aguirre Molina.

3.1.2. Recursos Hardware

Para el desarrollo de este trabajo he empleado mi ordenador personal cuyas especificaciones principales son:

CPU - AMD Phenom(tm) II X4 975

GPU - NVidia GTX 760

Memoria - 8GB RAM DDR3

3.1.3. Recursos Software

Para el desarrollo de la aplicación final, ha hecho falta usar distintas librerías que implementan de una forma eficiente las técnicas descritas en la sección anterior(2.3), así como distintos frameworks para desarrollar la interfaz de usuario por ejemplo. En la Figura 3.1 podemos observar los lenguajes de programación que han sido necesarios para la implementación de esta aplicación.

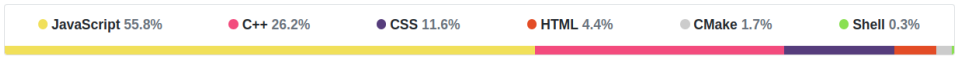

Figura 3.1: Lenguajes de programación que componen la aplicación[18].

OpenCV [19]

Por la facilidad de uso que me suponía debido a experiencia previa y su rapidez porque está implementada en C++ e usado esta librería para la manipulación general de imágenes.

Dlib [20]

Es una librería que implementa un gran número de algoritmos de Machine Learning en C++. En este caso ha sido usada sólo para el procesamiento de imágenes, pero solo he usado el detector de caras que implementa, el cual he introducido en la Sección 2.3, ya que presentaba menos falsos positivos que el detector de caras de OpenCV[11].

Caffe [21]

Es un framework de *Deep Learning* implementado en C++ aunque tiene una interfaz en Python. He escogido esta herramienta porque te permite construir tu propia red neuronal, definir tus propias capas de la arquitectura de red, posee buena documentación y buena comunidad. También puedes entrenar y testear tus modelos usando CPU o GPU, además de que ofrece herramientas que facilitan la creación de las bases de datos, todo esto desde la linea de comandos.

NVIDIA Deep Learning GPU Training System [22]

O también conocido como *NVIDIA DIGITS* es un sistema que funciona sobre *Caffe* y que ofrece una interfaz de usuario muy cómoda. Permite ahorrar mucho tiempo en la gestión de diferentes bases de datos, en la recolección de estadísticas de rendimiento de los modelos entrenados. Además

simplifica en gran medida la capacidad de entrenamiento con GPU, que mediante linea de comandos puede ser algo difícil de configurar.

Electron [23]

Este framework de código abierto permite crear aplicaciones de escritorio de forma nativa usando tecnologías web como son HTML, JavaScript y CSS. Esto nos permite compatibilidad con diferentes sistemas operativos y desarrollar una interfaz de usuario bonita de forma sencilla.

3.2. Especificación de requisitos

Decir que se partió de una especificación inicial de requisitos que a medida que se fueron implementando los prototipos se fue refinando posteriormente. Se puede poner la inicial y la final o solo la final indicando que se están poniendo los requisitos que finalmente tiene que tener el sistema.

Los requisitos se pueden referir a las necesidades del usuario del sistema (requisitos del usuario), a lo que tiene que hacer la aplicación (requisito funcional) o a cómo tiene que hacerlo (requisito no funcional). Ejemplo:

En este sistema un robot tiene que coger con sus pinzas un envase de medicamento y llevárselo a una persona anciana que por sí misma no puede recordar su medicación.

Requisitos del usuario:

RU1. La persona puede moverse libremente por una habitación donde está el robot.

RU2. La persona es capaz de coger el medicamento cuando se lo ofrece el robot.

RU3. La persona es capaz de tomarse el medicamento por sí misma.

Requisitos funcionales.

RF1. El robot mediante la cámara kinect debe poder localizar a la persona.

RF2. El robot conoce la posición de la mesa pues tiene un mapa de la habitación.

RF3. El robot debe identificar el medicamento correcto según un plan de medicación previamente establecido.

RF4. El robot debe poder coger el medicamento con sus pinzas.

etc...

Requisitos no funcionales

RNF1. El robot no puede atropellar ni dañar a la persona en ningún momento.

RNF2. La aplicación debe ejecutarse en entornos linux

RNF3. La aplicación debe utilizar pocos recursos para reaccionar con rapidez.

algo de la interfaz, como tratar posibles fallos, etc.

3.3. Planificación

Poner una tabla de tiempos con las planificación del proyecto diciendo cuando se tiene previsto alcanzar cada subobjetivo planteado. Con su correspondiente división en fases y tareas, y la posterior comparación con los datos reales obtenidos tras realizar el proyecto. Entre las fases está la realización de los diferentes prototipos I, II y III por ejemplo.

Poner presupuesto según horas de trabajo estimadas.

3.4. Análisis funcional

A partir de aquí nos referimos solamente al prototipo final que da lugar a la aplicación final.

Hay que describir la funcionalidad que debe poseer el sistema para poder cumplir con los objetivos y requisitos que se han dicho previamente.La descripción de esta funcionalidad puede hacerse analizando las tareas (que aparecerán en la planificación) y estudiando la inter-relación entre ellas y sus conexiones.

Para la realización de este análisis se pueden utilizar Diagramas de Flujo para poder conocer generalmente un único punto de inicio y un único punto de término o en varios.

`https://es.wikipedia.org/wiki/Diagrama_de_flujo`

Se pueden plantear también casos de uso. Los diagramas de casos de uso sirven para describir la inter-relación entre el sistema y el usuario del mismo. Se pueden utilizar para plantear diferentes casos de interacción entre el robot y la persona y cómo tiene que reaccionar el sistema en cada caso.

`https://es.wikipedia.org/wiki/Diagrama_de_casos_de_uso`

3.5. Implementación y pruebas

Decir qué lenguaje de programación se ha utilizado y las tecnologías implicadas aunque se hayan comentado en el apartado de recursos. Justificar su uso (rendimiento, disponibilidad, etc.). Si se ha usado open source decirlo y explicar las ventajas.

Las pruebas se realizan para comprobar la verificación y validación del producto software. La verificación consiste es comprobar que el producto realiza lo que está programado, es decir la programación no tiene errores y funciona en todos los casos cumpliendo los requisitos. La validación tiene que ver con que cumpla con lo que espera el usuario.

Verificación y Validación: Conjunto de procesos de comprobación y análisis que aseguran que el software que se desarrolla está acorde a su especificación y cumple las necesidades de los clientes.

Verificación: ¿Estamos construyendo el producto correctamente? Se comprueba que el software cumple los requisitos funcionales y no funcionales de su especificación.

Validación: ¿Estamos construyendo el producto correcto? Comprueba que el software cumple las expectativas que el cliente espera Importante: Nunca se va a poder demostrar que el software está completamente libre de defectos

las pruebas que pueden utilizarse son muy diversas. Aconsejo centrarnos en pruebas de caja blanca y de caja negra.

Las pruebas de la caja blanca se centran en la estructura interna del programa para elegir los casos de prueba. El objetivo de estas pruebas consiste en probar todos los posibles casos de ejecución de la aplicación para comprobar que los datos se comportan de manera correcta internamente.

Decir que se han hecho las pruebas de caja blanca.

Las pruebas de caja negra son aquellas que se centran en las salidas y entradas de los módulos, sin atender a su comportamiento interno (comprobando mediante las pruebas de caja blanca). Las pruebas de caja negra garantizan la interconectividad entre los diferentes módulos de la aplicación, así como su correcto funcionamiento final.

Poner algunos casos de prueba de caja negra.

Capítulo 4

Conclusiones y trabajo futuro

Decir lo que se ha conseguido realizar comentando sus puntos fuertes y débiles. Decir si se han alcanzado los objetivos específicos y el general propuesto y en qué grado.

Indicar las asignaturas del grado más relacionadas con la ejecución del TFG y cómo el TFG ha ayudado a afianzar los conocimientos adquiridos en el Grado.

Valoración personal si se quiere.

Avanzar algunas líneas de trabajo futuro para solucionar las debilidades detectadas o para conseguir nuevas funcionalidades interesantes.

Capítulo 5

Bibliografía

Bibliografía

5.1. Artículos científicos

[1] R. Shbib, and S. Zhou. *Facial Expression Analysis using Active Shape Model*, School of Engineering, University of Portsmouth, United Kingdom 2015.

[2] B. Abboud, F. Davoine, and M. Dang. *Facial expression recognition and synthesis based on an appearance model*, Heudiasyc Laboratory CNRS, University of Technology of Compiegne, BP 20529, 60205 Compiegne Cedex, France 2004.

[3] T. F. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham. *Active Shape Models - Their Training and Application*, Department of Medical Biophysics, University of Manchester, Oxford Road, Manchester M13 9PT, England 1994.

[4] T. F. Cootes, G. J. Edwards, and C. J. Taylor. *Active Appearance Models*, Wolfson Image Analysis Unit, Department of Medical Biophysic, University of Manchester, Manchester M13 9PT, UK 1998.

[5] J. Kumari, R. Rajesh, and KM. Pooja. *Facial expression recognition: A survey*, Dept. of Computer Science, Central University of South Bihar, India 2015.

[6] N. Dalal, and B. Triggs. *Histograms of Oriented Gradients for Human Detection*, INRIA Rhône-Alps, 655 avenue de l'Europe, Montbonnot 38334, France 2005.

[7] D. E. King. *Max-Margin Object Detection*, arXiv:1502.00046 [cs.CV], 2015.

[8] C. Cortes, and V. Vapnik. *Support Vector Networks*, Kluwer Academic Publishers, Boston. Manufactured in The Netherlands, 1995.

[9] Y. Bengio, A. Courville, and Pascal. Vincent. *Representation Learning: A Review and New Perspectives*, Department of computer science and

operations research, U. Montreal also, Canadian Institute for Advanced Research (CIFAR), 2014.

[10] V. Nair, and G. E. Hilton. *Rectified Linear Units Improve Restricted Boltzmann Machines*, Department of Computer Science, University of Toronto, Toronto, ON M5S 2G4, Canada 2010.

5.2. URLs

[11] D. King. *Dlib 18.6 released: Make your own object detector!*, URL: http://blog.dlib.net/2014/02/dlib-186-released-make-your-own-object.html, 2014.

[12] A. Moujahid. *A Practical Introduction to Deep Learning with Caffe and Python*, URL: http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/, 2016.

[13] A. Deshpande. *A Beginner's Guide To Understanding Convolutional Neural Networks*, URL: https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/, 2016.

[14] A. Deshpande. *A Beginner's Guide To Understanding Convolutional Neural Networks - Part 2*, URL: https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/, 2016.

[15] CS231n *Convolutional Neural Networks for Visual Recognition*, URL: http://cs231n.github.io/convolutional-networks/.

[16] M. Nielsen, *Neural Networks and Deep Learning*, URL: http://neuralnetworksanddeeplearning.com/.

[17] Ecu Red Conocimiento con todos y para todos, *Modelo de Prototipos*, URL: http://www.ecured.cu/index.php/Modelo_de_Prototipos.

[18] F. Fajardo, *Face Expression Detector*, Repositorio GitHub del proyecto, URL: https://github.com/FranFT/Face-Expression-Detector, 2017

5.3. Herramientas

[19] *OpenCV*, URL: http://opencv.org/.

[20] *Dlib*, URL: `http://dlib.net/`.

[21] *Caffe*, URL: `http://caffe.berkeleyvision.org/`.

[22] *NVIDIA DIGITS*, URL: `https://developer.nvidia.com/digits`.

[23] *Electron*, URL: `https://electron.atom.io/`.

5.4. Bases de datos

Capítulo 6

Anexo

Al final de la memoria hay que añadir un anexo de una página o dos, explicando como se usa el software a modo de manual de usuario. Es decir, como se llaman los comandos, qué parámetros hay que darle, como se llaman los ficheros de datos de entrada, etc.