GITI EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE GRADO

CONTROL DE UN COCHE TELEDIRIGIDO MEDIANTE RECONOCIMIENTO DE GESTOS

Autor: Marcos Ventosa Pontes
Director: Álvaro Sánchez Miralles

Madrid
Julio de 2019

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Marcos Ventosa Pontes

DECLARA ser el titular de los derechos de propiedad intelectual de la obra: Control de un coche teledirigido mediante reconocimiento de gestos, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente).*

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.
d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción

de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 18 de Julio de 2019

ACEPTA

Fdo……

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

....Control.....de.....una.....cadre.....teledirigido....mediante....

...reconocimiento.....de.....gestos.....

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2018/2019.... es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Marcos Santos Fecha: 18/07/2019

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Fecha: 18/07/2019

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GITI EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE GRADO

CONTROL DE UN COCHE TELEDIRIGIDO MEDIANTE RECONOCIMIENTO DE GESTOS

Autor: Marcos Ventosa Pontes
Director: Álvaro Sánchez Miralles

Madrid
Julio de 2019

CONTROL DE UN COCHE TELEDIRIGIDO MEDIANTE RECONOCIMIENTO DE GESTOS

Autor: Ventosa Pontes, Marcos.

Director: Sánchez Miralles, Álvaro.

Entidad Colaboradora: ICAI - Universidad Pontificia Comillas.

RESUMEN DEL PROYECTO

El objetivo principal del proyecto es comprobar la viabilidad de un vehículo controlado con el movimiento de las manos capturado con una cámara en un entorno industrial. Para hacerlo, se ha utilizado un coche teledirigido, al que se le ha sustituido el habitual receptor de radio control por un microcontrolador, éste genera las señales para controlar la dirección (servomotor) y la velocidad (ESC que controla el motor).

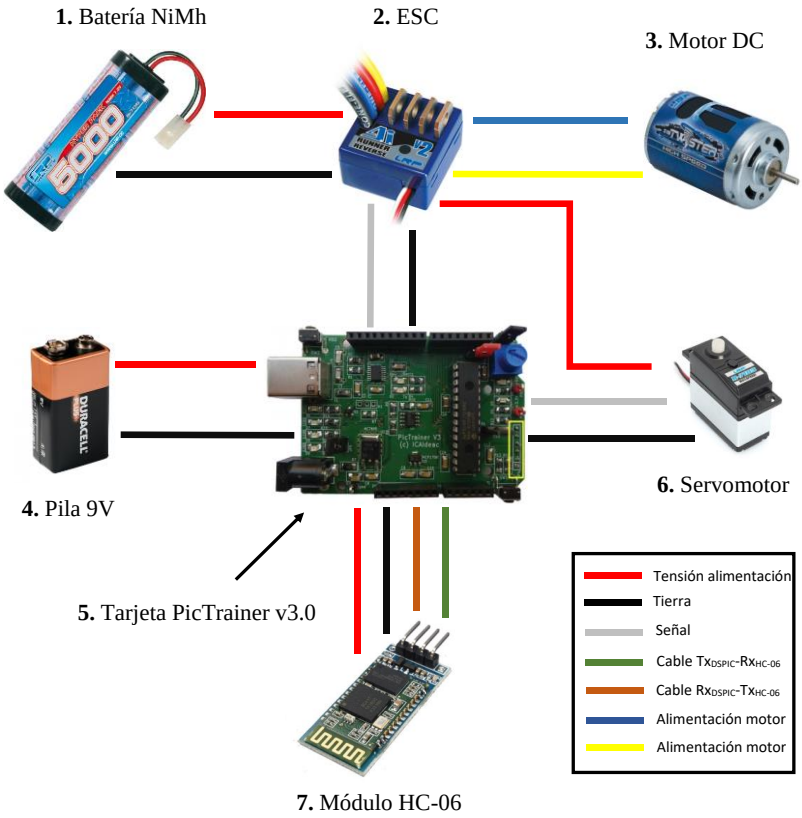


Figura 1 Esquema de las diferentes partes del coche RC

La información de velocidad y dirección le llegan al microcontrolador a través de un módulo bluetooth HC-06. Este módulo bluetooth está conectado al bluetooth de un ordenador. Es en este ordenador donde se procesa todo antes de enviarse la velocidad y la dirección al microcontrolador. El ordenador tiene conectada una cámara Kinect que ofrece una imagen de profundidad, la cual se usa para crear una máscara de la mano

aislada del entorno. La dirección se calcula en función de la inclinación de la mano (con el ángulo de los ejes de inercia).

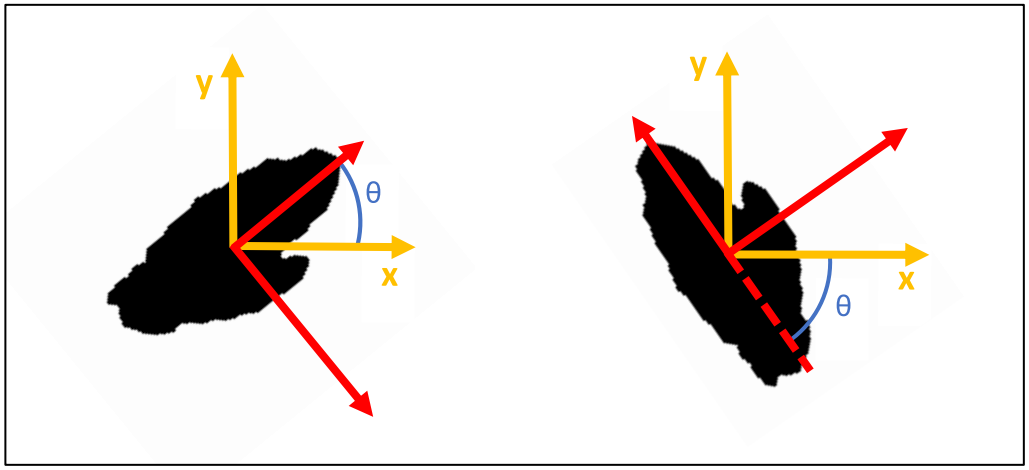


Figura 2 Ángulo θ de inclinación de los ejes de inercia

La velocidad en función de la distancia de la mano respecto de un valor de referencia.

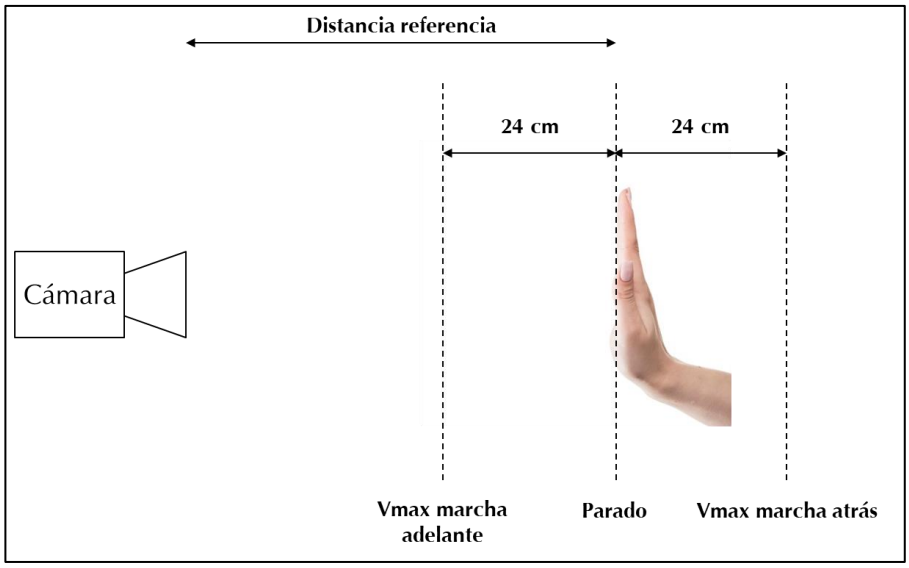


Figura 3 Velocidad del coche respecto de una distancia de referencia

Adicionalmente se han entrenado 2 clasificadores de machine learning (SVM) para identificar y reconocer 4 gestos distintos que hace la mano, un clasificador determina si se está haciendo un gesto o no y el otro determina qué gesto se está haciendo en caso de que el primer clasificador diga que se está realizando un gesto. Estos gestos se usan para apagar y encender el envío de información al vehículo.

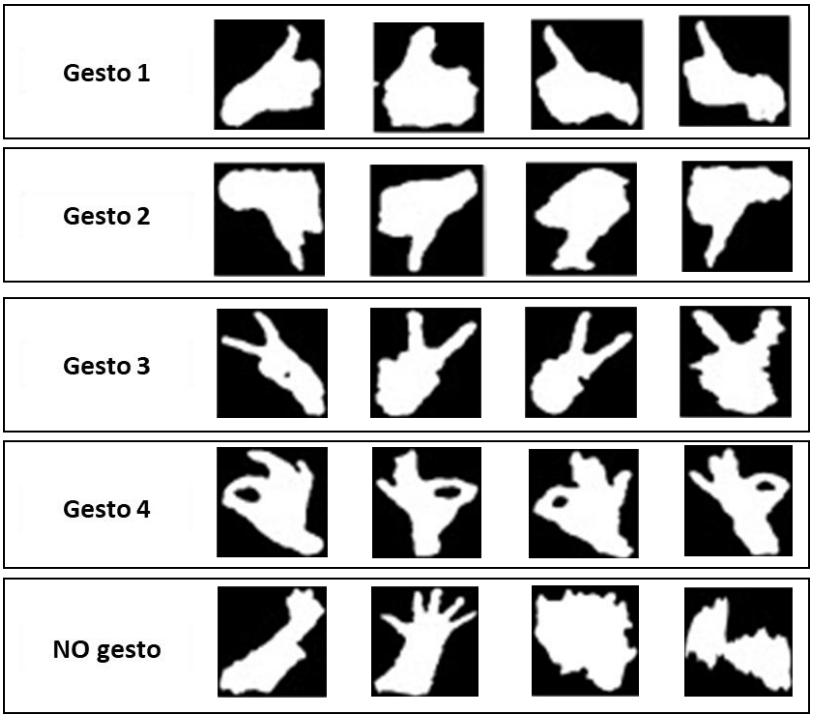


Figura 4 Muestras ejemplo con las que se entrenan los clasificadores

Para poder aplicar esta solución a un entorno industrial habría que hacer algunos cambios, uno de ellos sería cambiar la comunicación a una tecnología que permita más alcance (WIFI), también se debería usar otra cámara que se siga comercializando y pueda funcionar en exteriores.

CONTROL DE UN COCHE TELEDIRIGIDO MEDIANTE RECONOCIMIENTO DE GESTOS

Author: Ventosa Pontes, Marcos.

Director: Sánchez Miralles, Álvaro.

Associated Entity: ICAI - Universidad Pontificia Comillas.

PROJECT SUMMARY

The main objective of the project is to verify the viability of a vehicle controlled with the movement of hands captured with a camera in an industrial environment. To do so, a remote control car has been used, in which the usual radio control receiver has been replaced by a microcontroller, which generates the signals to control the steering (servomotor) and the speed (ESC that controls the engine).

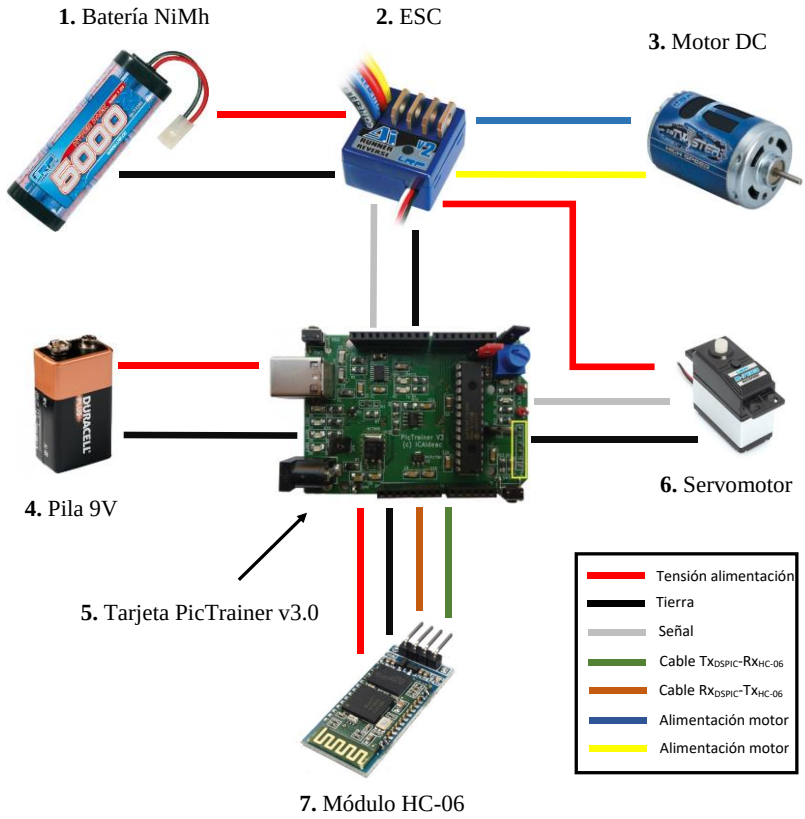


Figure 1 Scheme from the different parts of the RC car

The speed and steering information arrives to the microcontroller through a bluetooth module HC-06. This bluetooth module is connected to the bluetooth of a computer. It is on this computer where everything is processed before sending the speed and steering to the microcontroller. The computer has a Kinect camera connected that offers a depth image, which is used to create a mask of the hand isolated from the environment. The steering is calculated according to the inclination of the hand (with the angle of the axes of inertia)

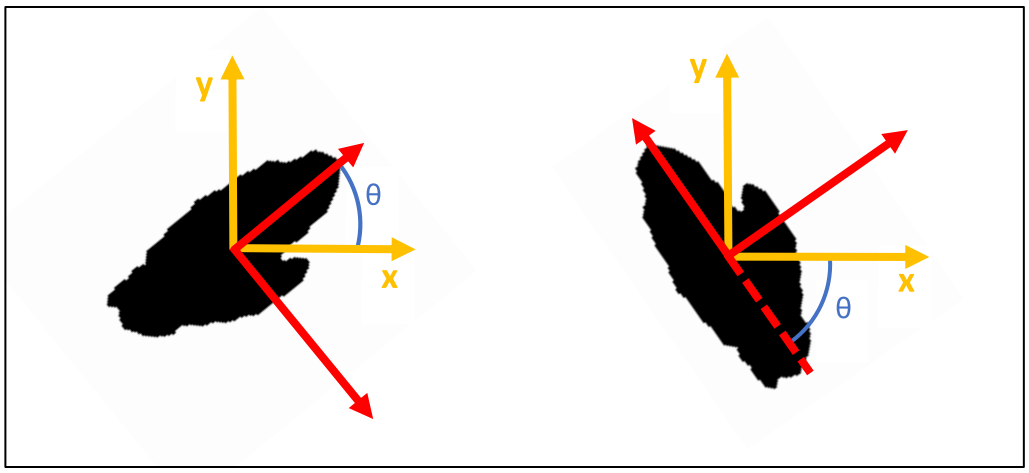


Figure 2 Angle θ from the axis of inertia

The speed as a function of the distance of the hand from a reference value.

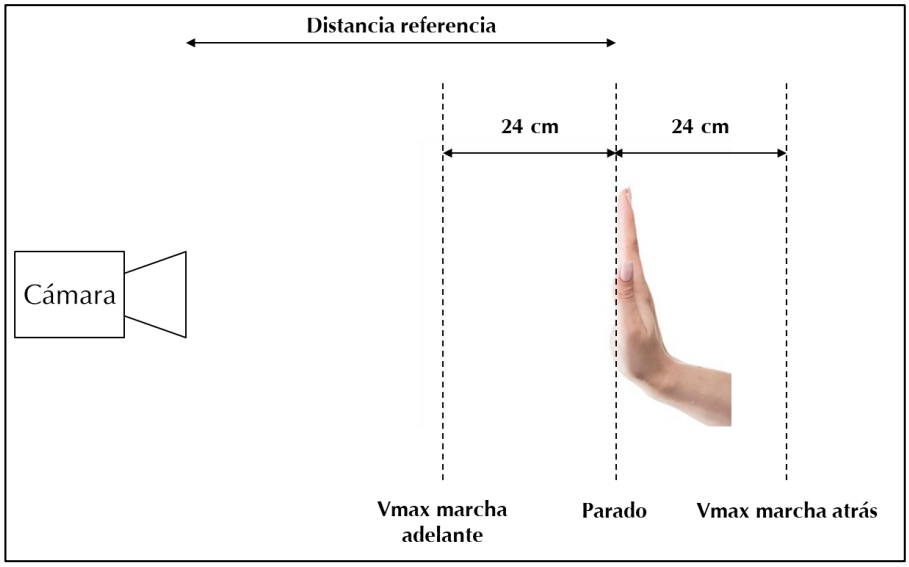


Figure 3 Car's speed as a function of the distance of the hand from a reference value.

In addition, two machine learning classifiers (SVM) have been trained to identify and recognize 4 different gestures made by the hand, one classifier determines whether a gesture is being made or not and the other determines what gesture is being made in case the first classifier says that a gesture is being made. These gestures are used to turn off and turn on the sending of information to the vehicle.

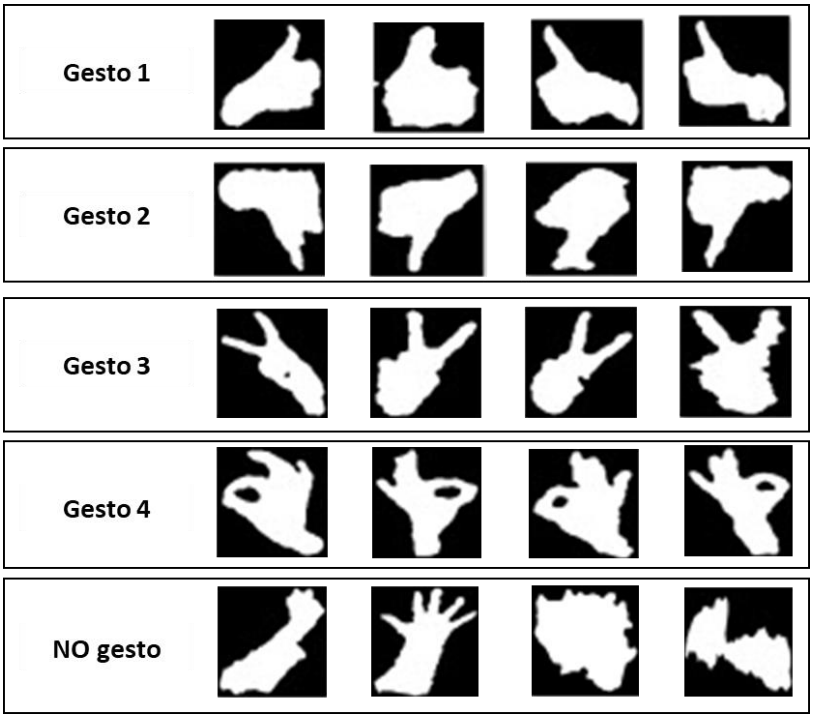


Figure 4 Some samples used to train the classifiers

To be able to apply this solution to an industrial environment, some changes should be made, such as changing the communication to a technology that allows more range (WIFI) and using another camera that is still commercialized and can operate outdoors.

1. Introducción

Con el auge de tecnologías que implementan visión artificial (caracteres, objetos, etc.) ya sea mediante inteligencia artificial o mediante otros métodos, parece interesante intentar implementar algunas de estas técnicas para llevar a cabo el control de un pequeño vehículo (coche teledirigido) mediante los gestos o la posición de las manos. Pese a que para la elaboración de este proyecto se usará un coche teledirigido, cabe destacar que se podría adaptar a cualquier vehículo motorizado, lo cual podría tener muchas aplicaciones en diferentes industrias como por ejemplo la construcción o fabricación.

1.1 Motivación

El control de vehículos a distancia con el cuerpo a través de una cámara sin necesidad de un contacto físico, es algo que no está muy extendido en la industria. Con este proyecto se desarrolla un sistema que usa la cámara Kinect para capturar la posición y orientación de una mano y traducirlo en órdenes para controlar un vehículo.

Un sistema de este tipo es muy portable ya que, en vez de usar unos mandos físicos específicos para controlar el vehículo, solo se necesita un ordenador y una cámara Kinect.

Con este proyecto se pretende comprobar la viabilidad de un sistema controlado con el movimiento de las manos capturado con una cámara en un entorno industrial. Un ejemplo podría ser un almacén en el que se controlan todos los vehículos necesarios para mover el stock de un lado a otro desde un puesto de observación y gestión

1.2 Objetivos

El proyecto se puede dividir en diferentes objetivos.

1.2.1 Implementación de un microcontrolador en un vehículo (coche teledirigido) para controlarlo.

Para llevar a cabo esta parte, se sustituirá el receptor de radio frecuencia de un coche teledirigido por un microcontrolador para controlar desde éste el servo de la dirección y el ESC (Variador de velocidad) que a su vez controla el motor del coche.

1.2.2 Comunicación del coche teledirigido con un ordenador mediante bluetooth

El microcontrolador montado en el coche, se conectará a un bluetooth que se comunicará con el bluetooth de un ordenador para que el microcontrolador reciba 'órdenes' desde el ordenador. Además, se diseñará un sistema de comunicación ya que solo se pueden mandar 2 bytes de información por el bluetooth. Cabe destacar que esto podría realizarse por wifi o 3G para aumentar el alcance y convertirlo en un proyecto más escalable.

1.2.3 Reconocimiento de localización y gestos de la mano

El ordenador se conectará a una cámara la cual, junto con un script, reconocerá los gestos y la localización de las manos.

1.2.4 Elaboración de un lenguaje de comunicación con los gestos

Habrá que establecer un sistema que relacione los gestos de las manos con las acciones con el coche. Tendrá que ser intuitivo y fácil de aprender.

2. Estado del arte

2.1 Vehículos controlados

Actualmente, el control de vehículos y robots controlados remotamente es algo frecuente. Muchas empresas de distintos ámbitos invierten en control remoto para desarrollar diferentes actividades. No obstante, es conveniente recordar que el control remoto de robots es una tecnología que lleva desarrollándose desde principios del siglo XX, ya que fue en 1903 cuando el ingeniero Español Leonardo Torres Quevedo presentó *telekino* en la Academia de las Ciencias de París, un robot que aprovechaba las ondas hertzianas Este fue el primer dispositivo que funcionaba a través de control remoto. A lo largo del siglo XX y como consecuencia de la carrera espacial de Estados Unidos y la URSS, junto con el desarrollo del televisor y su integración en la vida cotidiana, las tecnologías de control remoto se fueron modernizando y usando en ámbitos muy diversos, llegando hoy a estar presentes en la práctica totalidad de las industrias.

a) **Aplicaciones espaciales:** fueron las primeras aplicaciones en desarrollarse. Actualmente, éstas se materializan en el Curiosity, un vehículo de exploración en Marte desarrollado por la NASA. Sin embargo, a pesar de la desarrollada tecnología que posee, tarda 13 minutos en ejecutar una orden por la distancia a la Tierra, por lo que en muchas ocasiones se considera un vehículo de conducción autónoma más que un vehículo dirigido por control remoto.

Figura 5 Rover de Marte

No obstante, actualmente la NASA está desarrollando un sistema de control remoto llamado RAPID que se ha usado para robots caminantes o grúas robóticas,

aprovechando técnicas ya desarrolladas para la exploración de fondos marinos a través de *joysticks*. Aun así, la NASA no renuncia a los vehículos automatizados, ya que los nuevos vehículos serán controlados de forma remota por astronautas (marcándoles el objetivo), pero que, aun así, encontrarán la ruta más segura y se dirigirán hacia el punto señalado. [2]

b) **Aplicaciones en medicina:** el ejemplo más claro es el robot Da Vinci, que es una máquina capaz de realizar operaciones quirúrgicas de forma teleoperada. Este sistema cambia de forma radical el paradigma de la cirugía, no solo por las posibilidades que ofrece, sino por la propia disposición del quirófano durante la cirugía, ya que el robot se sitúa en el centro del quirófano y lo ocupa en su práctica totalidad, mientras que el cirujano se sitúa en una esquina dirigiendo los mandos, este robot trasmite los movimientos del cirujano, pero los mejora en precisión, ya que, a diferencia de las personas, éste no tiembla. Este robot está en pleno auge, ya que el hecho de poder ser controlado por un *joystick* simplifica muchos procedimientos reduciendo el tiempo de hospitalización, el sangrado y disminuye el dolor, además del hecho de que un cirujano puede realizar más cirugías seguidas con mejor éxito utilizando Da Vinci, ya que al simplificarse la tarea, el cirujano acaba menos fatigado la cirugía, lo que conlleva una reducción acusada del número de errores humanos cometidos, que son la principal causa de mortalidad de las cirugías menos invasivas y sencillas. Este robot realizó con éxito 877.000 cirugías tan solo en 2017.

Aun así, hay que tener en cuenta que Da Vinci no toma decisiones autónomas, tan solo reproduce los movimientos del cirujano, pero permite al cirujano operar como lo haría sin tener que recibir apenas formación específica.

Figura 6 Sistema Da Vinci operando

Otros robots que están comenzando a utilizarse en medicina son:

- **Senhance:** es un robot parecido al Da Vinci pero que tiene unas cámaras que muestran al cirujano el cuerpo por dentro, de esta forma no hay que hacer una incisión muy grande (necesaria para que el cirujano vea el interior del cuerpo) al operar, aunque el Senhance acaba de ser comercializado, un hospital italiano ya lo está usando.

- **Flex:** es una serpiente que sigue las curvas del cuerpo humano con instrumentos quirúrgicos y una cámara de alta definición.

- **Broca:** también conocida como la versión pequeña de Da Vinci, y más asequible, pero con la mejora de que aporta una sensación de tacto al cirujano. [3]

2.2 Visión artificial

La visión artificial abarca los diferentes métodos para adquirir, procesar y analizar imágenes.

Podemos diferenciar 2 formas de implementar la visión artificial:
- Reconocimiento a base de reglas (*rule-based approach*)
- Reconocimiento a través de *machine learning*

2.2.1 Rule-based approach

Esta forma de reconocer características se basa en el análisis de los pixeles de una imagen para poder extraer determinada información, y luego se evalúa si se cumplen diferentes reglas. La precisión de este método se basa en que las reglas (decididas por un humano) sean adecuadas.

Las primeras aplicaciones de *computer vision* se desarrollaron de esta manera debido a razones de velocidad de procesamiento.

Rule-based approach incluye entre otras características:

- Detector de bordes

- Detector de esquinas

- Reconocimiento de regiones

- Detector de movimiento

2.2.2 Machine learning

El reconocimiento y caracterización óptico a través de *machine learning* es un tema de actualidad. Existen muchas aplicaciones ya desarrolladas que se basan en esta técnica, aunque para desarrollar estas herramientas, se utilizan muestras que ya estén caracterizadas para que nuestra aplicación pueda aprender de ellas.

A los algoritmos de *machine learning* tradicionales en *computer vision*, como k-means, SVM o *Random Forrest*, se les suele introducir características que ya se hayan extraído de la imagen [4]

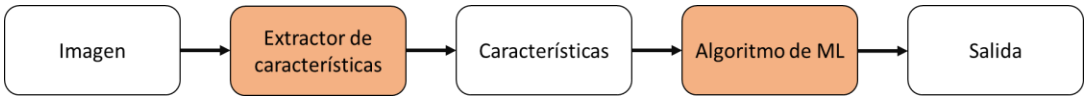


Figura 7 Flujo de machine learning tradicional

Los algoritmos de *machine learning* tradicionales suelen entrenarse con miles de muestras.

A continuación, se explica cómo funcionan algunos algoritmos para clasificación de imágenes:

1. **SVMs (Support Vector machines):** Este algoritmo de aprendizaje supervisado permite crear un modelo que determina la clase de una nueva muestra, a partir de muestras con las que se haya entrenado. Los SVM lineales trazan unas líneas(2D)/planos(3D)/hiperplanos(>3D) que separan las diferentes clases procurando dejar el espacio entre las clases y las líneas/planos/hiperplanos más grande posible.

   Los principales tipos de SVMs son:

   a. **Lineales:** Las clases se separan mediante líneas/planos/hiperplanos.

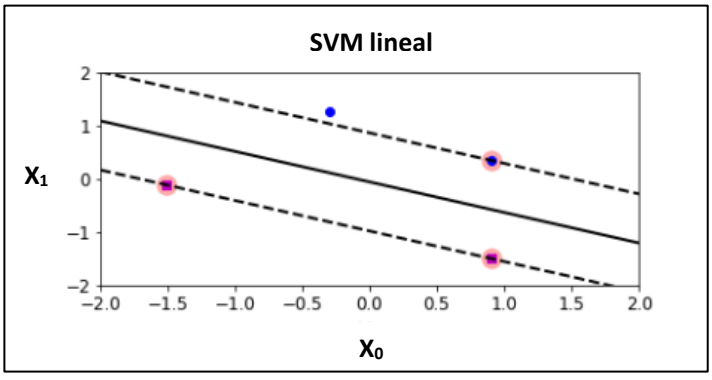


Figura 8 SVM lineal

b. No lineales:

- **Polinomial:** Las clases se separan con funciones polinómicas.

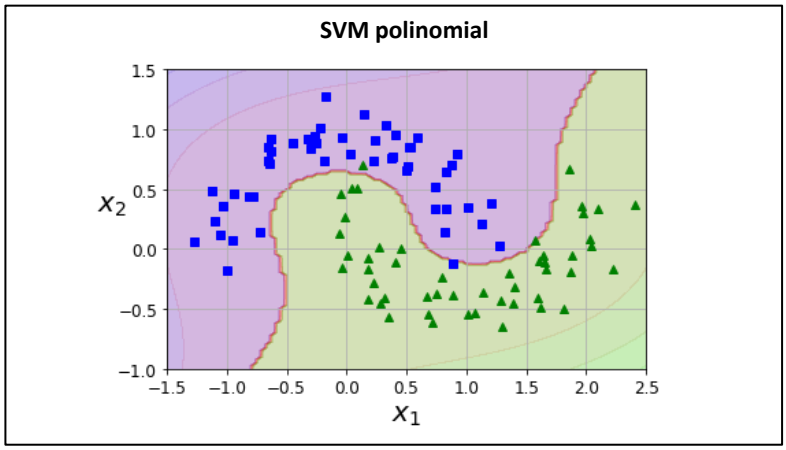


Figura 9 SVM polinomial

- **Gaussian Radial Basis Function (RBF):** Los RBFs son no lineales ya que añaden nuevas dimensiones usando una función gausiana esperando que estas nuevas dimensiones sean linealmente separables tal como se ve en la Figura 10 aplicado a un *dataset* de una sola dimensión. Al transformar la línea de las nuevas dimensiones a las dimensiones originales, ya no será una línea.

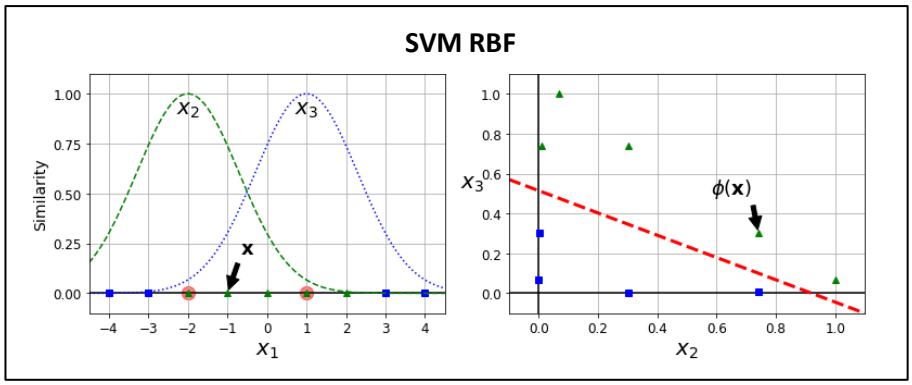


Figura 10 SVM RBF

Los SVMs tienen algunos hiperparámetros para tunear el modelo, los principales son:

a. **C:** Sirve para lineal, polinomial y rbf. C permite controlar el balance entre el espacio entre las clases y los límites de separación entre clases y el permitir *outliers.* Un C más pequeño permite un espacio más grande, pero más *outliers* y un C grande permite lo contrario.

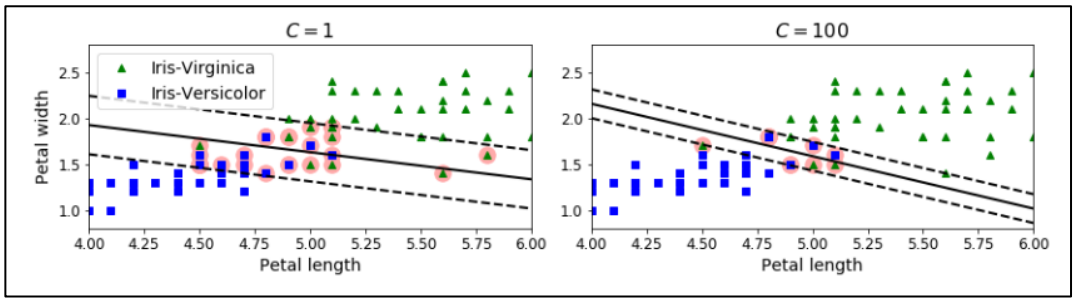


Figura 11 Efecto de C en SVM

b. **Gamma:** Sirve para polinomial y RBF. Gamma permite controlar cuanto se ajusta el modelo a los datos.

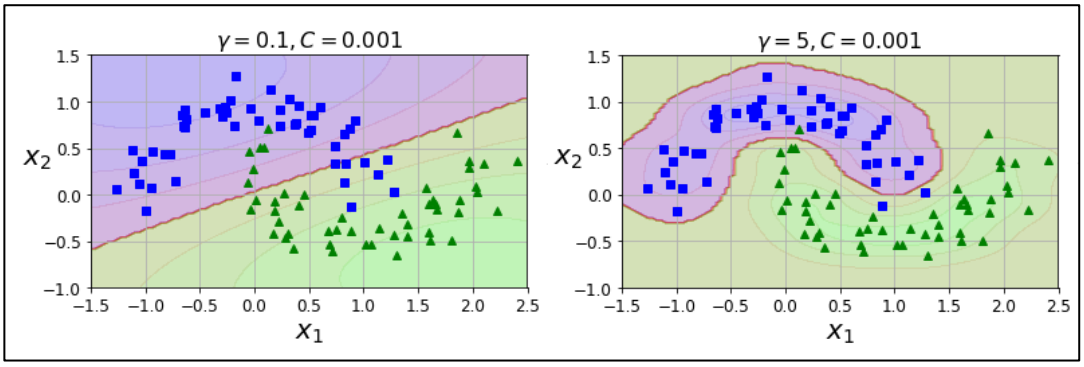


Figura 12 Efecto de gamma en SVM

c. **Grado:** Sirve solo para polinomiales. Determina el grado de la función polinomial.

2. **Random Forrest:** Random Forrest es un algoritmo de aprendizaje supervisado, utiliza árboles de decisión. Los árboles de decisión tienen diferentes nodos conectados y en cada nodo se toma una decisión en función de los valores de la muestra en diferentes dimensiones.

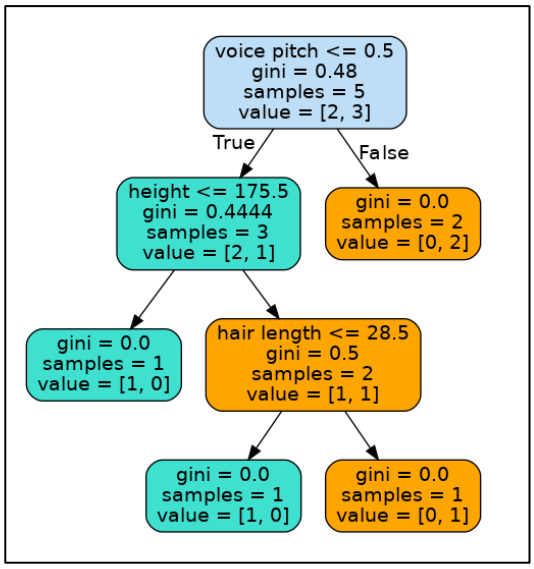


Figura 13 Árbol de decisión

Un Random Forrest está compuesto por más de un árbol de decisión, cada árbol tiene un cierto componente aleatorio para que los árboles sean distintos entre ellos.

Tienen algunos hiperparámetros para tunear el modelo, los principales son:

a. **Número de árboles:** Determina el número de árboles de decisión de los que el Random Forrest está compuesto.

b. **Número de hojas máximo:** Determina el número de nodos máximo que puede tener cada árbol.

3. **K-Nearest Neighbors:** Este algoritmo determina la clase, en función de las k muestras etiquetadas que estén más cerca de la muestra a etiquetar.

   Normalmente se usa la distancia euclídea para encontrar muestras cercanas, pero se pueden usar otros criterios.

   Este algoritmo cuenta con un hiperparámetro clave:

   - Número de vecinos: Este parámetro indica cuantas muestras cercanas se miran para clasificar la nueva muestra.

Para usar estos algoritmos para clasificar imágenes, cada pixel de la imagen es una dimensión que se introduce en el clasificador, por lo tanto, el modelo no conoce qué pixeles están cerca de otros pixeles, en cambio en *Deep learning* sí que se tiene en cuenta la relación de cercanía entre los pixeles. [6]

2.2.2.1 Deep learning

Deep learning es una rama del *machine learning*, esta rama ha cobrado una gran importancia en *computer vision* gracias al uso de CNN (*Convolutional neural networks* o redes neuronales convolucionales), las CNN son un tipo de red neuronal que tiene capas convolucionales, en estas capas se realizan operaciones de productos y sumas entre la capa de partida y un filtro que genera un mapa de características, la clave es que el mismo filtro permite extraer la misma característica de cualquier parte de la entrada.

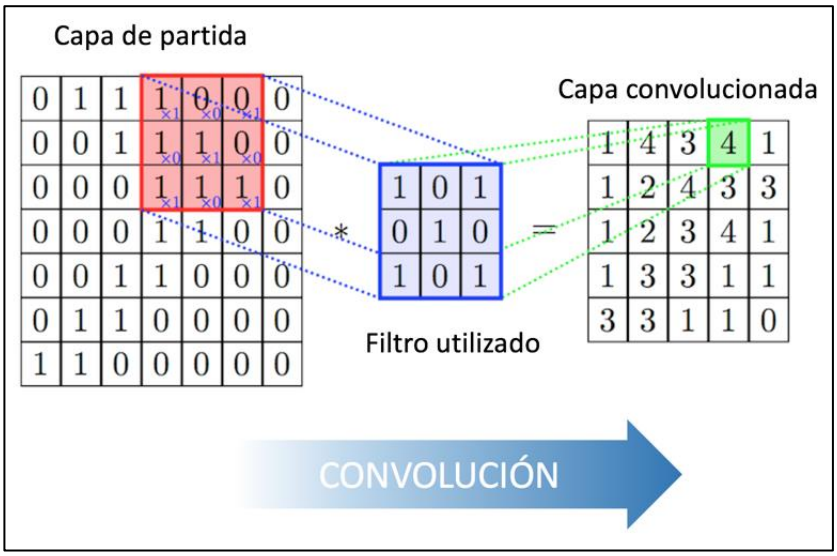


Figura 14 Funcionamiento de capa convolucional de una CNN

Las CNN también tienen una capa de reducción después de cada convolución (capa de *pooling*), en estas capas se disminuye la cantidad de parámetros al extraerse las características más comunes. La manera en la que se reducen los parámetros es mediante la extracción de estadísticas como el promedio o el máximo de una región.

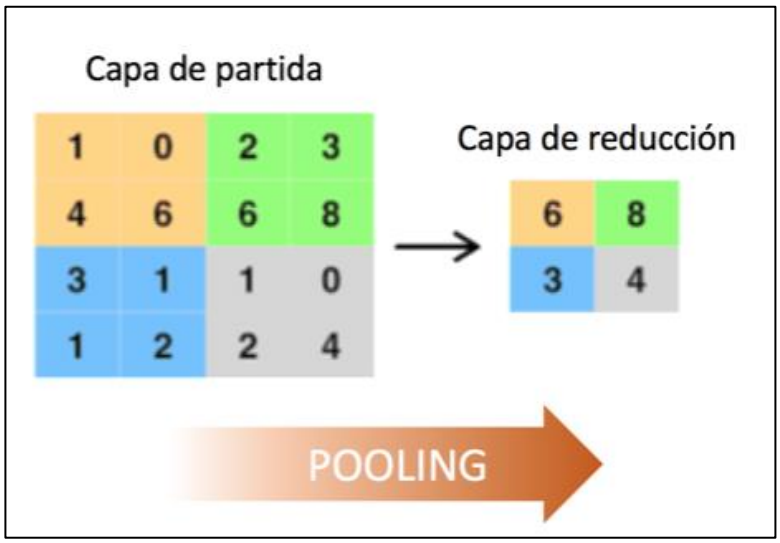


Figura 15 Funcionamiento de capa reductora de una CNN

Las ultimas capas de las CNN son capas totalmente conectadas, estas capas son las que clasifican la imagen. La última capa tiene tantas salidas como número de clases a predecir.

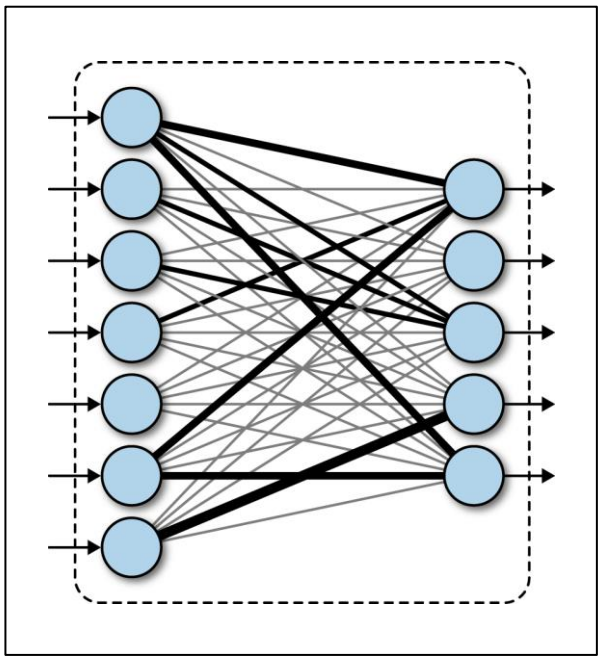


Figura 16 Capa totalmente conectada de una CNN

El flujo final de una CNN se puede ver en la Figura 17.

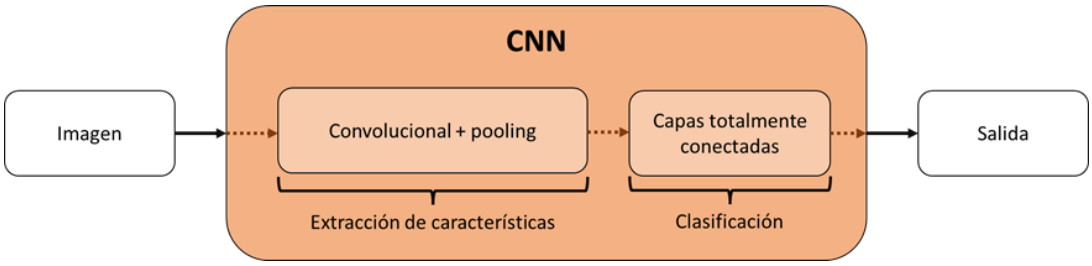


Figura 17 Flujo de una CNN

Las CNN necesitan ser entrenadas con un alto número de muestras (mayor que en los algoritmos de *machine learning* tradicionales) para conseguir una alta precisión. [5]

2.2.3 Lenguajes para *computer vision*

Actualmente los dos lenguajes más usados para *computer visión* son C++ y Python, posiblemente porque la librería OpenCV (librería más usada de *computer visión*) solo está disponible para Python y C++.

C++ es un lenguaje compilado mientras que Python es interpretado, lo cual hace que Python sea más lento que C++, aunque muchas librerías de Python están escritas en C++ por lo que son tan rápidas como C++ pese a usarse en Python. Además, existen formas de compilar código de Python (como se comentará más adelante).

Por el momento, Python es el lenguaje más utilizado para aplicaciones de *machine learning*.

2.3 Cámaras del mercado

Existen diferentes cámaras que proporcionan datos de profundidad en el mercado

1. **Kinect:** La Kinect V1 es un producto desarrollado por Microsoft para la videoconsola XBOX 360 en 2010. En el 2011 lanzó una versión para Windows. Más adelante se desarrolló la Kinect V2, que mejoraba algunas de las prestaciones de la V1. Microsoft ya no comercializa la Kinect. La Kinect V1 cuenta con una cámara RGB con una resolución máxima de 640x480 pixeles a 30 *fps* (*frames per second*), 4 micrófonos en diferentes zonas para poder percibir de donde procede el sonido, un motor para cambiar la inclinación, un acelerómetro, un proyector y un receptor de infrarrojos que, al estar algo separados actúan como una cámara de profundidad de 640x480 pixeles a 30 *fps* (perciben la profundidad de los objetos que están a más de 0.8 metros pero menos de 4 metros). En cambio, la Kinect V2 cuenta con una cámara RGB con una resolución máxima de 1920x1080 pixeles a 30 *fps* (*frames per second*), 4 micrófonos (igual que la V1) en diferentes zonas para poder percibir de donde procede el sonido, no cuenta con un motor para cambiar la inclinación, tiene un acelerómetro, un proyector y un receptor de infrarrojos que actúan como una cámara de profundidad de 512x424 pixeles a 30 *fps*. La luz solar puede afectar al funcionamiento del sensor de infrarrojos y por tanto esta cámara no funciona bien en exteriores.

   El precio de lanzamiento de la Kinect V1 para Windows fue de 250 dólares y la Kinect V2 para Windows 200 dólares. Actualmente no se pueden comprar a Microsoft pero existe un gran mercado de segunda mano (y algunos lugares donde los tienen en stock) cuyos precios son muy bajos. [7]

Figura 18 Kinect V1

2. **Orbbec Astra:** Orbbec Astra es una cámara desarrollada para aplicaciones de *computer visión*. Existen diferentes versiones (Astra Pro, Astra S y Astra), se comentarán las características de la Astra normal. Cuenta con una cámara RGB que proporciona una imagen de 640x480 a 30 *fps* (1280x720 a 30 *fps* en la Astra Pro) y una imagen de profundidad de 640x480 a 30 *fps* (percibe la profundidad de los objetos que están a más de 0.6 metros pero menos de 8 metros), también tiene 2 micrófonos. Funciona con un proyector y un emisor de infrarrojos, por lo tanto, en exteriores funciona mucho peor.

   Todas las versiones tienen un precio de 149.99 dólares. [8]

Figura 19 Orbbec Asra

3. **Intel Realsense:** Intel Realsense es un producto desarrollado por Intel para aplicaciones de reconocimiento facial, escaneado en 3D, reconocimiento del habla, reconocimiento de gestos… Existen muchas versiones, se comentarán las características de la versión D435. Cuenta con una cámara RGB que proporciona una imagen de 1920x1080 a 30 *fps* y una imagen de profundidad de 1280x720 a 90 *fps* (percibe la profundidad de los objetos que están a más de 0.1 metros, pero menos de 10 metros). A diferencia de las anteriores cámaras, que cuentan con un

proyector y un emisor de infrarrojos para capturar profundidad, Realsense calcula la profundidad viendo la disparidad entre las imágenes tomadas por dos cámaras separadas (como los humanos), por esta razón Realsense funciona tanto en interiores como en exteriores.

La versión D435 tiene un precio de 179 dólares, aunque hay una versión, la D415 (algunas características inferiores), cuyo precio es de 149 dólares. [9]

Figura 20 Intel RealSense D435

3. Arquitectura del proyecto

En esta sección se exponen las diferentes partes y recursos utilizados en cada parte del proyecto en cuanto a hardware y software centrándose en sus características y por qué se han elegido. En los capítulos siguientes se explicará en más detalle su función en el proyecto.

3.1 Hardware

Como ya se ha descrito anteriormente, este proyecto consta de 2 sistemas principales, el coche RC y el procesamiento en el ordenador. En esta sección se explica el hardware usado en cada una de las partes.

3.1.1 Hardware del Coche RC

El coche RC usado es el COCHE LRP EP 1/10 S10 TW BX 2WD RTR 2.4GHz, es un coche de la marca LRP, con escala 1/10 y tracción trasera. El coche posee entre otras características, una caja de piñones cerrada y un embrague tipo *slipper*.

Figura 21: COCHE LRP EP 1/10 S10 TW BX 2WD RTR 2.4GHz

Se seleccionó este coche por la facilidad que ofrece para acceder a sus diferentes componentes, lo cual es imprescindible para la conexión de los elementos de fábrica con los sistemas necesarios para el control del coche remotamente desde el ordenador. Además, al ser este coche relativamente simple sólo con tracción trasera, es más fiable al no necesitar conectar los dos ejes del coche.

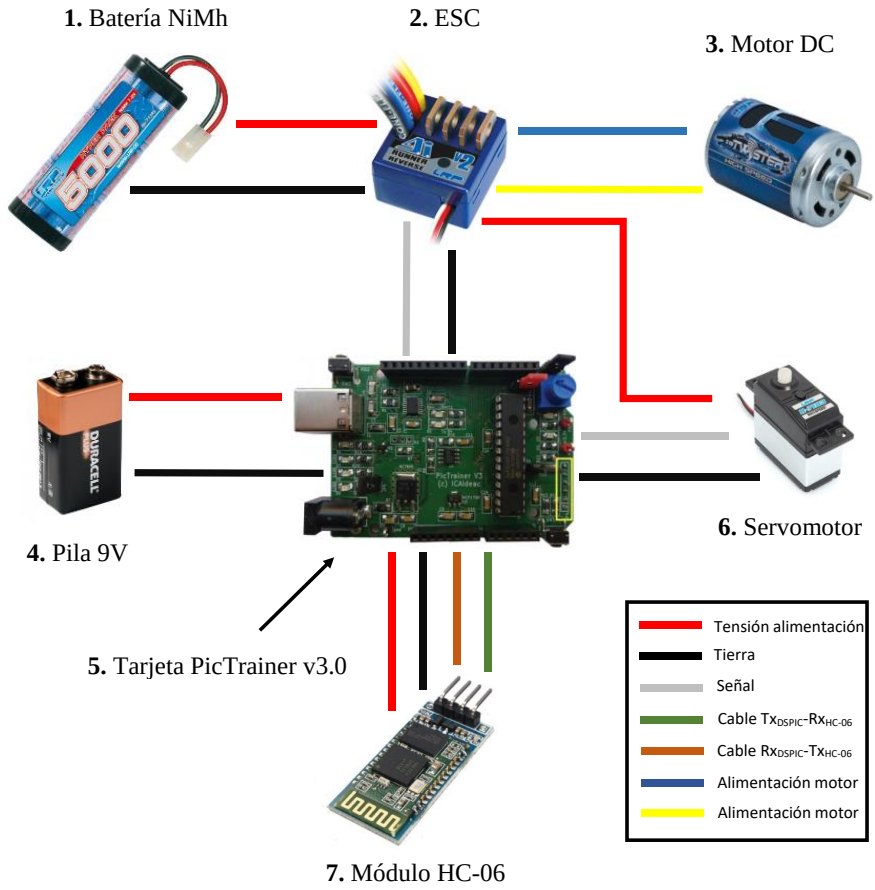


Figura 22 Esquema de las diferentes partes del coche RC

1. **Batería NiMh:** Batería tipo NiMh de 7.2 V y 5000 mAh, esta batería se encarga de alimentar el ESC.

2. **ESC (*Electronic Speed Controler*):** El variador de velocidad se alimenta de la batería y su función es controlar el Motor DC a través de la diferencia de tensión entre los terminales de alimentación. El ESC es comandado por el microcontrolador de la tarjeta PicTrainerv3.0 mediante un tren de pulsos.

3. **Motor DC:** Motor de corriente continua con escobillas que mueve el coche. El rotor del motor es alimentado por el ESC a través de las escobillas, al circular corriente eléctrica, se crea un campo electromagnético que interactúa con el imán del estator y crea un par que hace girar el motor. Si queremos cambiar el sentido, basta con cambiar la polaridad de la alimentación y para aumentar/disminuir la velocidad basta con aumentar/disminuir la diferencia de tensión de la alimentación. Este motor está conectado únicamente al par trasero del coche.

4. **Pila 9V:** Pila alcalina de 9 Voltios, alimenta la tarjeta PicTrainer v3.0.

5. **Tarjeta PicTrainer v3.0:** Esta tarjeta cuenta con 3 bloques funcionales diferenciados:

    a. **USB_uart (*Universal Asynchronous Receiver-Transmitter*):** Permite a la tarjeta comunicarse con un ordenador a través del puerto serie.

    b. **Alimentación:** Contiene toda la circuitería necesaria para alimentar la tarjeta, esta alimentación puede venir de diferentes fuentes, a través del USB, conectando una fuente externa al *jack* de alimentación o conectando una batería a un pin concreto del microcontrolador. La selección de la fuente de alimentación puede realizarse de forma automática o manual. Este bloque alimenta tanto el bloque USB_uart como el microcontrolador.

    c. **Microcontrolador**: Este bloque se alimenta con 3.3 voltios que salen del bloque de alimentación. El microcontrolador es un dsPIC33FJ32MC202. Además de los componentes imprescindibles, este bloque cuenta con un pulsador conectado a una entrada digital, un potenciómetro conectado a una entrada analógica y 4 diodos led conectados a salidas digitales, estos periféricos pueden desactivarse en caso de necesitar usar esos puertos. Este bloque también cuenta con un conector de programación, que conectado a una PKOB (PICkit *On-board*) permite programar el microcontrolador desde un ordenador.

6. **Servomotor:** Servo motor de DC que controla la dirección del coche. Este servomotor es alimentado por el ESC, que es a su vez alimentado por la batería de NiMh. El servo es comandado por el micro a través de un tren de pulsos que le proporciona el microcontrolador de la tarjeta PicTrainer v3.0.

7. **Módulo HC-06:** Este módulo permite la comunicación inalámbrica del coche RC con un ordenador. Se comunica con la tarjeta PicTrainer v3.0 a través de la UART (no el bloque USB_uart de la tarjeta PicTrainer v3.0, sino directamente a los pines Rx y Tx del microcontrolador). Funciona solo como esclavo. Este módulo es alimentado por el microcontrolador con una tensión de 5V (rango de operación 3.3V-5V). El módulo funciona por defecto a 9600 baudios.

3.1.2 Hardware del procesamiento en el ordenador

El hardware del procesamiento en el ordenador tiene 2 elementos fundamentales:

1. **Kinect V1 para Windows:** La Kinect es un producto desarrollado por Microsoft para la videoconsola XBOX 360 en 2010. En el 2011 lanzaron una versión para Windows, que es la que se usa en este proyecto. Más adelante se desarrolló la Kinect V2, que mejoraba algunas de las prestaciones de la V1. Microsoft ya no comercializa la Kinect V1. La Kinect V1 cuenta con una cámara RGB con una resolución máxima de 640x480 pixeles a 30 *fps* (*frames per second*), 4 micrófonos en diferentes zonas para poder percibir de donde procede el sonido, un motor para cambiar la inclinación, un acelerómetro, un proyector y un receptor de infrarrojos

que, al estar algo separados, actúan como una cámara de profundidad de 640x480 pixeles a 30 *fps*. Esta cámara de profundidad puede indicar con bastante precisión la distancia de los pixeles que estén a más de 0.8 metros, pero a menos de 4 metros. La razón por la que se ha decidido escoger la Kinect V1 para Windows frente a otras cámaras del mercado es por su precio más competitivo y por la gran comunidad de usuarios y desarrolladores que tiene.

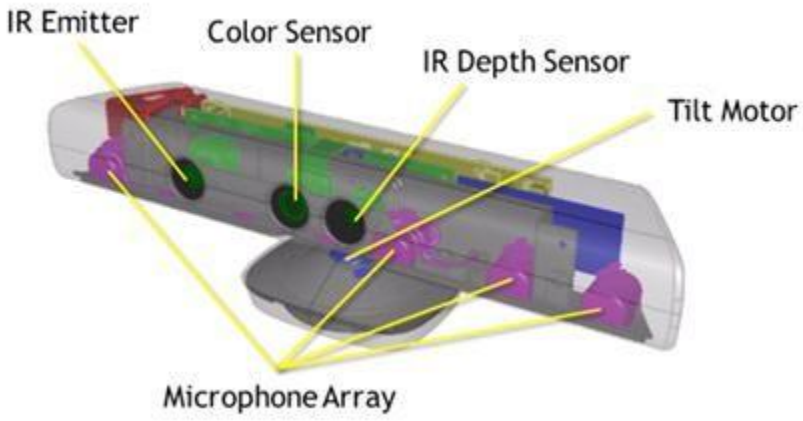


Figura 23 Figura con las diferentes partes de la Kinect V1 para Windows

2. **Ordenador**: El ordenador usado para procesar toda la información es un portátil Asus UX303UA del año 2016, cuenta con un procesador Intel Core i7-6500U a 2.5 Ghz, también tiene 8 GB de memoria RAM. Este ordenador cuenta con un modulo Bluetooth 4.0. Estas características son suficientes para cumplir las exigencias del proyecto.

Figura 24 Ordenador UX303UA

3.2 Software

A continuación, se detalla el software utilizado tanto en la parte del coche como en la parte de procesamiento en el ordenador.

3.2.1 Software del Coche RC

El software empleado para apoyar el desarrollo del sistema del Coche RC ha sido el siguiente.

1. **MPLAB X IDE:** MPLAB es un editor gratuito destinado para productos de la marca Microchip (Como el microprocesador dsPIC33FJ32MC202). Además de editar código, este IDE permite compilar el código y cargarlo al microprocesador (A través de una PKOB). Una característica fundamental de MPLAB es que permite *debugear* el código mientras está siendo ejecutado en el microprocesador para comprobar como cambian las variables, lo cual ha sido fundamental para el desarrollo del proyecto.

2. **C:** C es el lenguaje de programación usado para programar el microprocesador dsPIC33FJ32MC202. Es un lenguaje compilado de medio nivel muy potente y eficiente, lo cual permite hacer programas rápidos y compactos.

3. **Bitbucket:** Bitbucket es un servicio de alojamiento online, compatible con el sistema de control de versiones Git. Git fue desarrollado por Linus Torwalds (creador del *kernel* Linux) y ha permitido gestionar de forma fácil y eficiente diferentes versiones del proyecto en un repositorio.

3.2.2 Software del procesamiento en el ordenador

En esta sección se describe el software que ha apoyado el desarrollo de la parte del procesamiento en el ordenador.

1. **Bitbucket:** De la misma forma que con el microprocesador, todo el código correspondiente al procesamiento en el ordenador, se ha ido almacenando en un repositorio.

2. **Python 2.7:** Python ha sido el lenguaje elegido para el procesamiento en el ordenador. Es un lenguaje interpretado de alto nivel, orientado a objetos que ha crecido mucho en popularidad en los últimos años. La razón por la que se ha elegido Python es por su facilidad de uso, por la gran cantidad de librerías que le dan gran versatilidad y por la inmensa comunidad que lo respalda. La elección de la versión de Python ha venido obligada por una de las librerías usadas, Pykinect, de la que se hablará más adelante. Python 2.7 dejará de estar mantenido por los desarrolladores en enero de 2020.

3. **Sublime Text:** Sublime Text es el editor de texto que se ha usado para editar todo el código de Python. Se ha decidido usar este editor por su facilidad de uso y la baja cantidad de recursos que consume.

4. **Entorno virtual:** Con el objetivo de aislar el entorno de trabajo, se ha creado un entorno virtual que contuviera el intérprete de Python con la versión adecuada junto con las siguientes librerías.

    a. **Pykinect:** Pykinect es una librería desarrollada por Microsoft que permite controlar y comunicarse con la Kinect desde Python. Esta librería está muy poco documentada y los ejemplos de código de los desarrolladores están centrados únicamente en usarlo conjuntamente con una librería para crear juegos, lo cual ha hecho complicado el usar la información de la Kinect con otras librerías que interpreten los datos que proporciona la Kinect. La librería está únicamente implementada para funcionar con Python 2.7, lo cual ha forzado a usar esta versión en todo el proyecto.

    b. **OpenCV:** Esta librería orientada a la visión artificial fue originalmente desarrollada por Intel y es muy eficiente y rápida a la hora de desarrollar aplicaciones al estar programado en C y C++ y muy optimizado. OpenCV tiene muchas funciones y clases que abarcan diferentes áreas de la visión artificial. Esta librería ha sido fundamental ya que si el proyecto hubiera sido desarrollado puramente en Python no habría sido posible procesar los frames a 30 *fps*.

    c. **Numpy:** Numpy proporciona a Python soporte para vectores y matrices además de funciones para operar con ellos, Python por defecto solo tiene soporte para listas.

    d. **Numba:** Esta librería permite compilar código escrito en Python y de esta forma aumentar su velocidad drásticamente. Numba ha sido clave para desarrollar funciones particulares que no estuvieran implementadas en OpenCV pero que, de haberse escrito en Python sin compilarse, el procesado de *frames* se hubiera ralentizado mucho.

    e. **Math:** La librería Math da a Python diferentes funciones matemáticas como funciones trigonométricas, las cuales han sido fundamentales para analizar los *frames* y extraer información relevante.

    f. **Pyserial:** Esta librería le permite a Python controlar el puerto serie, lo cual ha permitido la comunicación del ordenador con el coche.

    g. **Struct:** Struct ha permitido convertir las variables que se querían enviar por el puerto serie a bytes para que pudieran ser comprendidas por el microprocesador. El uso de esta librería no hubiera sido necesario si se hubiera usado Python 3 ya que Python 3 tiene por defecto una clase bytes mientras que Python 2 no.

h. **Scikit-learn:** Esta librería de *machine learning* aporta a Python diferentes algoritmos de clasificación, regresión y análisis de grupos (*clustering*), los cuales han sido usados para identificar y diferenciar entre distintos gestos.

i. **Matplotlib:** Matplotlib permite la generación de gráficos para visualizar datos. Ha sido utilizado para comparar entre los distintos algoritmos de clasificación.

3.3 Comunicaciones

La comunicación entre ordenador y el módulo HC-06 se realiza a través de bluetooth y posteriormente el módulo pasará lo que le llegue a la tarjeta PicTrainer v3.0 a través de la UART.

El alcance máximo entre el coche y el ordenador viene dado por las limitaciones del bluetooth y por tanto está en unos 10 metros.

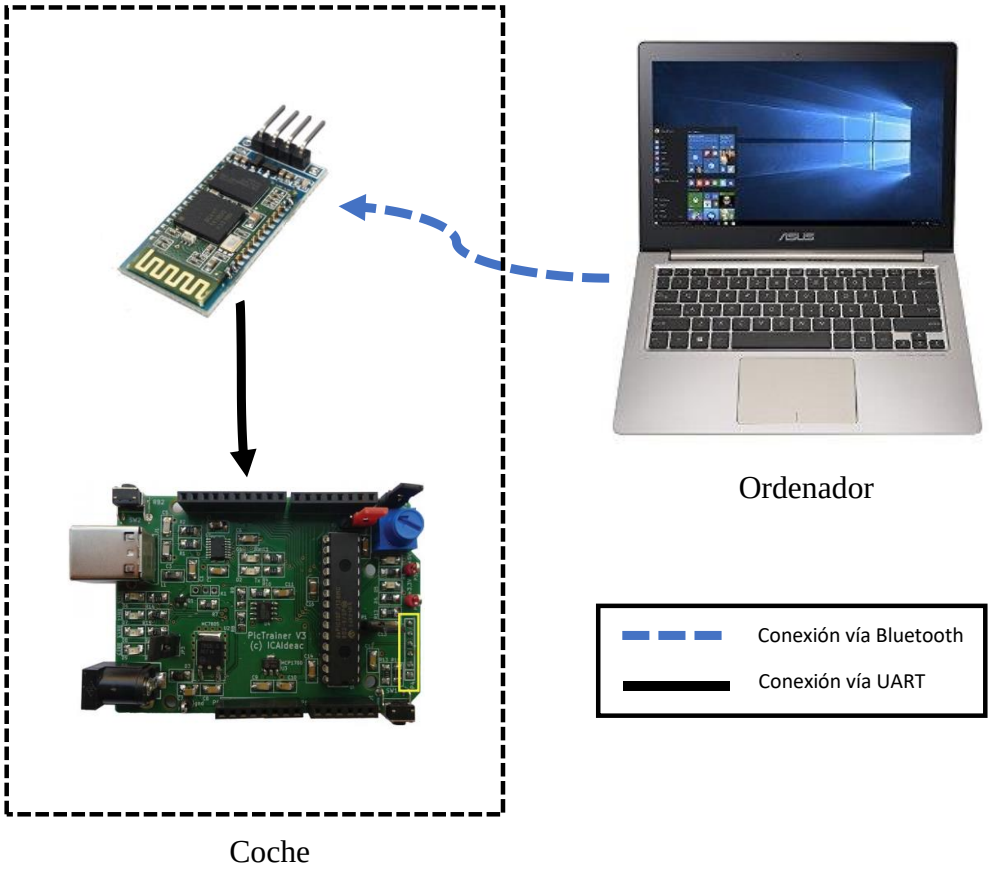


Figura 25 Esquema de comunicación entre ordenador y coche

La comunicación se ha configurado a 9600 baudios con un bit de *stop* y sin bits de paridad. La razón es que ésta era la configuración que el módulo bluetooth Hc-06 tenía de fábrica y dado que era suficiente para lo que se quería enviar se decidió no cambiarlo.

En vez de usar un módulo bluetooth, éste podría haber sido sustituido por un módulo wifi o un módulo 3G para permitir conexiones a largas distancias y darle más escalabilidad al proyecto, pese a esto, se ha decidido elegir el módulo bluetooth por su simplicidad y porque el uso de otro modulo bluetooth frente a otros que permitan conexiones a largas distancias, no cambia el propósito general del proyecto.

4. Hardware

A continuación, se explicará con detalle el desarrollo del coche.

4.1 Componentes del coche

Cronológicamente hablando el coche fue lo primero que se desarrolló. El desarrollo de esta parte se caracterizó por entender cómo funcionaba el coche RC de fábrica y cómo las diferentes partes interactuaban la una con la otra.

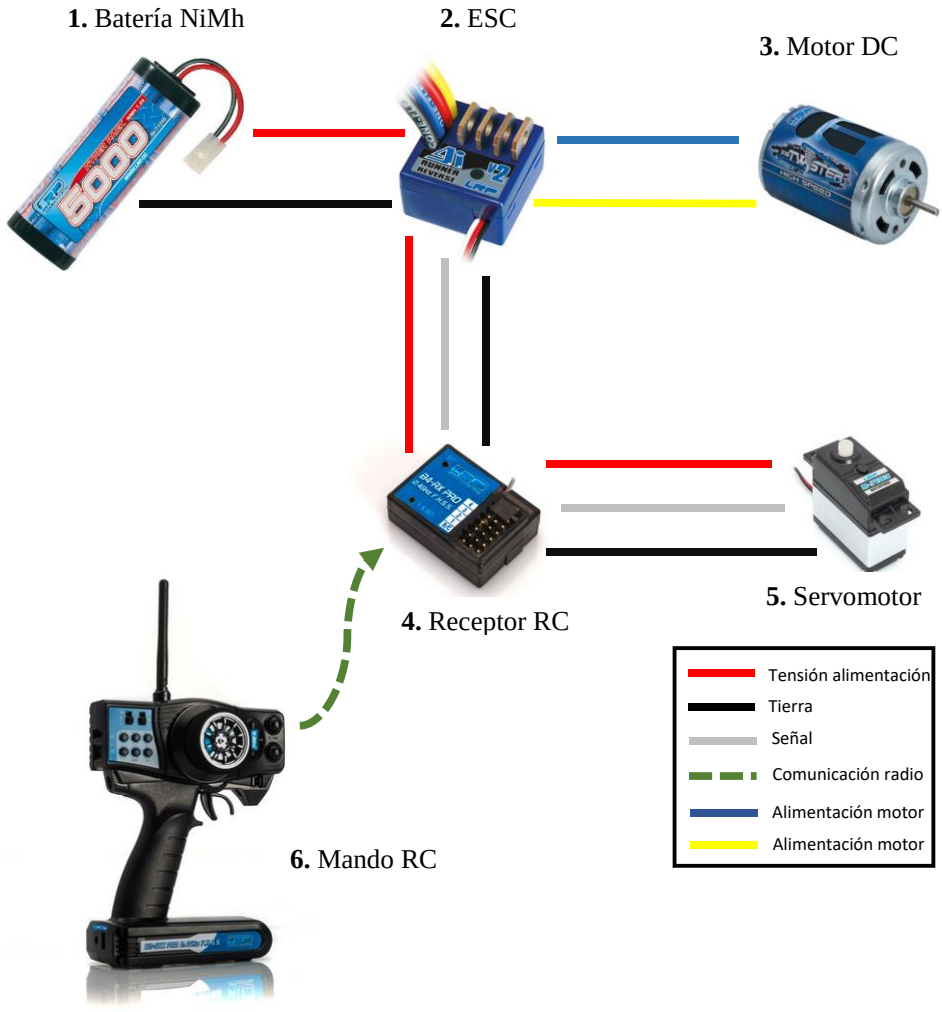


Figura 26 Esquema de los diferentes elementos del coche RC de fábrica

A continuación, se explica el funcionamiento de los diferentes elementos del coche RC de fábrica para posteriormente indicar qué se sustituyó.

1. **Batería NiMh:** Batería tipo NiMh de 7.2 V y 5000 mAh, esta batería se encarga de alimentar el ESC. Es la única fuente de alimentación de todo el coche (excluyendo el mando RC, que no forma parte del coche en sí) por lo tanto alimenta todos los componentes, aunque no estén directamente conectados.

2. **ESC (Electronic Speed Controler):** También conocido como variador de velocidad, este componente se alimenta de la batería y su función es controlar el motor de DC a través de la diferencia de tensión entre los cables de alimentación del motor. El ESC es comandado por el receptor de RC. Para comprender como el receptor RC controlaba el ESC hubo que hacer ingeniería inversa. Mientras se controlaba el coche RC con el mando, se midió con un osciloscopio las señales generadas por el receptor RC para controlar el ESC.

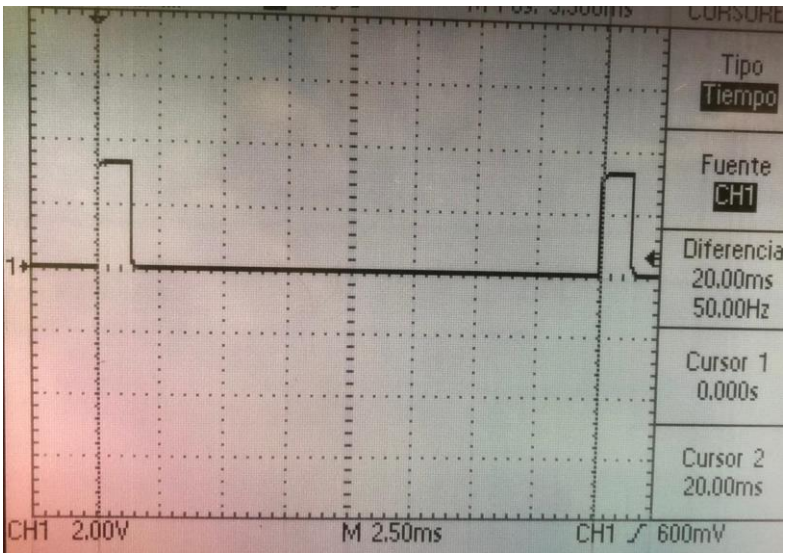


Figura 27 Periodo del tren de pulsos del ESC

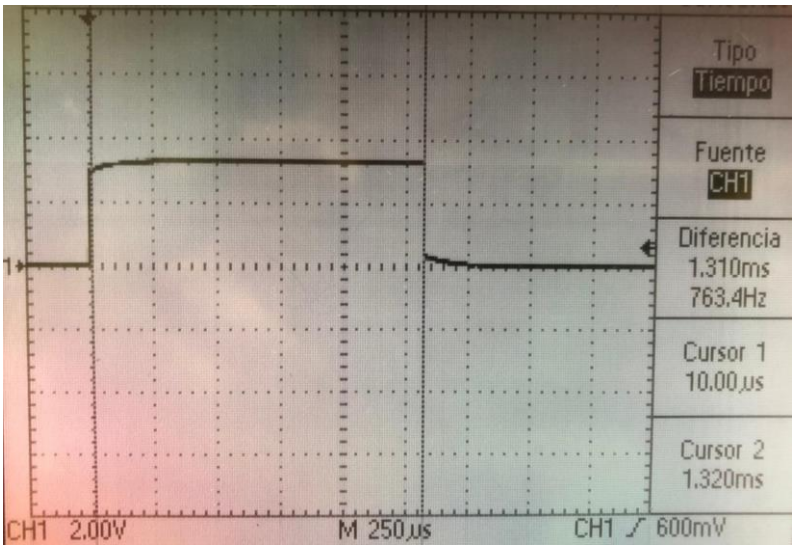


Figura 28 Ancho de pulso del ESC al estar el coche parado

- La Figura 27 y la Figura 28 muestra cómo es la señal cuando no se está presionando el “acelerador” desde el mando RC. Se puede ver que es un tren de pulsos con período de 20 ms y anchura de pulso de 1.3 ms.

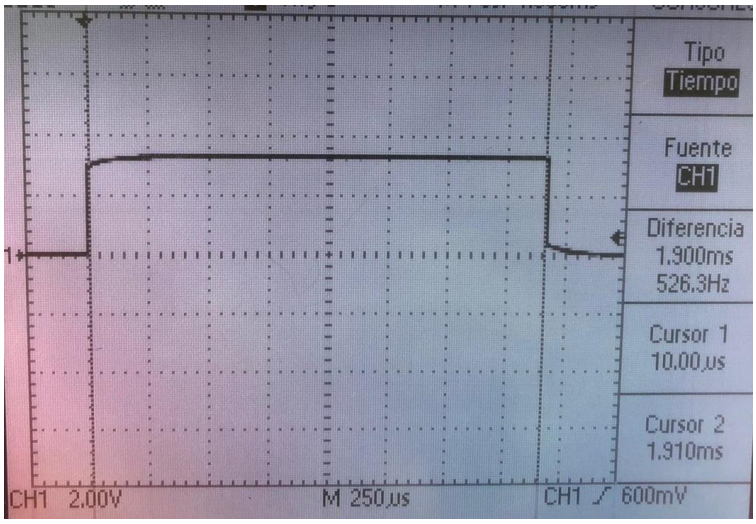


Figura 29 Ancho de pulso del ESC al ir a máxima velocidad hacia adelante

- La Figura 29 muestra cómo es la señal cuando se está presionando el “acelerador” al máximo para ir para adelante desde el mando RC. Se puede ver que es un tren de pulsos con periodo de 20 ms igual que el anterior y anchura de pulso de 1.9 ms.

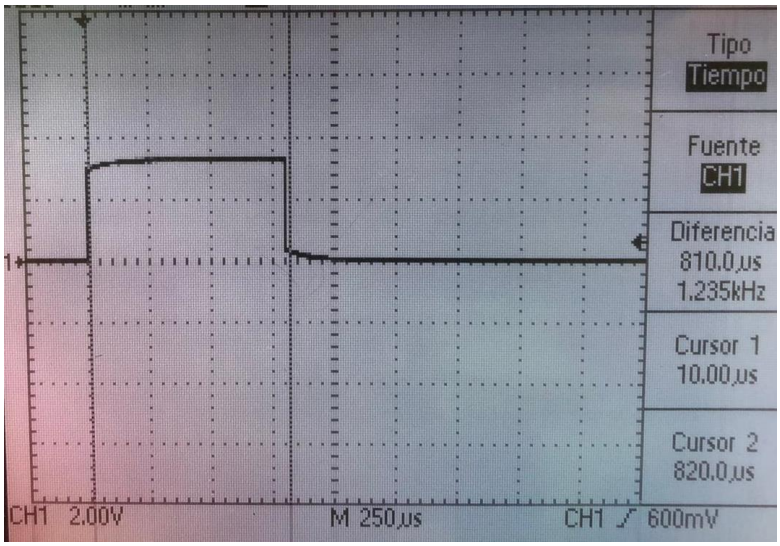


Figura 30 Ancho de pulso del ESC al ir a máxima velocidad hacia atrás

- La Figura 30 muestra cómo es la señal cuando se está presionando el “acelerador” al máximo para ir para atrás desde el mando RC. Se puede ver que es un tren de pulsos con periodo de 20 ms igual que el anterior y anchura de pulso de 0.8 ms.

Se pudo por tanto concluir que para controlar el motor de DC se tiene que enviar una señal al ESC de periodo 20 ms y cuya anchura de pulso varíe entre 0.8 ms para ir marcha atrás a máxima velocidad, 1.9 ms para ir marcha adelante a máxima velocidad y 1.3 ms para estar quieto. Aun así, se pudo comprobar que las velocidades se calculan relativas a la anchura de pulso inicial que se mande al ESC, por lo tanto, si inicialmente se manda un tren de pulsos de periodo 20 ms y anchura de pulso 1.5 ms el ESC tomará este valor como su referencia y el coche estará quieto cuando se mande un tren de pulsos con periodo 20 ms y anchura de pulso 1.5 ms.

Sabiendo esto, se puede afirmar que el motor de DC se puede controlar con una señal de periodo 20 ms y una anchura de pulso que varíe entre 1 ms para ir marcha atrás a velocidad máxima (más o menos), 2ms para ir marcha adelante a máxima velocidad, y 1.5 ms para estar quieto. Esta será la forma en la que se implementará.

Mientras se llevaban a cabo estas pruebas se vio que después de ir marcha adelante, si se comandaba la marcha atrás desde el mando, el coche no iba marcha atrás, sino que no se movía, pero el tren de pulsos sí que cambiaba. La razón de que esto ocurriera era por tanto provocada por el ESC, posiblemente el motivo es que es un coche RC de carreras y el ESC interpreta que cuando das marcha atrás después de estar acelerando, lo que realmente quieres es frenar.

3. **Motor DC:** Motor de DC con escobillas que controla la velocidad del coche. El rotor del motor es alimentado por el ESC a través de las escobillas, al circular corriente eléctrica se crea un campo electromagnético que interactúa con el imán del estator y crea un par que hace girar el motor. Si queremos cambiar el sentido, basta con cambiar la polaridad de la alimentación. Este motor está conectado únicamente al par trasero del coche.

4. **Receptor RC:** El receptor de RC recibe señales de radio desde el mando RC y genera las señales para controlar el ESC y el servomotor. El receptor RC es alimentado por el ESC con una tensión de 5 voltios.

5. **Servomotor:** Este componente permite controlar la dirección de las ruedas. Está alimentado por el receptor de RC. De la misma forma que se hizo con el ESC, para saber cómo era la señal con la que se controla el servomotor, se manejó el coche con el mando mientras se monitorizaba la señal con un osciloscopio.

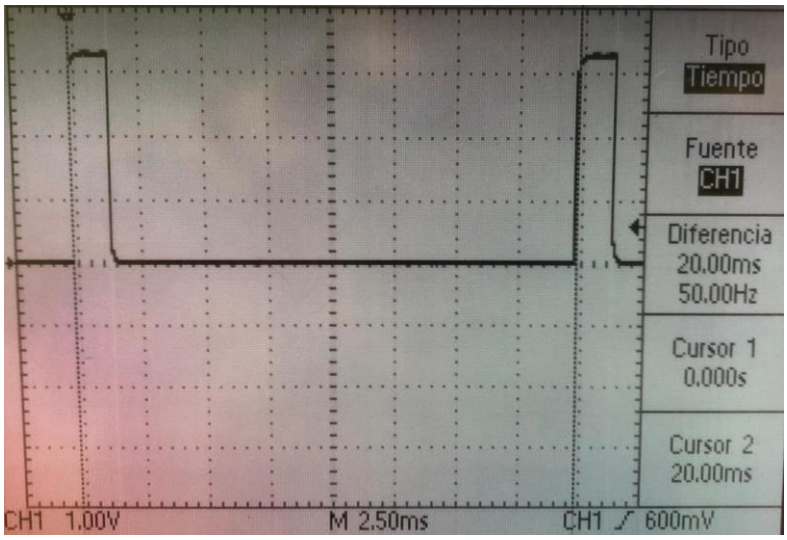


Figura 31 Periodo del tren de pulsos del servomotor

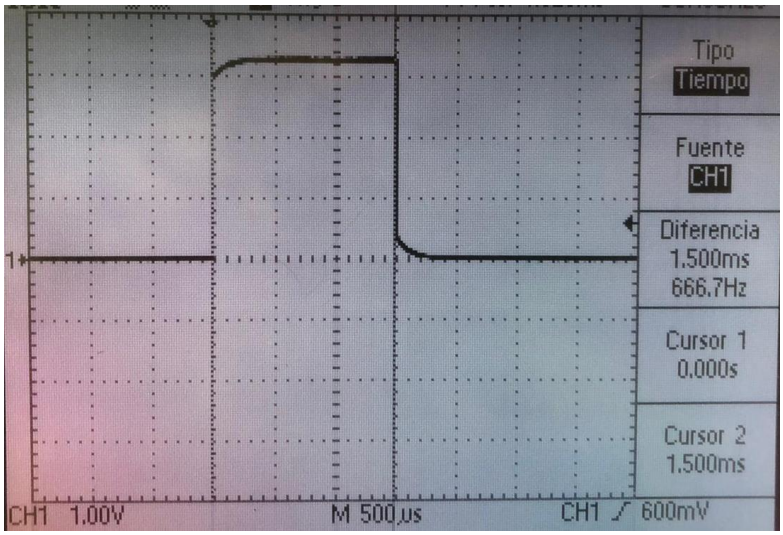


Figura 32 Ancho de pulso del servomotor al ir recto

- La Figura 31 y Figura 32 muestra cómo es la señal cuando no se está girando la dirección desde el mando RC. Se puede ver que es un tren de pulsos con periodo de 20 ms y anchura de pulso de 1.5 ms.

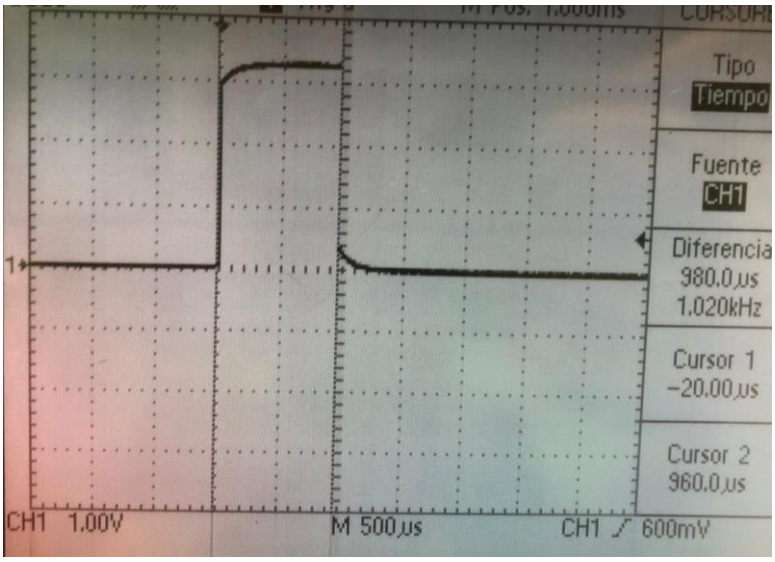


Figura 33 Ancho de pulso del servomotor al girar totalmente a la izquierda

- La Figura 33 muestra cómo es la señal cuando se está girando la dirección desde el mando RC totalmente a la izquierda. Se puede ver que es un tren de pulsos con periodo de 20 ms y anchura de pulso de 1 ms.

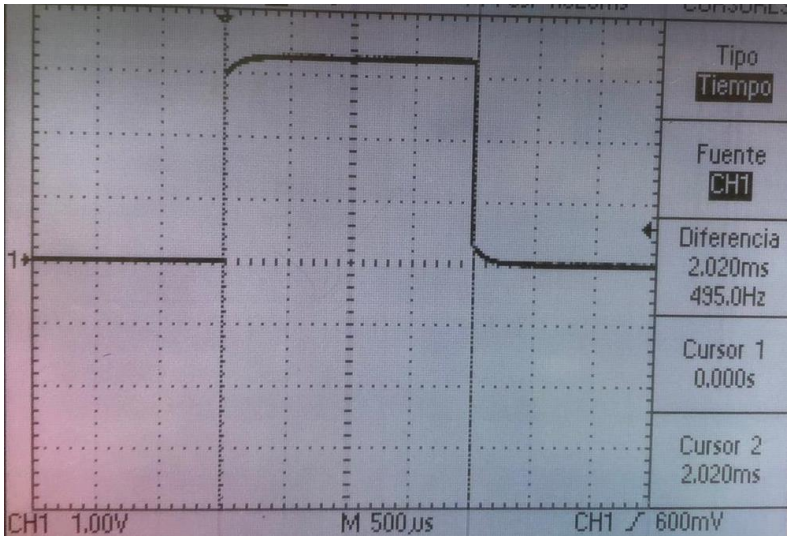


Figura 34 Ancho de pulso del servomotor al girar totalmente a la derecha

- La Figura 34 muestra cómo es la señal cuando se está girando la dirección desde el mando RC totalmente a la derecha. Se puede ver que es un tren de pulsos con periodo de 20 ms y anchura de pulso de 2 ms.

Al observar estos resultados, se puede ver que el tren de pulsos es igual al tren de pulsos para controlar el ESC, el tren siempre tiene un periodo de 20 ms y una anchura de pulso que varía entre 1ms para ir a la izquierda al máximo y 2 ms para ir a la derecha al máximo.

6. **Mando RC:** El mando RC manda señales de radio al receptor de radio que éste interpreta para generar las señales que controlan tanto el servomotor como el ESC. El mando RC se alimenta con unas pilas de 3 voltios.

Después de comprender cómo funcionaba el coche de fábrica para ser capaz de controlar el coche desde un ordenador, bastaría con sustituir el receptor de RC por un microcontrolador conectado a un módulo bluetooth tal como se ve en la Figura 22 y hacer que el microcontrolador genere los trenes de pulso para controlar el servomotor y el ESC en función de la información que recibe por el módulo bluetooth.

4.2 Programación del microcontrolador

A continuación, se explica cómo se ha programado el microcontrolador para poder controlar tanto el servomotor como el ESC.

Inicialmente se configuran el RB9 (señal del ESC) y el RB10 (señal del servomotor) como salidas digitales.

Para generar un código más modular se han creado una serie de *drivers* (Los *drivers* InicializarReloj y RemapeaPerifericos ya estaban creados):

1. **InicializarRelog:** Se encarga de inicializar el reloj del microprocesador, el cual tiene una frecuencia de 40Mhz.

2. **RemapeaPerifericos**

3. **InicializarMotores:** Para generar el tren de pulsos necesario para controlar el ESC, se utilizará el timer1, por tanto, se configura el timer1 del microcontrolador, poniendo su cuenta a 0, habilitando sus interrupciones con prioridad 4 (por defecto) y poniendo su *flag* a 0. Para conseguir un período en los pulsos de 20 ms se pondrá pre-escalado de 8, valor de fin de cuenta de 200 y un contador *cont1* (variable estática) que aumentará en 1 cada vez que el timer1 interrumpa y llegará hasta 500.

$$0.02 = \frac{1}{40 \times 10^6} \times 8 \times 200 \times 500$$

Ecuación 1 Calculo del periodo del pulso

   Para poder controlar el ancho del pulso se utilizará una variable *velocidad* (se recibe a través de la uart) y si cuando el timer1 interrumpa *cont1* es menor que *velocidad* la salida RB9 será 1 y en el caso contrario será 0. La variable *velocidad* podrá tener valores entre 25 (ancho de pulso de 1 ms y se iría marcha atrás a toda potencia) y 50 (ancho de pulso de 2ms y se iría marcha adelante a toda potencia). Si la variable *velocidad* toma el valor de 37.5 las ruedas no girarán.

$$0.001 = \frac{1}{40 \times 10^6} \times 8 \times 200 \times 25 \qquad 0.002 = \frac{1}{40 \times 10^6} \times 8 \times 200 \times 50$$

Ecuación 2 Calculo del ancho de pulso

```
void T1interrupt()
        Flag de interrupción del timer1=0

        if cont1 > velocidad
                RB9=1

        else
                RB9=0

        if cont1>=500
                cont1=0

        cont1++
```

Pseudocódigo 1 Pseudocódigo de interrupción del timer1

4. **InicializarServo:** Para generar el tren de pulsos necesario para controlar el ESC se utilizará el timer2, por tanto, igual que anteriormente con el timer1, se configura el timer2 del microcontrolador, poniendo su cuenta a 0, habilitando sus interrupciones con prioridad 4 (por defecto) y poniendo su *flag* a 0. Para conseguir un período en los pulsos de 20 ms se pondrá pre-escalado de 8, valor de fin de cuenta de 200 y un contador *cont1* (variable estática) que aumentará en 1 cada vez que el timer1 interrumpa y llegará hasta 500. Con esto se consigue un periodo de pulso de 20 ms como se ve en Ecuación 1 Calculo del periodo del pulso Ecuación 1.

   Para poder controlar el ancho del pulso, se utilizará una variable *dc* (se recibe a través de la uart), si cuando el timer2 interrumpa *cont2* es menor que *dc* la salida RB8 será 1 y en el caso contrario será 0. La variable *dc* podrá tener valores entre 25 (ancho de pulso de 1 ms y se giraría al máximo a la izquierda) y 50 (ancho de pulso de 2 ms y se giraría al máximo a la derecha) como se ve en la Ecuación 2. Si la variable *dc* tiene valor de 37.5 las ruedas irán recto.

   Dentro de la interrupción del timer2 también se gestiona una medida de seguridad. Para que en caso de que hubiera algún problema relacionado con las comunicaciones que impidiera que llegaran ordenes por la uart, el coche no siguiera haciendo lo último que le había llegado por la uart, cada vez que llega información a través de la uart se inicializa un contador *cont3* el cual va aumentando en uno cada vez que el *cont2* se resetea (cada 20 ms) y cuando el *cont3* llegue a 25 (tiempo equivalente a medio segundo) las variables *dc* y *velocidad* se igualan a 37.5, que sería su estado de reposo.

$$0.5 = \frac{1}{40 \times 10^6} \times 8 \times 200 \times 500 \times 25$$

Ecuación 3 Cálculo de tiempo máximo sin recibir ordenes

Por último, se mostrará el pseudocódigo de la interrupción del timer2

```
void T2interrupt()
        Flag de interrupción del timer2=0

        if cont2 > dc
                RB8=1

        else
                RB8=0

        if cont2>=500
                cont2=0
                cont3++

        if cont3>=25
                dc=37.5
                velocidad=37.5

        cont2++
```

Pseudocódigo 2 Pseudocódigo de interrupción del timer2

5. **InicializarUART:** Este *driver* se comentará en el apartado de comunicaciones dentro de realización del proyecto.

5. Software

Ya se ha hablado sobre cómo se controla el coche en función de 2 variables, *dc* y *velocidad* las cuales las recibe a través de la uart. Ahora se explica cómo se calculan estas variables en el ordenador a partir de la información que recibe de la cámara Kinect.

5.1 Kinect

La información de la Kinect llega al ordenador a través de un puerto USB. Pese a que la Kinect pueda proporcionar información desde sus altavoces, solo se usará la información proveniente de la cámara RGB y la cámara de profundidad.

La cámara RGB proporciona un vector de tamaño 1228800 que por cada *frame* es convertido a una matriz de 640x480x4, esta matriz contiene en la primera capa los valores de cada pixel de B, la segunda los valores de G, la tercera los de R y la última es una capa llena de ceros sin explicación aparente. Los valores de estas capas van de 0 a 255 (8 bits). Se recibe esta información 30 veces por segundo. La imagen que se recibe es simétrica a la realidad (como un espejo).

Figura 35 Imagen ejemplo cámara RGB

La cámara de profundidad proporciona un vector de tamaño 307200 que por cada *frame* es convertido a una matriz de 640x480x1, este valor se proporciona mediante un número de 16 bits, el más significativo es siempre 0 y los 3 menos significativos tampoco aportan información sobre la distancia. Los 12 bits restantes aportan la distancia en milímetros de cada píxel. Se necesita que las imágenes tengan una resolución de 8 bits (256 valores) por tanto para poder visualizar la imagen de profundidad se utilizarán los 8 bits más representativos de los 12 bits. Se recibe esta información 30 veces por segundo. Igual que en la cámara RGB, la imagen que se recibe es simétrica a la realidad (como un espejo).

Tal como se comentó en recursos utilizados, la Kinect solo proporciona datos de profundidad de los objetos que están a más de 0.8 metros y menos de 4 metros, para los pixeles que están fuera de este rango, la Kinect da un 0.

Figura 36 Imagen ejemplo cámara de profundidad (GrayScale)

En la Figura 36 se puede ver un ejemplo de la cámara de profundidad, los pixeles cercanos estarán oscuros y los lejanos más claros. Los pixeles que están muy cerca (<0.8 metros) o muy lejos (>4 metros) se ven negros, ya que su valor de profundidad es 0.

Se puede ver que existe una zona en negro a la derecha del contorno de la persona de la imagen. El motivo de esto es que son zonas de puntos ciegos.

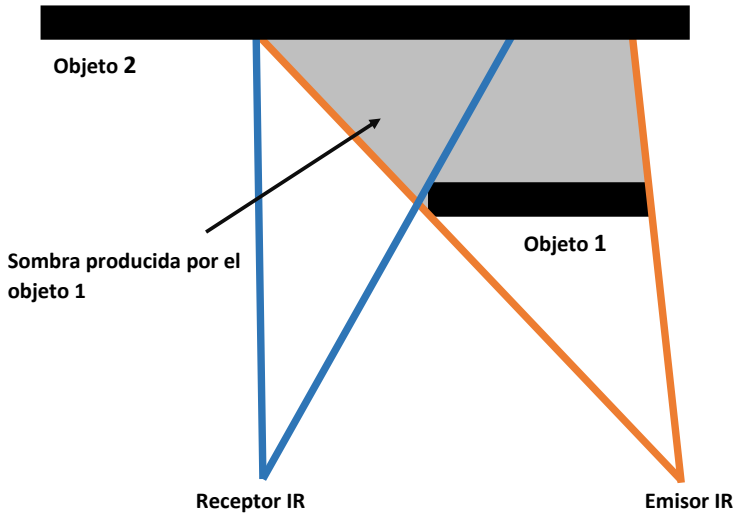


Figura 37 Puntos ciegos Kinect

Como se puede ver en la Figura 37, al estar separados el proyector de infrarrojos y el receptor, si tenemos dos objetos y el objeto 1 está por delante, se proyecta una sombra sobre el objeto 2 que el receptor de infrarrojos no puede captar. La razón por la que en la Figura 37 esta sombra está a la izquierda mientras que en la Figura 36 está a la derecha, es que la imagen que da la Kinect es una imagen simétrica a la realidad.

En ocasiones, si una forma está muy cerca del receptor y el emisor de IR y es suficientemente pequeña, esta zona de sombra parece que duplica la forma del objeto que está cerca, la razón de esto se puede ver explicado en la Figura 38. El objeto 1 proyecta una sombra sobre el objeto 2 y esto hace que se duplique el objeto 1 en vez de ser un contorno a la derecha, esto ocurre por el espacio del que si hay información de profundidad que el receptor ve entre la sombra del objeto 1 y el propio objeto 1. Se puede ver un ejemplo en la primera imagen de la Figura 40.

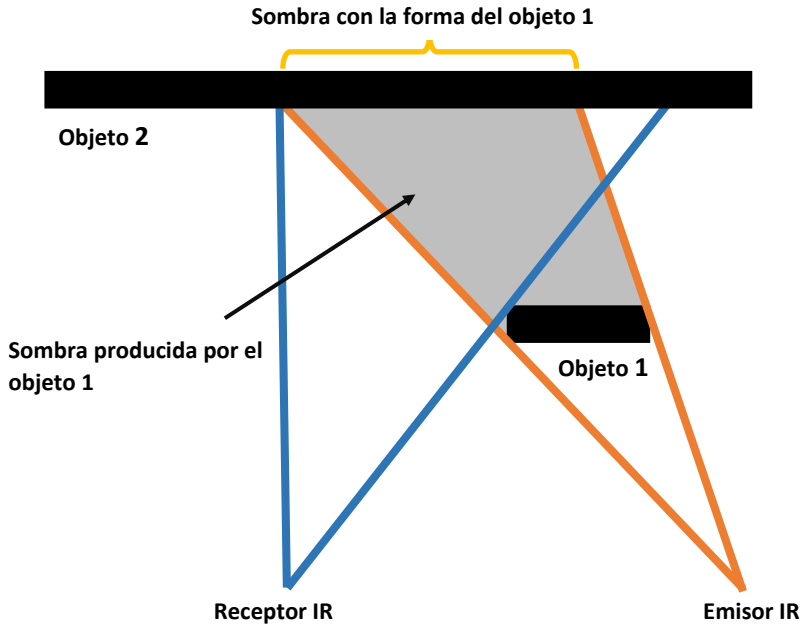


Figura 38 Puntos ciegos con sensación de duplicación

Como al ojo le es más difícil distinguir los cambios cuando se ve en una escala de grises, siempre que se visualice la imagen de profundidad se hará convertido a una escala multicolor.

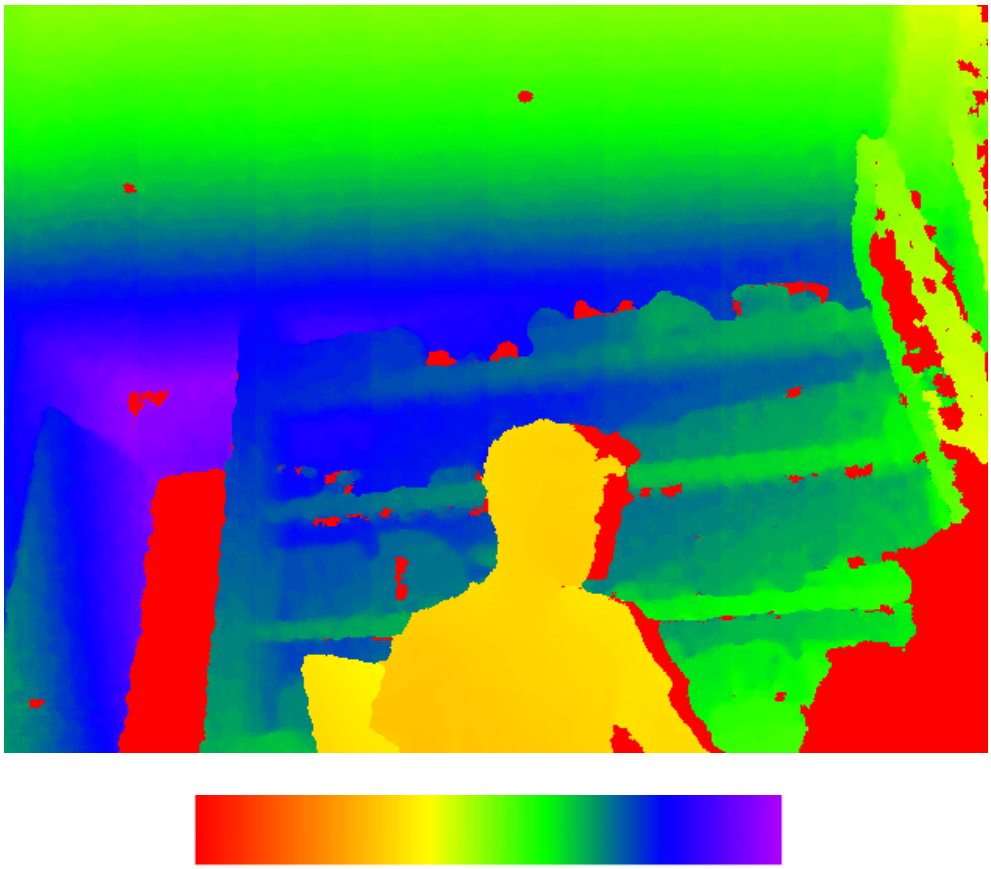


Figura 39 Imagen ejemplo cámara de profundidad (RainbowScale)

Se aprecia que los pixeles que están cerca tienen colores rojizos/naranjas y los lejanos azules/morados, los pixeles totalmente rojos son los que están muy cerca, muy lejos, o que son puntos ciegos.

5.2 Identificación de la mano

Una vez que tenemos una imagen de profundidad, podemos usar esos datos para diferentes fines, en el caso de este proyecto, se usará para aislar una mano del resto del entorno, ya que con una única mano será como se controle el coche. Para poder controlar el coche, habrá que tener la mano extendida hacia la cámara, de tal forma que se facilite el procesamiento de la imagen de profundidad.

Para conseguir aislar la mano, lo primero que se hace es pasar imagen de profundidad (primera imagen de la Figura 40) por una función que encuentra y devuelve el menor valor de profundidad en imagen que no sea 0 (porque esos 0 no representan la profundidad, sino que no se tiene valor de la profundidad, ya sea porque está muy lejos, o muy cerca o que están en una zona de sombra). Con esta profundidad mínima podemos obtener una máscara (valores de 1 y 0) de la imagen de profundidad que mantiene únicamente los pixeles que tienen una profundidad entre la profundidad mínima y la profundidad mínima más un pequeño valor, este valor se ha escogido de 64 mm después de probar diferentes valores. El resultado se ve en la segunda imagen de la Figura 40. Por último, se aplica un *median filter* a la máscara para suavizarla, este filtro sustituye el valor

de profundidad de cada píxel por el valor de la mediana de los 8 pixeles de su alrededor (tercera imagen de la Figura 40).

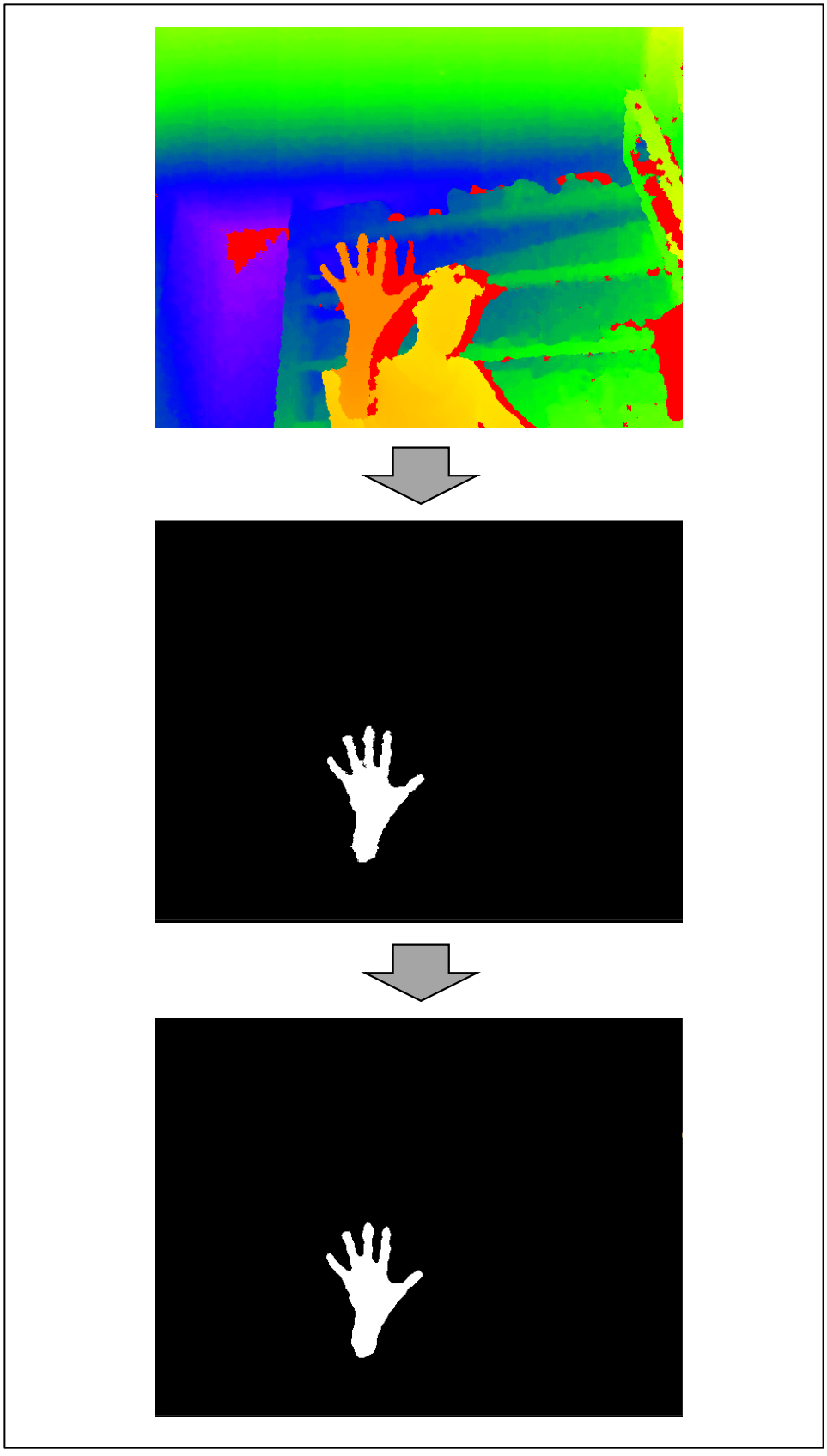


Figura 40 Proceso de identificación de la mano

Teniendo la mano aislada del entorno, se pasa esta máscara por una función de OpenCV que encuentra el contorno, esta función utiliza el algoritmo Suzuki85 [1]. El resultado se puede ver en la Figura 41.

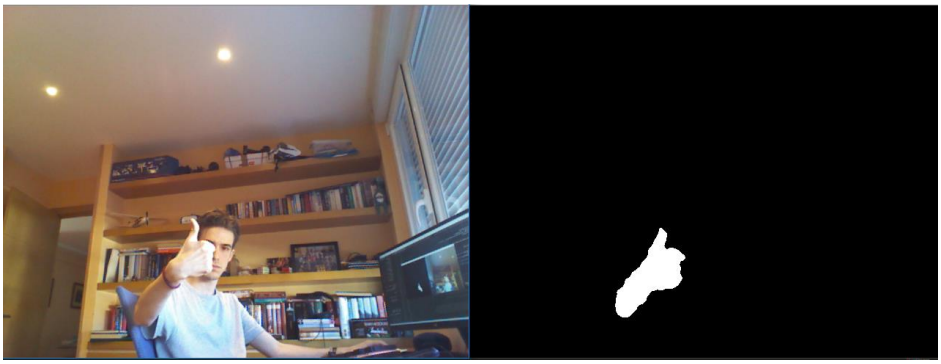
Figura 41 Máscara con contorno

```
def identificacion_mano (depth_img):
    menor = menor valor de depth_img distinto de 0
    mask = valores de depth_img entre menor  menor + 64 mm
    mask = median_filter(mask)
    contorno = contorno de la mascara
    depth_image = depth_image + contorno
    return depth_image
```

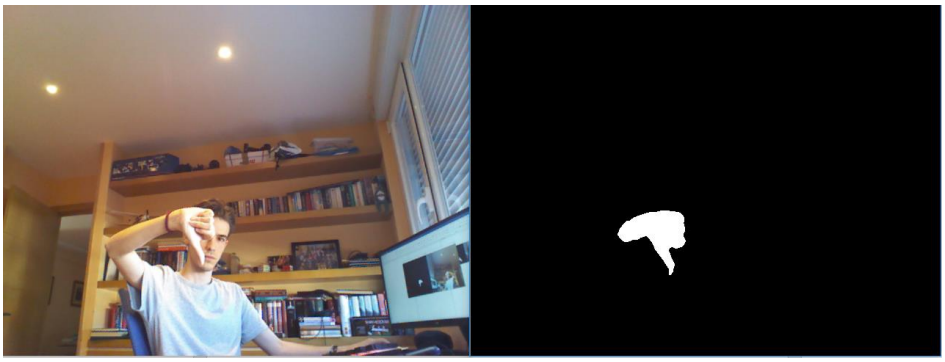
Pseudocódigo 3 Pseudocódigo de la identificación de la mano

5.3 Identificación y clasificación de gestos

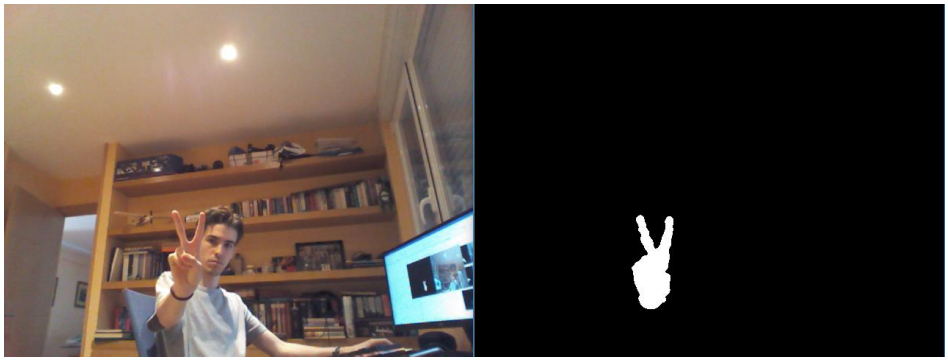
A continuación, se explica cómo se crean dos clasificadores, el primero identificará si la mano aislada está haciendo algún gesto y el segundo clasificará el gesto entre los 4 posibles en caso de que el primero indique que se está haciendo un gesto.
Los 4 gestos que se reconocen se pueden ver en la Figura 42.



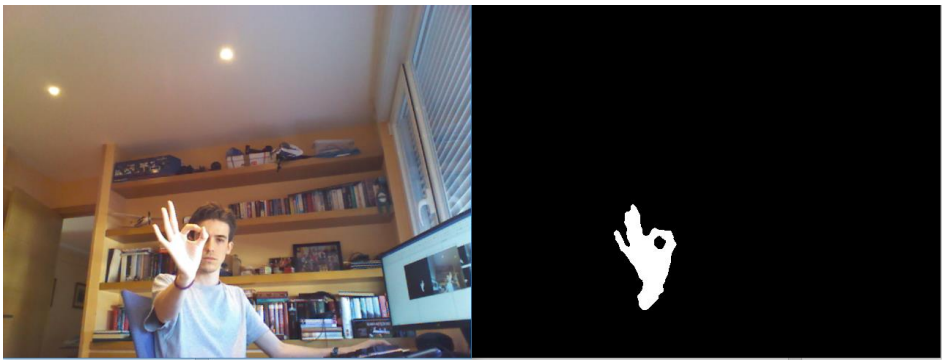
Gesto 1



Gesto 2



Gesto 3



Gesto 4

Figura 42 Gestos reconocibles y clasificables

Para no estar introduciendo al clasificador una imagen de 480x640 pixeles en la que casi todos los valores son 0 y solo algunos son 1 (los que realmente importan), se recorta en un cuadrado la zona de la mano y se re-escala a 40x40 pixeles (al clasificador siempre hay que meterle el mismo número de pixeles), la forma de conseguirlo es la siguiente:

Como contamos con el contorno, podemos utilizar los puntos máximos y mínimos de x e y para tener justo la zona de la mano (se añaden 5 pixeles por arriba, abajo, derecha e izquierda para tener algo de borde). Esta zona se re-escala a 40x40 pixeles y se pasa por el clasificador.

Se ha creado un *dataset* con el que se entrenará al clasificador guardando ejemplos de imágenes etiquetadas de 40x40 pixeles (iguales que los que entran en el clasificador). El *dataset* cuenta con 60 muestras de cada gesto (240 en total) y 260 muestras que no son gestos, estas muestras pueden ser manos que no están haciendo gestos o pueden ser formas aleatorias que podrían entrar en el clasificador. Cabe destacar que las muestras de los gestos son de ambas manos, por lo que el clasificador podrá identificar el gesto independientemente de la mano con la que se haga.

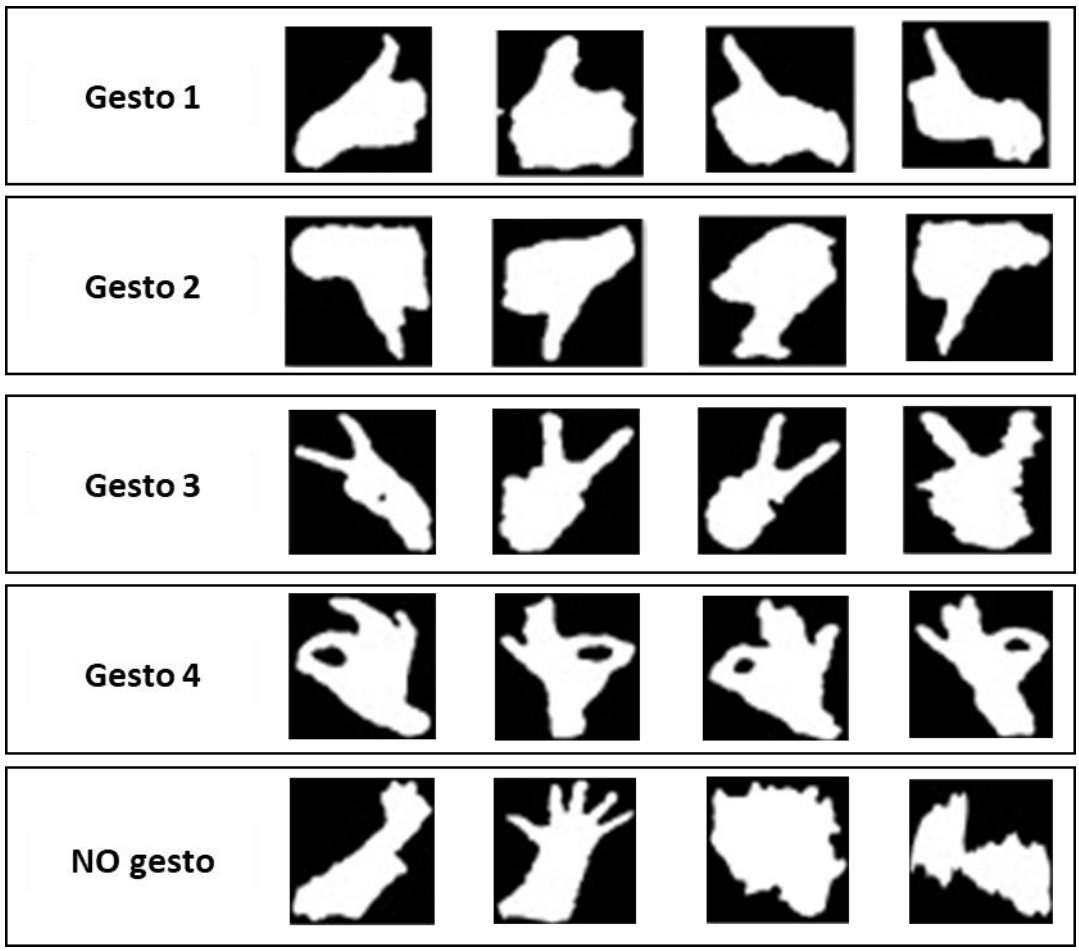


Figura 43 Muestras ejemplo con las que se entrena el clasificador

Teniendo en cuenta que el *dataset* es relativamente pequeño, no tendría sentido intentar entrenar una red neuronal convolucional, pese a que seria posiblemente el mejor

clasificador si se contara con un *dataset* suficientemente grande. Por lo tanto, las diferentes alternativas que se han barajado como posibles clasificadores son SVM, *Ranfom Forrest* y *K-Nearest Neightbors*.

La identificación y clasificación de gestos se utiliza en este proyecto para gestionar el envío de información al coche, es decir apagar y encender que se mande ordenes al coche. Mientras está apagado, el usuario que estuviera controlando el coche, podría moverse libremente sin preocuparse de que el coche se mueva provocado por sus movimientos. Esto se consigue teniendo inicialmente apagado el envío de información, en caso de que se detecte que se está haciendo el gesto 1 (gesto de OK) y se siga detectando en todos los *frames* durante un segundo entero, se encenderá el envío de información en 0.3 segundos. Mientras el envío de información está encendido, si se detecta que el gesto 2 (gesto de no OK) se esta haciendo durante 0.3 segundos (tiempo menor como medida de seguridad) el envío de información se apaga.

5.3.1 Identificación de gestos

Este primer clasificador indicará si se esta haciendo un gesto o no y se entrenará con muestras de manos haciendo gestos (240 muestras) frente a muestras de NO gestos (260 muestras).

Este es un clasificador binario y deberá devolver un verdadero en caso de que se esté haciendo un gesto y un falso en caso de que no.

Se comparan 3 clasificadores (SVM, *Ranfom Forrest* y *K-Nearest Neightbors*) de forma simple (parámetros de cada clasificador sin tunear) para decidir cuál usar antes de tocar sus diferentes parámetros.

Se utilizarán 2 métricas diferentes para evaluarlos:

1. Precisión: Número de verdaderos positivos entre la suma de verdaderos positivos y falsos positivos. Es decir, es una medida de cuánto se equivoca al decir que un gesto se está haciendo.

2. *Recall*: Número de verdaderos positivos entre la suma de verdaderos positivos y falsos negativos. Es decir, es una medida de cuanto porcentaje de las veces que se ha hecho un gesto es capaz de reconocer.

Para comparar los 3 clasificadores (y ajustar el que se elija) se va a utilizar *cross-validation*. Este método consiste en separar las muestras en N partes, entrenar con N-1 y predecir el grupo sobrante y finalmente hacer esto N veces para predecir todos los grupos. Se ha decidido usar un N de 10.

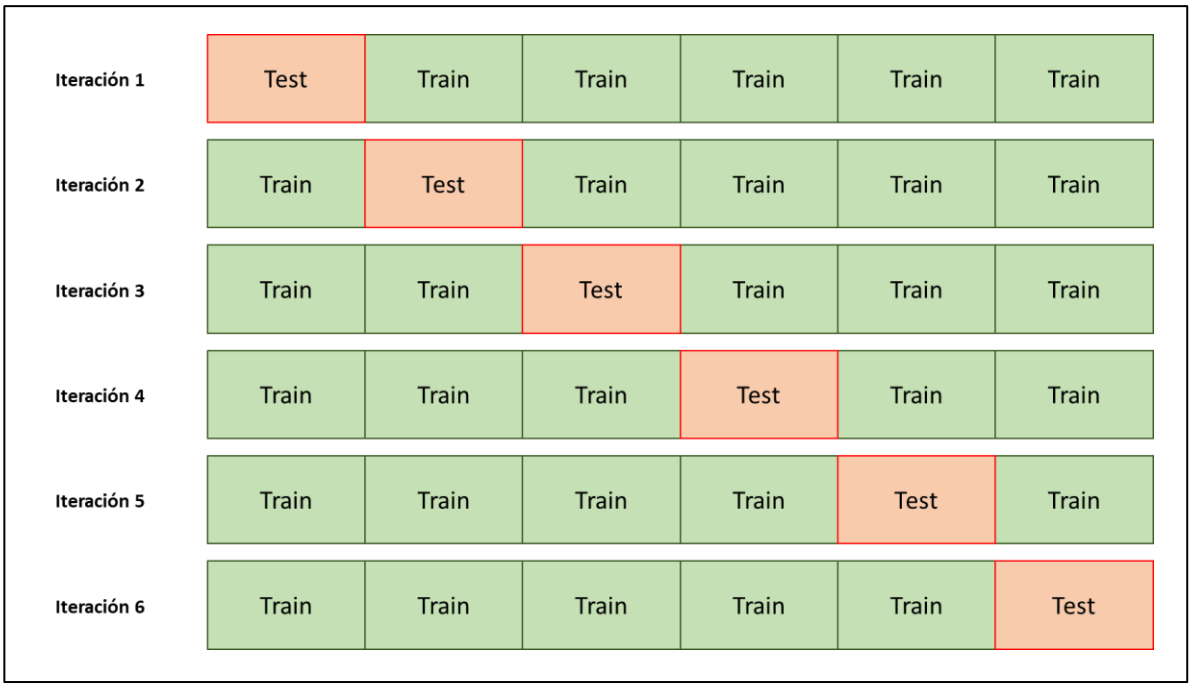


Figura 44 Ejemplo de cross-validation de N=6

Con los resultados del *cross-validation* se saca la matriz de confusión y a partir de ahí se sacan los valores de precisión y *recall*.

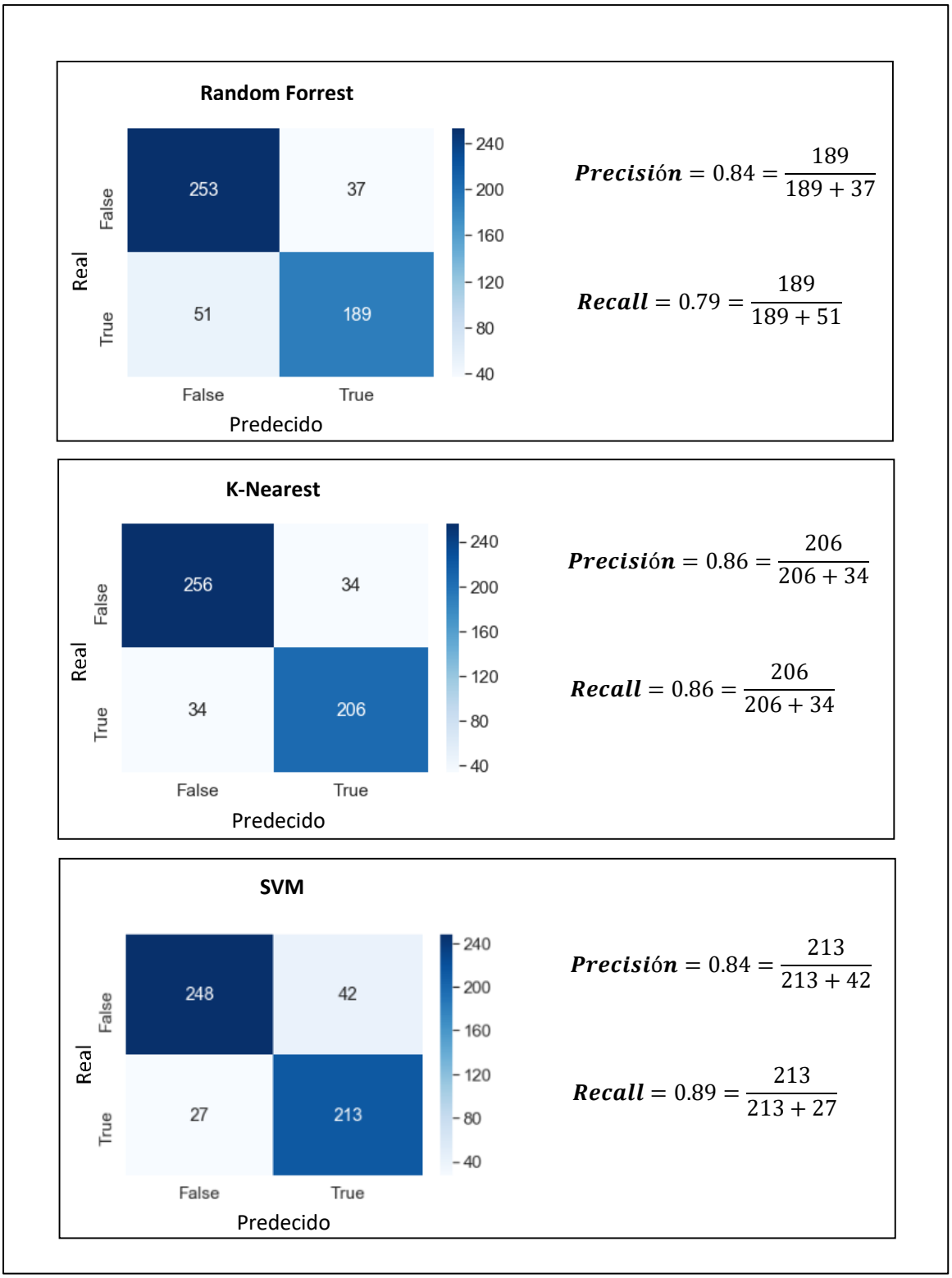


Figura 45 Precisión y recall de 3 clasificadores evaluados para el clasificador 1

Como se puede ver en la Figura 45 SVM parece más prometedor (se podrá sacrificar *recall* para aumentar precisión) que *Random Forrest* y *K-nearest Neighbors* y por eso SVM se usará para el clasificador 1.

A la hora de tunear los parámetros del SVM, se van a probar todas las combinaciones de estos valores de estos parámetros con el objetivo de maximizar precisión:

$$C = [100{,}10\ ,1\ ,0.1]$$

$$\gamma = [0.1\ ,1\ ,10, 'auto']$$

$$\text{kernel} = ['\text{poly}'\ ,'rbf'\ ,'linear']$$

$$degree = [1\ ,2\ ,3\ ,4]$$

La mejor combinación de estos parámetros es:

$$C = 10$$

$$\gamma = 0.1$$

$$\text{kernel} = '\text{ poly}'$$

$$degree = 2$$

Los resultados son los siguientes:

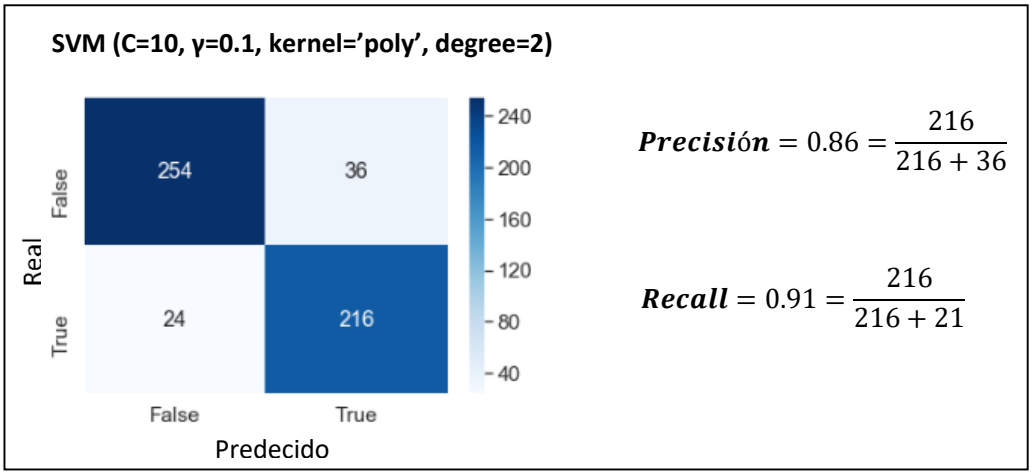


Figura 46 Precisión y recall de SVM con los parámetros tuneados para el clasificador 1

Podemos comprobar que ha aumentado tanto la precisión como el *recall* de forma considerable.

En este clasificador es mejor priorizar precisión a *recall*, ya que es más importante acertar cuando algo se califica de gesto que detectar un altísimo porcentaje de gesto.
Un SVM puede devolver la probabilidad de que una muestra sea un gesto y esto se usa aquí ara aumentar la precisión a costa de bajar *recall*.

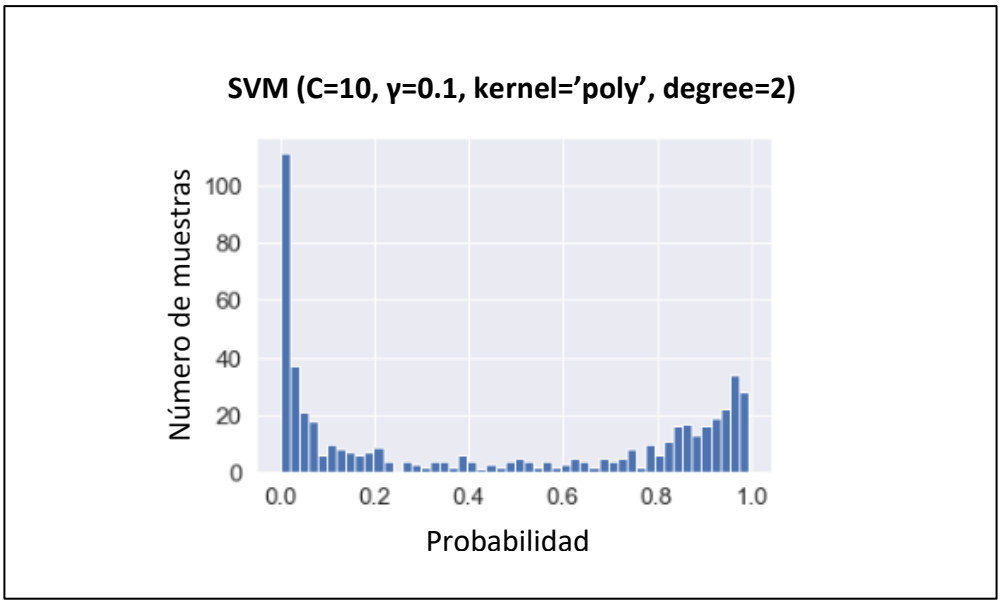


Figura 47 Número de muestras/Probabilidad SVM tuneado

Se ve que, en un gran porcentaje de las muestras, el clasificador tiene muy claro que no son gestos, y hay otro alto porcentaje en el que el clasificador tiene claro que sí son gestos. Aun así, hay un cierto número de muestras que el clasificador no tiene del todo claro si son gestos o no lo son, son esas muestras las que son clave para modificar *recall* y precisión a costa del otro.

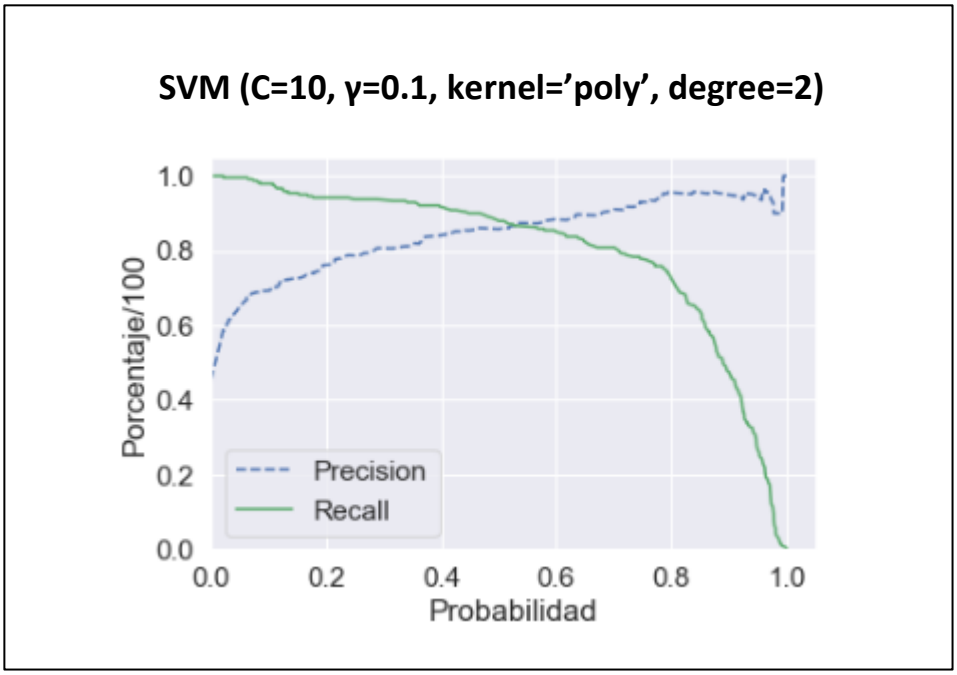


Figura 48 Precisión/Recall frente probabilidad SVM tuneado

En la Figura 48 se ve cómo van cambiando precisión y *recall* al cambiar la probabilidad a la que se acepta que el clasificador determine qué es un gesto. Si aceptas todo como gesto (probabilidad 0.0) la precisión es 0.48 (240 muestras de gestos y 260 de muestras de NO gestos) y el *recall* es 1, ya que reconoce todos los gestos. Según va aumentando la

probabilidad, el *recall* irá bajando (hasta llegar a 0) y la precisión subiendo (hasta llegar a 1), parece que un compromiso aceptable es una probabilidad de 0.7, aumenta la precisión, pero no se desploma el *recall*.

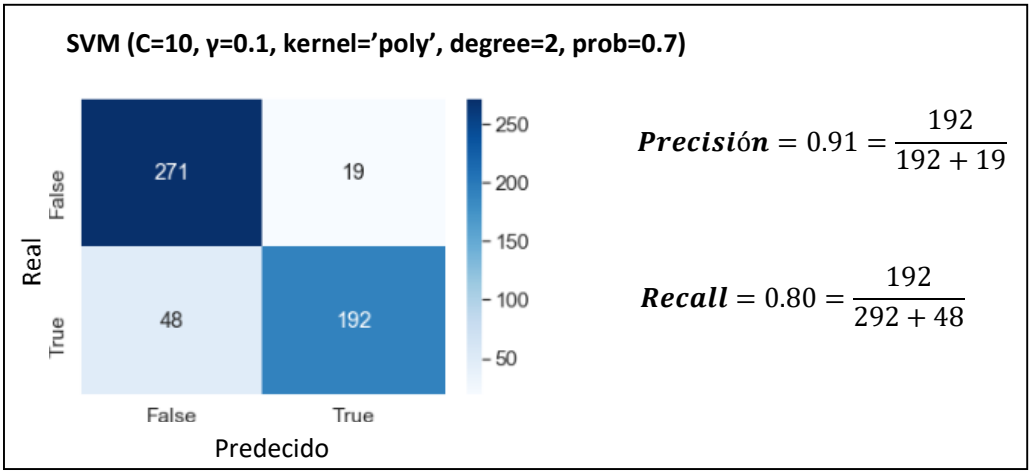


Figura 49 Precisión y recall de SVM con los parámetros tuneados para el clasificador 1 y probabilidad de 0.7

Como se puede observar, la precisión aumenta a costa del *recall*, pero la precisión es muy alta, justo lo que se quería. Los parámetros de la Figura 49 son los parámetros finales del clasficador1.

5.3.2 Clasificación de gestos

Este segundo clasificador tendrá que determinar qué gesto se esta haciendo (de los 4) y lo hará únicamente en el caso de que el clasificador 1 determine que se esta haciendo un gesto. Se entrenará con las 60 muestras que hay de cada gesto (240 en total).

A diferencia del primer clasificador este no es binario, ya que tiene 4 posibles salidas.

Igual que en el primer clasificador, se comparan 3 clasificadores (SVM, *Ranfom Forrest* y *K-Nearest Neightbors*) de forma simple (parámetros de cada clasificador sin tunear) para decidir cuál usar antes de tocar sus diferentes parámetros usando *cross-validation*. La métrica que se usará para evaluarlos será precisión (*recall* no tiene sentido para este clasificador).

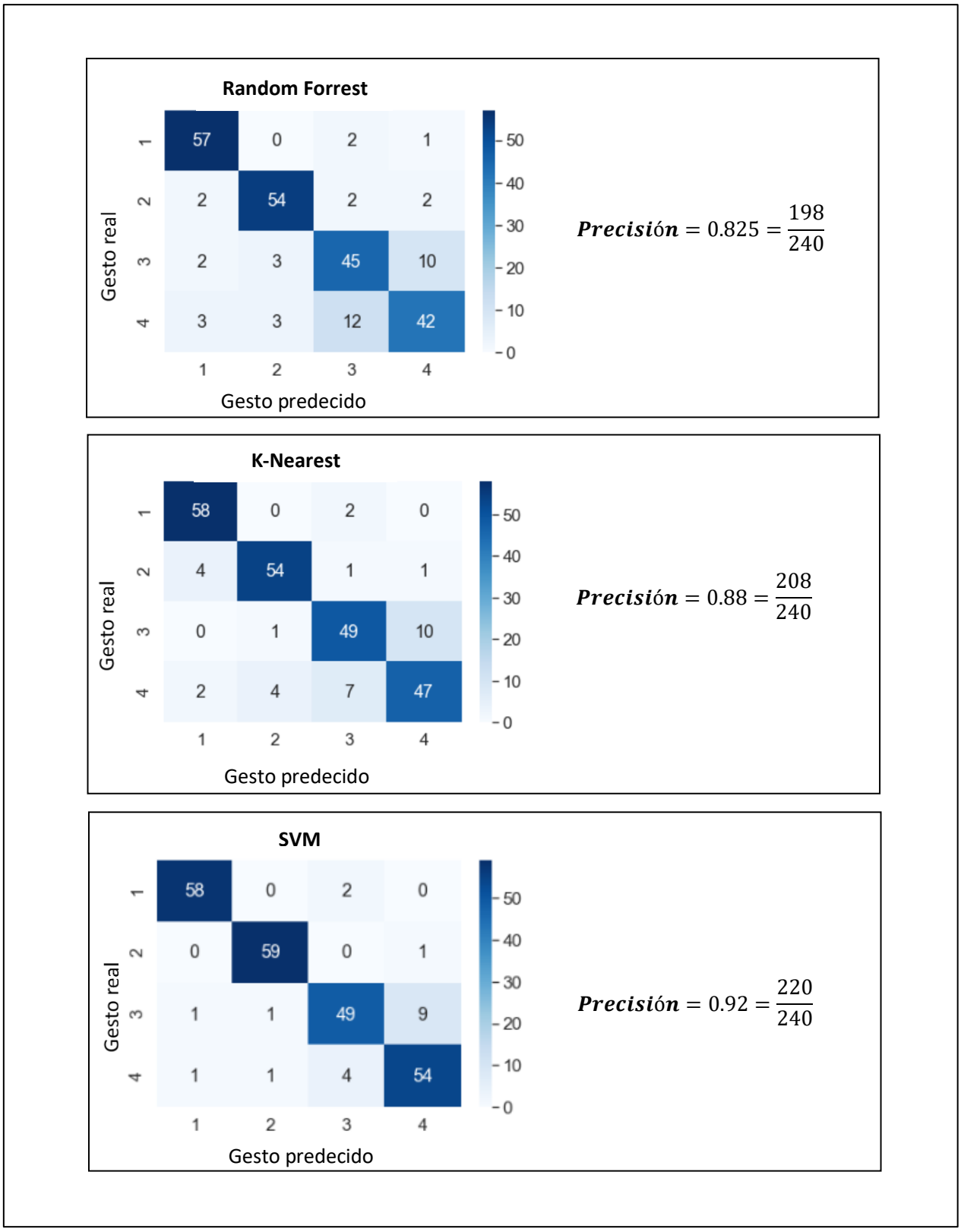


Figura 50 Precisión de 3 clasificadores evaluados para el clasificador 2

De nuevo, SVM es el que parece más prometedor (Random Forrest aumentaría mucho su precisión si se hubiera puesto un número de árboles mucho mayor).

A la hora de tunear los parámetros del SVM, se van a probar todas las combinaciones de estos valores de estos parámetros con el objetivo de maximizar precisión:

$$C = [100\,, 10\,, 1\,, 0.1]$$

$$\gamma = [0.1\,, 1\,, 10, 'auto']$$

$$\text{kernel} = ['\text{poly}'\,, 'rbf'\,, 'linear']$$

$$degree = [1\,, 2\,, 3\,, 4]$$

La mejor combinación de estos parámetros es:

$$C = 100$$

$$\gamma = 'auto'$$

$$\text{kernel} = '\text{ rbf}'$$

Los resultados son los siguientes:

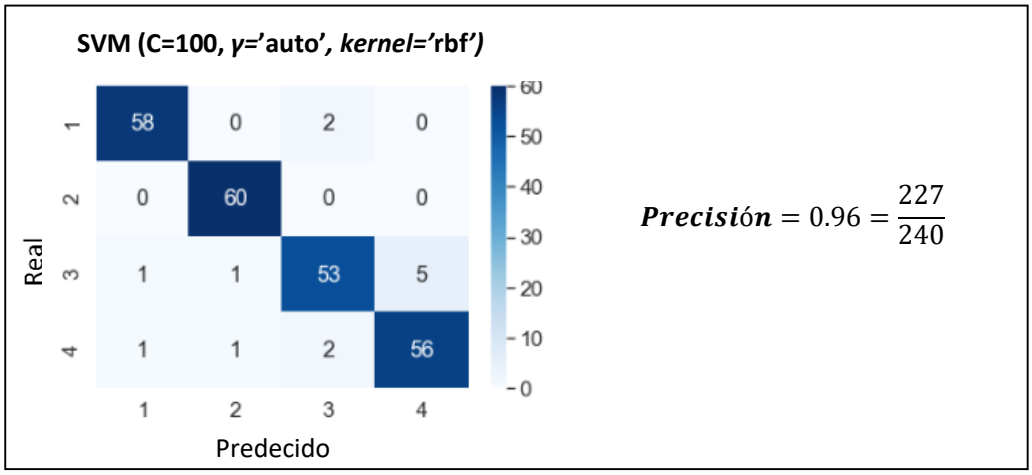


Figura 51 Precisión de SVM con los parámetros tuneados para el clasificador 2

Como se ve en la Figura 51 Precisión de SVM con los parámetros tuneados para el clasificador 2 la precisión ha aumentado.

Cabe destacar que la precisión es distinta para cada gesto y es más alta para los gestos 1 y 2 que para los gestos 3 y 4, esto se debe a lo similares que son los gestos 3 y 4 entre ellos.

En este clasificador no tiene sentido utilizar la probabilidad ya que cada muestra se etiqueta en función de su probabilidad de estar en cada grupo, por lo tanto, los parámetros finales del clasificador son los que se ven en la Figura 51.

A continuacion se puede ver el pseudocódigo de la implementación de los clasificadores en el proyecto.

```
def classifiers (mask, contorno, enviar_info):

    left , right, top, botton = coord. de pixeles máximo y mínimo de Y y X de mask
    recorte = mask recortada por left, right, top, botton + 5 pixeles de margen
    recorte = recorte re-escalado a 40x40 pixeles

    if ( predicción clasificador1 == True ) and (predicción clasificador2 == 1)  //
            and (enviar_info== False):

        if cuenta_atras1 == True:

            if time - starttime1 > 1:

                cuenta_atras1 = False
                cuenta_atras2 = True
                starttime2 = time actual
            else:

                cuenta_atras1 = True
                starttime1 = time actual

    elif ( predicción clasificador1 == True ) and (predicción clasificador2 == 2)  //
            and (enviar_info == True):

        if cuenta_atras1 == True:

            if time - starttime1 > 0.3:

                enviar_info = False
                cuenta_atras1 = False
        else:

            cuenta_atras1 = True
            starttime1 = time actual

    else:

        cuenta_atras1 = False

    if cuenta_atras2 == True:

        if time - starttime2 > 0.3:

            ref = depth del valor del centro del contorno
            cuenta_atras2 = False
            enviar_info = True
```

Pseudocódigo 4 Pseudocódigo de identificación y clasificación de gestos

5.4 Control del coche

Ya se ha explicado como se activa y desactiva el envío de información, en esta sección se explica como se calcula el giro y la velocidad que se manda al coche cuando el envío de información está activado.

5.4.1 Dirección del coche

La dirección del coche se calcula en función de la dirección de la mano, si la mano está apuntando hacia arriba, el coche ira recto, y si se gira la mano, el coche girará en esa dirección de forma proporcional al ángulo de giro.

Para conseguirlo se utilizarán los ejes de inercia de la máscara de la mano aislada (comentada anteriormente) ya que, debido a la forma alargada de la mano, serán como se ve en la Figura 52. Si la mano tuviera una forma más redondeada, los ejes no serian tan claros y por tanto no se podría calcular la inclinación de la mano. El control de la dirección del coche solo se puede llevar a cabo con la mano extendida, nunca con la mano cerrada ya que en ese caso los ejes no estarían claros.

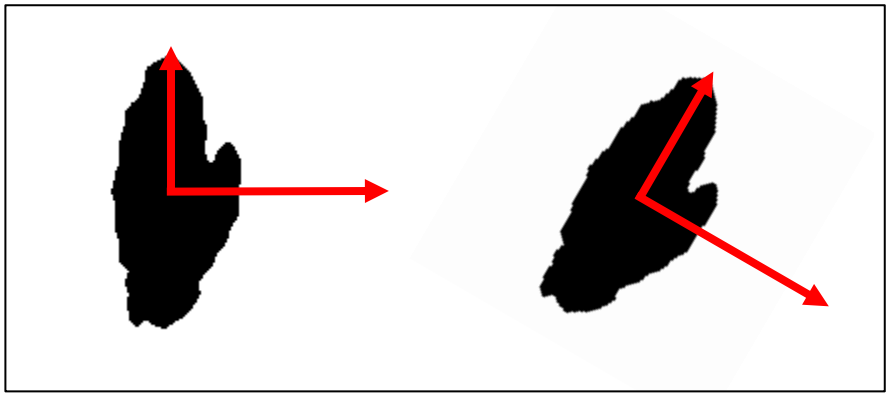


Figura 52 Ejes de inercia de la mano

Para calcular el ángulo de la mano, primero se calcularán los momentos de la máscara de la mano, estos momentos son como los momentos de un cuerpo físico, solo que se calculan en función de la intensidad de los pixeles en vez de la masa de cada partícula de un cuerpo físico. Los momentos permiten calcular el centro de masas.

$$M_{ij} = \sum_{y}\sum_{x} x^i y^j I(x,y) \qquad I(x,y) = Intensidad\ de\ cada\ pixel$$

$$\bar{x} = \frac{M_{10}}{M_{00}}$$

$$\bar{y} = \frac{M_{01}}{M_{00}}$$

Ecuación 4 Calculo de momentos de inercia y centro de masas

Una vez que se tiene el centro de masas se calcularán los momentos respecto el centro de masas.

$$\mu_{ij} = \sum_y \sum_x (x - \bar{x})^i (y - \bar{y})^j I(x,y) \qquad I(x,y) = Intensidad\ de\ cada\ pixel$$

Ecuación 5 Cálculo de momentos de inercia respecto del centro de masas

Con los momentos de inercia respecto del centro de masas, se calcula el ángulo θ de inclinación de los ejes de inercia respecto del eje x.

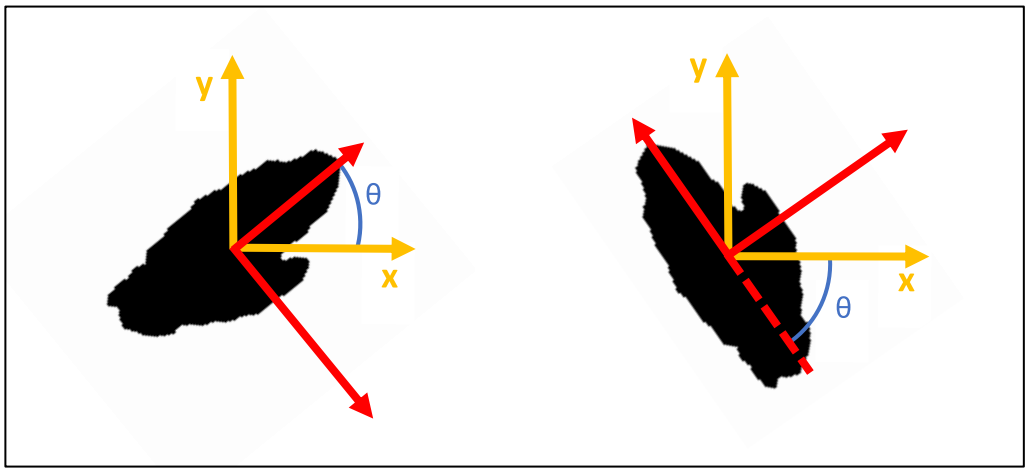


Figura 53 Ángulo θ de inclinación de los ejes de inercia

$$\theta = \frac{1}{2}\arctan\frac{-2\mu_{11}}{\mu_{20} - \mu_{02}}$$

Ecuación 6 Cálculo del ángulo θ de inclinación de los ejes de inercia

El ángulo θ está siempre entre -90 y 90, ya que, si la mano cruza la perpendicular, el ángulo θ que devuelve la ecuación pasa a ser el complementario del que era antes (con un signo menos ya que está por debajo del eje x) como se ve en la Figura 53.

Para saber cuánto es el ángulo de inclinación de la mano (ángulo 0 si la mano está extendida hacia arriba), se hará una interpolación, si el ángulo es mayor que 0 (caso de la izquierda de la Figura 53) se interpolará de [0 a 90] a [-90 a 0] y si el ángulo es menor que 0 (caso de la derecha de la Figura 53) se interpolará de [0 a -90] a [90 a 0].

```
def calculo_direccion ( mask )
        Mu = momentos de mask respecto del centro de masas
        theta = 1/2 *  atan( -2 * Mu.m11 / ( Mu.m20 - Mu.m02 ) )

        if theta > 0 grados:
                theta = interpolar (theta, [ 0, 90 ], [ 90, 0 ])
        if theta < 0 grados:
                theta = interpolar (theta, [ 0, -90 ], [ -90, 0 ])
        return theta
```

Pseudocódigo 5 Pseudocódigo del cálculo de la dirección

5.4.2 Velocidad del coche

La velocidad del coche también se controla únicamente con una mano. Si se acerca la mano hacia la cámara, la velocidad aumentará y si se aleja, la velocidad disminuirá.

La forma de conseguir esto es la siguiente: cada vez que se activa el envió de información al coche, se mira la distancia de la mano ya que se interpretará como velocidad 0. Después en cada *frame* se comparan la distancia de la mano en el *frame* y la distancia de la mano en la referencia. La velocidad máxima -tanto marcha adelante como marcha atrás- se consigue alejando/acercando la mano 24 cm respecto de la referencia.

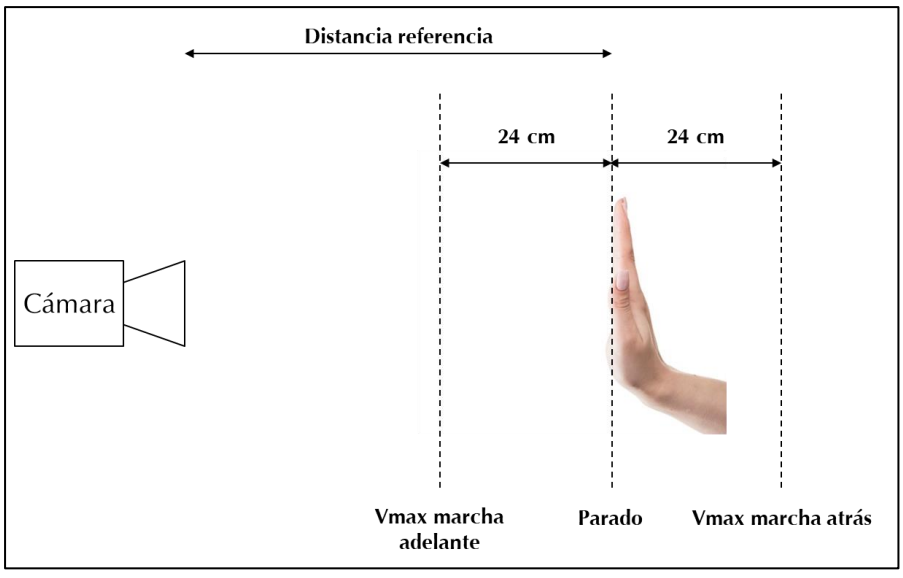


Figura 54 Velocidad del coche respecto de una distancia de referencia

def calculo_velocidad (*ref* calculada en el Pseudocódigo 4 al activar *enviar_info*, // *mask* de la mano)

M = momentos de *mask*

cX = *M*.m10/M.m00
cY = *M*.m01/M.m00

distancia = *depth* del punto medio del contorno (*cX* y *cY*)

vel = *ref* – *distancia*

if *vel* > 24 cm:

vel = 24 cm

if *vel* < -24 cm:

vel = -24 cm

return *vel*

Pseudocódigo 6 Pseudocódigo del cálculo de la velocidad

6. Comunicaciones

En esta sección se comenta como se envía la información de dirección y velocidad del coche al coche desde el ordenador.

La comunicación bluetooth a través de la UART permite enviar 8 bits cada vez que se envía información. La forma más sencilla de realizar las comunicaciones sería enviar la velocidad codificada en 4 bits y la dirección codificada en los otros 4, pero 4 bits permiten únicamente una resolución de 16 valores, la cual es bastante poca, por lo tanto, se ha implementado lo siguiente:

Los valores de velocidad y dirección se codificarán en 7 bits, lo cual aporta 128 valores (mucho más alta). El bit más significativo de los 8 que se envían por bluetooth se utilizará para informar si se trata de un dato de velocidad (bit más significativo igualado a 1) o un dato de dirección (bit más significativo igualado a 0) y se va alternando el envío de un dato de dirección y uno de velocidad.

En el microcontrolador se interpolan estos valores que llegan por la UART para determinar el ancho de los pulsos de [0 a 127] a [1 a 2] ms. Realmente no se convierte a ms sino a valores que las interrupciones del microcontrolador puedan interpretar (de 25 a 50). En el caso de la velocidad, estos valores se capan para no permitir demasiada velocidad y en vez de ir de [1 a 2] ms van de [1.3 a 1.7] ms.

```
def envio_info (vel calculada en el Pseudocódigo 6, theta calculado en el Pseudocódigo 5, //
                serial, turno)

    if turno==True:

        vel = interpolar (vel, [ -15, 15 ], [ 0, 127 ])
        serial.write (vel+128)

    else:

        theta=interpolar (theta, [ -90, 90 ], [ 0, 127 ])
        serial.write (theta)

    turno = not turno
```

Pseudocódigo 7 Pseudocódigo de las comunicaciones desde el ordenador

```
void RXinterrupt()

        Flag de interrupción de RX =0
        recibo=  registro de datos de RX

        if recibo & 0x0080 ==0
                dc = interpolar(recibo, [ 0, 127 ], [ 25, 50 ])
        else
                velocidad = interpolar(recibo, [ 0, 127 ], [ 32.5, 42.5 ])
```

Pseudocódigo 8 Pseudocódigo de las comunicaciones desde el coche

7. Resultados

A continuación, se comentan los resultados de las diferentes partes del proyecto.

Resultados del coche

En la parte del coche se ha conseguido controlar tanto la dirección del coche como su velocidad generando los mismos trenes de pulsos que genera el receptor de RC del coche, para comprobar que esto fuera así, se usó un osciloscopio para ver los trenes de pulsos que genera el microcontrolador.

Resultados del ordenador

En la parte del ordenador diferenciamos los resultados de dos partes:

- Detector e identificador de gestos: La precisión del detector y el identificador de gestos ya se mostró en la sección 5.3 Identificación y clasificación de gestos (el detector de gestos tiene una precisión del 91% con un *recall* del 80% y el clasificador tiene una precisión del 96%). Pese a saber esto, se ha querido comprobar el funcionamiento cuando lo usan personas distintas al autor del proyecto y los resultados son muy buenos. Se ha comprobado que detecta muy bien los gestos de distintas personas pese a que las muestras de gestos con los que fue entrenado fueran todos del autor del proyecto.

- Cálculo de la distancia y el ángulo de la mano (velocidad y dirección respectivamente): Para ver el funcionamiento de esta parte se ha puesto a diferentes personas a hacer diferentes ángulos con la mano mientras se acercaba y alejaba. Se analizaron las variables de ángulo y distancia que el ordenador calculaba y se concluyó que la precisión del ángulo y la distancia era perfecta.

Realmente, tanto el detector e identificador como el cálculo de la distancia y el ángulo de la mano dependen totalmente de que la mano se aislé correctamente del resto del entorno, por lo tanto, a partir de los resultados anteriores se puede concluir que la mano se aísla con una precisión muy alta.

Resultados de las comunicaciones

Para ver como de rápida y efectiva es la comunicación entre coche y microcontrolador, se ha dejado usar a diferentes personas un prototipo de este proyecto. Todos los que han utilizado el sistema informan que el coche responde de forma instantánea y precisa a sus gestos.

8.Conclusiones

Ahora se comentan las conclusiones de las diferentes partes del proyecto.
Respecto a la parte del coche, se puede concluir que se han conseguido los objetivos iniciales, ya que se ha podido controlar un coche RC (tanto velocidad como dirección) sustituyendo el receptor de RC por un microcontrolador que se comunica con un ordenador a través de bluetooth.

En cuanto al procesamiento en el ordenador se ha conseguido superar los objetivos iniciales, ya que no solo se ha podido controlar tanto la velocidad como la dirección usando la distancia a la cámara de la mano y el giro de la mano con información obtenida de una cámara Kinect, sino que además se han entrenado 2 clasificadores para identificar y clasificar 4 gestos distintos, que se usan para activar y desactivar el envío de información para controlar el coche. Estos gestos podrían usarse para añadir diferentes funcionalidades al proyecto.

9. Referencias

[1] Suzuki, S. y Abe, K., “Topological Structural Analysis of Digitized Binary Images by Border Following”. CVGIP 30 1, pp 32-46 (1985)

[2] Europapress,” https://www.europapress.es/ciencia/laboratorio/noticia-nasa-trabaja-robots-control-remoto-20130729111452.html”, junio 2019

[3] Lola M. Morente, “http://www.expansion.com/tecnologia/2017/04/15/58f24ada22601d67308b460b.html”

[4] Haritha Thilakarathne, “https://naadispeaks.wordpress.com/2018/08/12/deep-learning-vs-traditional-computer-vision/”, junio 2019

[5] Diego Calvo, “http://www.diegocalvo.es/red-neuronal-convolucional/”, junio 2019

[6] Aurélien Géron, ”Hands-On machine learning with Scikit Learn and Tensorflow”, marzo 2017

[7]Skarredhost, “https://skarredghost.com/2016/12/02/the-difference-between-kinect-v2-and-v1/”, junio 2019

[8] Orbbec, “https://orbbec3d.com/product-astra-pro/”, junio 2019

[9] Intel, “https://www.intel.es/content/www/es/es/architecture-and-technology/realsense-overview.html”, mayo 2019