Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza
1542

Trabajo Fin de Grado

Guiado de drones basado en reconocimiento de gestos con técnicas de Deep Learning

Drone guidance based on gesture recognition with Deep Learning

Óscar León Barbed Pérez

Directora: Ana Cristina Murillo Arnal

Grado en Ingeniería Informática
Computación

Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

19 de Noviembre de 2018

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Óscar León Barbed Pérez,

con nº de DNI 73161122P en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Guiado de drones basado en reconocimiento de gestos con técnicas de Deep Learning,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 22 de noviembre de 2018

Fdo:

Resumen

Un UAV (*Unmanned Aerial Vehicle*) o dron es un tipo de robot aéreo inicialmente creado con fines militares, pero que en la actualidad está siendo introducido como apoyo en muchas tareas de reconocimiento y rescate. Sus características lo hacen capaz de maniobrar en situaciones donde a un humano le resultarían complicadas. Sin embargo, su pilotaje tiene la desventaja de que el piloto habitualmente es incapaz de realizar otras tareas mientras controla el robot, pues el control remoto requiere el uso de ambas manos constantemente.

En este trabajo se propone una solución a este problema a través de un reconocedor visual de gestos: un sistema capaz de procesar las imágenes captadas por una cámara y determinar la dirección en la que el piloto desea que avance el robot. Para ello, se hace uso de técnicas de *deep learning*, en concreto *Convolutional Neural Networks* o CNNs, por su demostrada eficacia en los últimos años en el campo del reconocimiento visual automático.

El reconocimiento de gestos tiene dos fases principales:

- Detección de personas. Fase en la que el sistema detecta si hay personas en la imagen y dónde se encuentran. Para esta fase se proponen dos alternativas: mediante segmentación semántica o mediante estimación de la postura. Ambos problemas han sido abordados haciendo uso de técnicas recientes basadas en *deep learning*.

- Clasificación del gesto. Una vez detectada la persona, esta fase coge esa información y determina qué gesto está realizando esa persona. En esta fase se proponen muchas alternativas para cada una de las opciones de la fase anterior, basadas en distintas técnicas de clasificación. En concreto se han estudiado *Support Vector Machines* o SVMs y CNNs.

Este trabajo realiza una evaluación exhaustiva de las distintas alternativas para decidir cuál es la mejor arquitectura para el sistema final. Para esta evaluación se ha recopilado un nuevo conjunto de datos de evaluación más completo que los existentes en este ámbito.

Además se ha implementado un prototipo funcional integrando las mejores alternativas de este estudio. El prototipo desarrollado funciona en tiempo real con una cámara RGB. El reconocedor está conectado con un módulo de ROS (*Robot Operating System*) que se encarga de gestionar la comunicación bien con un simulador de drones como con un dron real.

Este sistema es evaluado en pruebas de integración, y comparado con un prototipo previo existente en el grupo de investigación que fue creado con objetivos similares, aunque con un enfoque más heurístico. Los resultados obtenidos muestran cómo el nuevo sistema obtiene resultados más robustos que el sistema anterior. Por lo tanto, el sistema desarrollado en este trabajo cumple las expectativas de no sólo reconocer los gestos con más precisión media que el prototipo previo, sino que lo hace de una manera más flexible a cambios y cómoda en su uso.

Todo el código relacionado con el prototipo desarrollado está disponible en el repositorio del grupo de investigación: `https://github.com/anacmurillo/unizar_interactive_robotics`

Índice general

Capítulo 1

Introducción

1.1. Motivación

Desde sus inicios, el ser humano se ha diferenciado del resto de seres vivos por el empleo, y en muchos casos dependencia, de la tecnología y herramientas que ha sido capaz de desarrollar. Estas herramientas tienen el cometido de reducir el esfuerzo y/o el riesgo de tareas necesarias para el avance de la sociedad.

En particular, en los últimos años, los campos de las tecnologías relacionadas con la inteligencia artificial y la robótica están teniendo gran impacto en la sociedad, pero a menudo, la interacción y el uso de estos sistemas todavía resulta complicada o poco intuitiva. El foco de este trabajo recae en desarrollar técnicas de aprendizaje/reconocimiento automático para mejorar la interacción entre un usuario y un robot, en concreto un UAV (*Unmanned Aerial Vehicle*) o dron.

En la actualidad, los drones han traspasado las barreras del uso exclusivo militar que tuvieron las primeras décadas tras su aparición. No es difícil encontrarlos siendo usados con fines lúdicos en parques urbanos, pero aún más vital es su uso como apoyo a operarios en la industria o a servicios de rescate. Su maniobrabilidad y reducido tamaño permiten analizar, por ejemplo, un campo de visión mucho más amplio que el de una persona, lo que es un activo muy valioso en las situaciones asociadas a este tipo de actividades.

Figura 1.1: A la izquierda, militar pilota su dron usando un mando. A la derecha, la transición a realizar en este trabajo, con el dron reconociendo los comandos por medio de una cámara.

No obstante, la manera actual de trabajar con un dron tiene algunas desventajas. Los drones actuales tienen alta maniobrabilidad, pero a costa de que el piloto emplee ambas manos para manejar el control remoto que comunique los movimientos a realizar. Como se puede observar en la figura 1.1, con ambas manos ocupadas en el mando, el piloto no sería capaz de manipular otras herramientas o, por ejemplo, avanzar por un terreno complicado. Esa desventaja es la que se pretende mitigar con el trabajo aquí realizado: desarrollar un método de captura de comandos para el dron evitando el uso de un mando a través de un sistema de reconocimiento visual automático.

Trabajos existentes. El reconocimiento automático de comandos para interacción con un sistema "manos libres"no es un problema recién descubierto, y ha sido explorado en otros campos como, por ejemplo, en el ámbito de los videojuegos con el sistema *Kinect* [1]. Encontramos distintos tipos de estrategias, según el tipo de sensores que se utilicen para la tarea: mediante comandos por voz [2], mediante sensores de captura de movimiento [3], mediante reconocimiento visual de gestos [4, 5], etc. Estas técnicas de reconocimiento automático han avanzado notablemente en los últimos años, con los métodos basados en *deep learning* marcando el estado del arte en muchos de estos problemas, como se puede observar en las citas antes mencionadas. Los estudios existentes en el campo se describen en el capítulo 2 en más detalle.

Es por ello que en este trabajo se ha hecho hincapié en el uso de técnicas de *deep learning* como herramienta principal para resolver el problema planteado en este trabajo: desarrollar un reconocedor visual de gestos que permita enviar comandos de navegación a un dron.

Este propósito no es nuevo en este trabajo, es decir, existe ya un interés entre empresas tecnológicas por proyectos así, como se puede observar, por ejemplo, en una patente solicitada por la empresa Amazon [6]. Esta patente recoge la idea de un dron capaz de reconocer gestos o comandos por voz. Así, se demuestra que no sólo existe un interés científico en proyectos como este, sino que también existe un mercado y una manifestación clara del interés de la sociedad en avances de este tipo.

1.2. Contexto de realización del trabajo

El contexto en el que se ha realizado el trabajo es el grupo *Robotics, Perception and Real Time* (RoPeRT), un grupo de investigación dentro del Instituto de Ingeniería e Investigación de Aragón (I3A). En particular, el trabajo ha sido desarrollado bajo la supervisión de la directora del mismo, y con el apoyo y experiencia de otros integrantes del grupo.

Desde un punto de vista de **motivación personal**, esto me permite observar el ambiente de trabajo dentro de un grupo de investigación dentro de mi facultad, lo que me ayudará a decidir el rumbo que quiero tomar una vez finalizados mis estudios actuales. Además, quiero un trabajo relacionado con el *deep learning* porque aunque en el Grado recibí una breve introducción a este campo, tengo interés por conocer las metodologías usadas en él para afrontar problemas y formular soluciones.

Para las pruebas de navegación de un dron, se ha hecho uso de un sistema de simulación existente dentro de la plataforma *Robot Operating System* (ROS)[1], usado ampliamente por el grupo de investigación. Algunos de los datos empleados han sido cedidos por un proyecto anterior en colaboración con la universidad ETH Zurich. De este proyecto anterior, está disponible, como punto de partida para este trabajo, tanto dichos datos como un prototipo de base (detallado en la subsección 3.3.1) construido con heurísticas basadas en características más tradicionales de procesamiento de imagen. Como se describe a continuación, el objetivo en este trabajo es conseguir un prototipo mejor con técnicas de *deep learning*.

1.3. Objetivos y tareas

El **objetivo principal** de este proyecto es crear un prototipo capaz de dar comandos de navegación a un dron, mediante un interfaz de reconocimiento automático visual de gestos.

En particular, este trabajo se centra en cómo resolver dos de los problemas principales para desarrollar dicho interfaz:

- **Detección de personas.** Es el proceso por el cual se afirma la aparición o no de alguna persona en una imagen, y adicionalmente en este caso se requiere conocer su postura, es decir, la posición de las distintas articulaciones visibles de la persona.
- **Clasificación del gesto.** Una vez detectada una persona, este problema trata de identificar cuál es el comando que ésta quiere dar al dron. En particular, en este trabajo, nos centramos en comandos de dirección (en qué dirección se quiere avanzar) para dar al dron comandos de movimiento.

Para conseguir los objetivos propuestos, el trabajo a desarrollar se ha organizado en una serie de tareas que sirven como pasos lógicos en el desarrollo del prototipo:

- **Tarea 1.** Estudio de los conceptos (modelos y componentes) básicos de las redes neuronales convolucionales (CNN de su nombre en inglés, *Convolutional Neural Network*). Se pretende poder entender el funcionamiento de las mismas y su uso hoy en día.
- **Tarea 2.** Estudio e instalación de software necesario para trabajar con técnicas de *Deep Learning* (*Caffe2*[2] y *Tensorflow*[3]). Familiarización con el software actualmente relacionado con el desarrollo y empleo de CNNs en sus distintas aplicaciones. Sobre todo el uso de los entornos más utilizados para poder trabajar con ellas. Esta tarea tiene por objetivo aprender a emplear CNNs codificadas en estos entornos que sean útiles para el proyecto, y crear y usar CNNs pequeñas propias.
- **Tarea 3.** Estudio de métodos existentes y puesta en marcha de los más adecuados. Por un lado, estudiar métodos de detección de personas, con el

[1] `http://www.ros.org/`
[2] `https://caffe2.ai/`
[3] `https://www.tensorflow.org/`

objetivo de detectarlas en los alrededores de un dron (entre 2 y 10 metros) y conseguir detalles de la postura de la persona. Por otro lado, estudiar distintos métodos de clasificación, en particular *Support Vector Machines* (SVM) y CNNs.

- **Tarea 4.** Diseño e implementación de un detector visual de indicaciones para establecer el objetivo. Poner en marcha mecanismos de detección de personas y proponer distintas representaciones de los resultados. Diseñar y evaluar distintos clasificadores de gestos basados en dichas representaciones. Elegir el más apropiado para el sistema final.
- **Tarea 5.** Puesta en marcha del simulador de drones existente en ROS (Robot Operating System) y conectarlo con el reconocedor de gestos desarrollado.
- **Tarea 6.** Evaluar y comparar las distintas versiones con el reconocedor previo disponible, tanto en datasets existentes como en demostraciones en tiempo real.
- **Tarea 7.** Redacción de la documentación del proyecto y el manual de usuario.

Distribución temporal. La distribución temporal de estas tareas a lo largo del proyecto se puede ver de manera aproximada en la tabla 1.1.

	N	D	E	F	M	A	M	J	J	A	S	O	N
T1- Estudio Deep Learning													
T2 - Software relacionado con CNNs													
T3 - Estudio de modelos actuales													
T4 - Diseño y comparación de modelos													
T5 - Integración con simulador													
T6 - Evaluación del prototipo													
T7 - Redacción de la documentación													

Tabla 1.1: Diagrama de Gantt aproximado de la distribución de las tareas en la duración del trabajo. Cada fila representa una tarea, y cada columna representa un mes, empezando por noviembre de 2017 hasta noviembre de 2018.

1.4. Guía de la memoria

En el capítulo 2 se hace una recopilación del otros trabajos que han servido como punto de partida o fuentes de información a lo largo de la realización de este proyecto. El capítulo 3 contiene la parte del trabajo en la que se han probado y escogido distintos modelos para los dos núcleos de cálculo del prototipo (detección de personas y clasificación). En el capítulo 4 se presenta el prototipo final construido y la evaluación del mismo con datos en tiempo real. A lo largo

de todo el proceso se han ido extrayendo conclusiones que se ven recogidas en el capítulo 5.

Como material adicional, se incluyen cuatro anexos al trabajo. El anexo A contiene nociones básicas sobre redes neuronales convolucionales. En el anexo B se recogen algunos ejemplos representativos del contenido de los datasets utilizados en el trabajo. A lo largo del trabajo se han realizado experimentos, cuyos resultados completos se encuentran en el anexo C. Por último, el prototipo desarrollado en el proyecto tiene un manual de uso que se ha incluído en el anexo D.

Capítulo 2

Trabajo relacionado

En este capítulo se recoge información sobre los campos de estudio más importantes para la realización de este trabajo. Además, se incluyen referencias a publicaciones concretas que han servido de apoyo en el desarrollo del mismo.

2.1. Sistemas de clasificación y reconocimiento visual

En la disciplina del aprendizaje automático, la clasificación es una tarea fundamental, ya que usualmente aparece en muchos de los problemas a los que este campo se enfrenta. De particular relevancia para este proyecto, es la clasificación o reconocimiento automático con datos visuales.

Hay muchas soluciones tradicionales en la literatura del aprendizaje automático para clasificación y reconocimiento en imágenes. Unas de las más utilizadas por su rapidez de uso son las máquinas de soporte vectorial o *Support Vector Machines* (SVM) [7]. Las SVMs son sistemas que se basan en la representación de los datos como puntos en un espacio, tomando los parámetros de los datos (o una transformación de ellos con una función definida en la creación del SVM llamada *kernel function*) como coordenadas en el mismo. Así, una SVM es capaz de calcular hiper-planos que cumplan ciertas propiedades sobre esos puntos en el espacio como, por ejemplo, servir como frontera de separación en distintas clases con los datos previamente etiquetados, es decir, clasificarlos. Una representación de esta aplicación se muestra en la figura 2.1.

Más recientemente, el campo de la clasificación automática de imágenes ha visto grandes avances gracias a soluciones basadas en redes neuronales profundas. Estas redes, que se construyen de manera similar a las redes neuronales más tradicionales pero con muchas capas ocultas, son la base del campo de estudio conocido como *deep learning* [8]. En concreto, las aplicaciones para el reconocimiento visual hacen uso de un tipo de redes neuronales conocidas como redes neuronales convolucionales (CNN). Estas redes, como su nombre indica, tienen como operación básica la convolución, que permite aprovechar la correlación e información espacial existente entre píxeles vecinos. Un ejemplo de la arquitectura típica de una CNN para clasificación de imágenes se puede observar en la

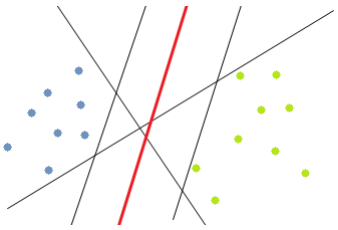


Figura 2.1: Ejemplo de hiper-planos calculados por SVM para clasificación (Fuente: Wikipedia;FilFile:SVM_Example_of_Hyperplanes.png). Un punto representa un dato. Los ejes (en este caso dos: vertical y horizontal) corresponden a las dimensiones donde se representan los parámetros de los datos. El color de cada punto lo identifica como perteneciente a una "clase". Las líneas trazadas en negro son posibles hiperplanos que clasifican los datos. La trazada en rojo es la separación óptima encontrada durante el entrenamiento.

figura 2.2. Uno de los primeros ejemplos de CNN que demostró su potencial en el reconocimiento visual fue el de *AlexNet* [9].

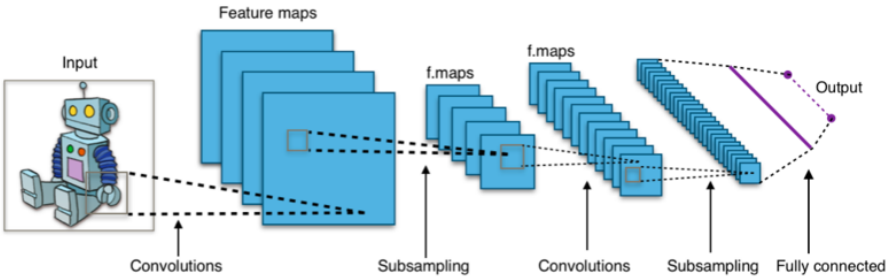


Figura 2.2: Ejemplo de arquitectura de una CNN típica para clasificación de una imagen (Fuente: Wikipedia;File:Typical_cnn.png). Las primeras capas son de convolución, y realizan cálculos sobre regiones de la imagen para extraer patrones (*features*) entre píxeles vecinos. Las últimas capas (*Fully connected*) realizan los cálculos finales de clasificación para generar la salida (*Output*)

Los estudios realizados en el campo y el avance del *hardware* específico (sobre todo las GPUs, pues su arquitectura interna es especialmente apta para la ejecución de CNNs) han permitido que las CNNs hayan mejorado en términos de eficiencia y precisión desde su aparición. Hoy en día existen arquitecturas específicamente diseñadas para una clasificación eficiente, que se puede ejecutar con pocos recursos de *hardware*, como es el ejemplo de las *MobileNets* [10], una familia de redes recientemente creadas con el propósito de obtener buenos resultados de clasificación con mínimo tiempo de cálculo.

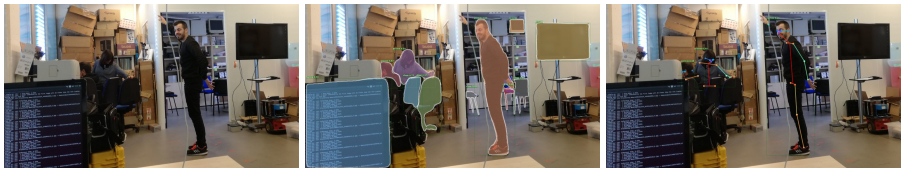
En los últimos años, se han ido desarrollando numerosas arquitecturas de CNNs específicas para distintas tareas. De particular interés en nuestro proyecto son aquellas relacionadas con la detección de personas en imágenes, que se detallan en la sección siguiente.

2.2. Detección de personas

El problema de la detección visual de personas es otro problema que el aprendizaje automático ha estudiado de manera extensa. Entre las distintas soluciones

propuestas para él, también existen técnicas con un carácter más tradicional, que se basan en el uso de características de las imágenes diseñadas de forma manual, basadas en conocimientos previos de las personas sobre el problema. Como uno de los ejemplos más conocidos de este tipo de técnicas, se encuentra el reconocedor de personas basado en histogramas de gradientes orientados (HOG) [11]. Las características mencionadas en este caso son calculadas observando las variaciones (gradientes) en intensidad en las imágenes en distintas direcciones (por eso son orientados). Estas características se calculan por toda la imagen, y un clasificador sencillo identifica qué regiones de la imagen pertenecen a una persona y cuáles no.

Como se ha comentado anteriormente, los trabajos basados en *deep learning* han conseguido avances muy importantes en los últimos años, también en cuanto a la detección de personas. La versatilidad de las CNNs permite tratar el problema de distintas maneras. Por un lado encontramos enfoques muy concretos, como la detección únicamente de personas en una imagen y de su postura en la misma (estas técnicas son también conocidas como *pose estimation*) [12]. Por otro lado, es posible tomar un enfoque más general del problema, realizando tareas de *segmentación semántica*. La segmentación semántica es una tarea en visión por computador que consiste en etiquetar cada píxel de una imagen dada con una "clase", es decir, determinar el valor semántico concreto de lo que aparece en ese píxel [13]. Esto permite que modelos diseñados para segmentación semántica de clases frecuentes, habitualmente tienen como una de sus clases la de "persona". Estas soluciones nos sirven como paso intermedio al problema de la detección de personas.



(a) (b) (c)

Figura 2.3: Ejemplos de aplicación de CNNs para detección de personas. (a) Imagen de entrada. (b) Resultado de la aplicación de un modelo de *Mask R-CNN* de segmentación semántica, donde se reconocen todos los objetos para los que ha sido entrenado, entre ellos el de "persona". (c) Resultado de la aplicación de un modelo de *Mask R-CNN* de detección de personas (estimación de la postura).

De entre las posibilidades para segmentación semántica basadas en CNNs que se han explorado, se ha optado por el uso de *Mask R-CNN* [14]. *Mask R-CNN* no es una arquitectura definida y cerrada, sino que es una estructura que propone las últimas capas de clasificación sobre cualquier sistema capaz de extraer regiones de interés (*region of interest*, RoI) [15] de una imagen. Los autores proponen utilizar como base arquitecturas de CNNs para clasificación, como *ResNet-50* o *ResNet-101* [16], para obtener las RoIs. Las capas finales de la red *Mask R-CNN* reciben como entrada las RoIs de una imagen, produciendo por cada región la siguiente salida:

- Puntuaciones de cada clase (la región es asignada a la clase de mayor puntuación).

- *Bounding box* del objeto de esa clase.
- Una máscara binaria de cada clase, con los píxeles pertenecientes a la clase correspondiente a 1 y el resto a 0.

Todos estos datos obtenidos por cada imagen hacen que *Mask R-CNN* sea capaz de realizar *instance segmentation*, que es un paso más allá de la segmentación semántica ya que permite reconocer no sólo qué píxeles pertenecen a cada clase, sino que además se conoce a cuál de las instancias de esa clase pertenecen. Por ejemplo, si hay tres personas en una imagen de entrada: reconoce en qué píxeles de la imagen aparecen personas, sabe que son tres y sabe qué píxeles pertenecen a cada persona.

Se ha seleccionado *Mask R-CNN* para este proyecto porque es una solución capaz de realizar segmentación semántica que marca el estado del arte en el momento de realización del trabajo. Además, como se puede observar en los ejemplos de la figura 2.3, además de los modelos centrados en segmentación semántica, también se dispone de modelos de esta red entrenados para añadir *pose estimation* a las personas encontradas dentro de la escena, es decir, dar información de la postura de las mismas. En este trabajo se considerarán ambas opciones.

Capítulo 3

Reconocedor de gestos desarrollado

Este capítulo detalla las alternativas construidas y evaluadas para las principales componentes del sistema de reconocimiento de gestos. El diagrama de la figura 3.1 ilustra el funcionamiento este sistema.

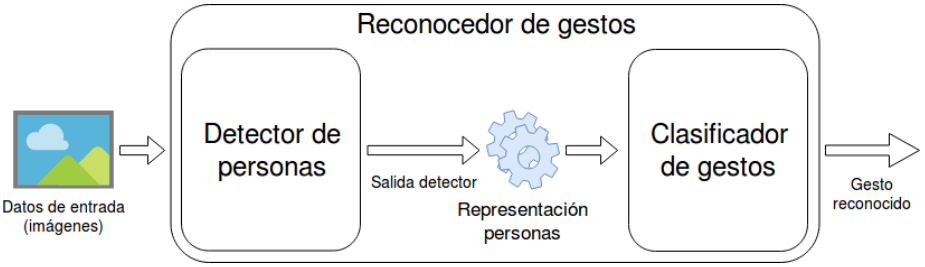


Figura 3.1: Diagrama de alto nivel sobre el funcionamiento del reconocedor de gestos del trabajo. Los datos de entrada o imágenes se pasan al reconocedor de gestos. Este reconocedor tiene en primer lugar un subsistema de *Detección de personas*. La salida de ese detector se procesa para *Representar los datos de la detección de persona* en la imagen, de tal forma que el segundo subsistema, el *Clasificador de gestos*, sea capaz de procesar esa representación y reconocer el gesto que está realizando la persona. Este gesto es la salida del reconocedor.

3.1. Detección y representación de personas

Al tratarse de un sistema de reconocimiento visual, los datos de entrada son proporcionados en forma de imagen. Esta imagen será procesada, en primer lugar, por el modelo entrenado con la arquitectura escogida para la detección. Concretamente, el sistema tiene como objetivo la detección de personas, por lo que la salida producida por el modelo debe representar las personas que aparezcan en la imagen, dependiendo de las posibilidades del modelo y de las necesidades del sistema en fases posteriores.

3.1.1. Detector

Para la primera fase de detección de personas, se ha decidido tomar la mejor opción actual disponible. La arquitectura escogida es la de *Mask R-CNN* [14], que como ya se ha mencionado anteriormente, marca el estado del arte en el momento de su elección. Se ha decidido usar su implementación oficial. Esta implementación se llama *Detectron* [17] y está programada en el *framework* para redes neuronales denominado *Caffe2*[1].

Una de las ventajas más importantes de usar la implementación oficial es la posibilidad de utilizar directamente modelos entrenados en datasets conocidos. En este caso, se ha decidido utilizar un modelo entrenado con datos del dataset *Common Objects in Context* (COCO)[2], que contiene objetos y clases comunes, entre ellos *persona*. Las arquitecturas de estos modelos entrenados tienen la garantía de que ya han sido probados y su valía ha quedado demostrada en las publicaciones de sus creadores. De no disponer de estos modelos, sería necesario realizar el entrenamiento de modelos propios, que en casos de redes profundas con datasets grandes, puede requerir varios días por modelo, además de muchísimos datos etiquetados para entrenar. En concreto, los modelos empleados han sido dos:

- Modelo de segmentación semántica[3]: Este modelo realiza la segmentación semántica de imágenes, proporcionando segmentos con los píxeles pertenecientes a cada objeto. En adelante, cuando se haga referencia al reconocedor de gestos resultante de usar este detector, se hablará de **Reconocedor con Segmentación**.

- Modelo de estimación de la postura[4]: Este es un modelo utilizado para detección de personas con estimación de la postura. Proporciona, por cada persona en la imagen, una lista con las posiciones de sus puntos clave (*keypoints*) en coordenadas de la imagen, detallados en la lista 3.1.

$$\begin{aligned} Kp = \{&\textit{nariz}, \textit{ojo izquierdo}, \textit{ojo derecho}, \textit{oreja izquierda}, \\ &\textit{oreja derecha}, \textit{hombro izquierdo}, \textit{hombro derecho}, \textit{codo izquierdo}, \\ &\textit{codo derecho}, \textit{muñeca izquierda}, \textit{muñeca derecha}, \textit{cadera izquierda}, \\ &\textit{cadera derecha}, \textit{rodilla izquierda}, \textit{rodilla derecha}, \textit{tobillo izquierdo}, \\ &\textit{tobillo derecho}\} \end{aligned} \tag{3.1}$$

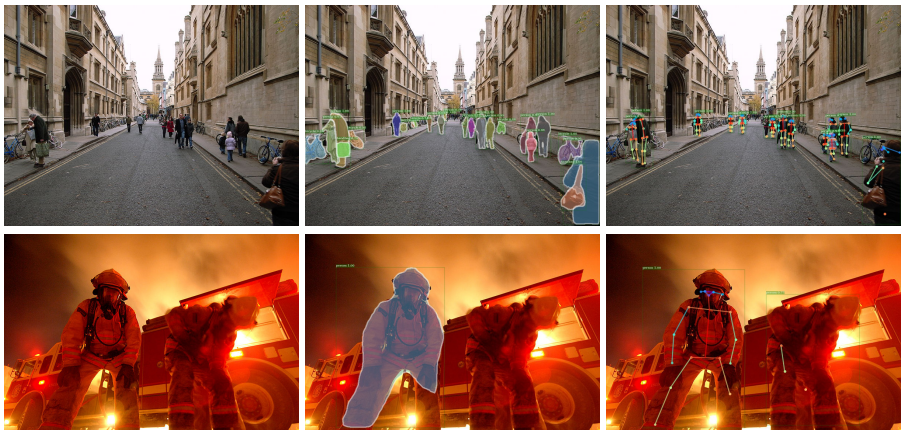
  En adelante, cuando se haga referencia al reconocedor de gestos resultante de usar este detector, se hablará de **Reconocedor con Esqueleto**.

Evaluación del funcionamiento. La evaluación cuantitativa más general de estos modelos se encuentra recogida en la publicación de sus autores [14]. En este trabajo, se ha realizado una valoración cualitativa adicional de los resultados obtenidos con esta implementación, una vez conseguida su puesta en marcha.

[1] https://caffe2.ai/
[2] `http://cocodataset.org/#home`
[3] En el conjunto 12_2017_baselines, el modelo e2e_mask_rcnn_R-101-FPN_2x [17]
[4] En el conjunto 12_2017_baselines, el modelo e2e_keypoint_rcnn_R-101-FPN_1x [17]

Para ello se ha hecho uso de *scripts* de inferencia disponibles como parte del código distribuido con el modelo [17].

En la figura 3.2 se encuentran recogidos algunos de los ejemplos de inferencias realizadas en esta valoración. En la primera fila se observa una imagen de una calle con varios transeúntes. En ella, la detección es casi perfecta: se señalan y delimitan más de una decena de personas, bicicletas y bolsos cercanos con exactitud. En ella se puede observar que con buena iluminación y si la persona aparece completamente en la imagen, el detector es certero independientemente del número de personas. En la segunda fila se muestra uno ejemplo de imagen de gran dificultad: la iluminación es inusual (cercanías de un incendio) y las personas que aparecen llevan un traje de cuerpo completo que cubre incluso la cara (bomberos en uniforme de rescate). El detector es capaz de apuntar a uno de ellos y, en el caso de estimación de la postura, marca que hay otro más. Estas pruebas y las realizadas con imágenes de los datasets propios (explicados en profundidad en la subsección 3.4.1), han servido como prueba preliminar para confirmar que los modelos son aptos para realizar las tareas de detección en este trabajo.



(a) (b) (c)

Figura 3.2: Dos ejemplos de inferencias (predicciones) realizadas con la implementación oficial de *Mask R-CNN*. (a) Imagen de entrada (Fuentes: Wikipedia;File:Turl_Street,_Oxford.jpg y Wikipedia;File:Firefighters_in_Iraq.JPG). (b) Resultado de la aplicación del modelo de segmentación semántica (**Reconocedor con Segmentación**). (c) Resultado de la aplicación del modelo de detección detallada de personas, incluyendo estimación de la postura (**Reconocedor con Esqueleto**).

Como se ha mencionado, con estos dos modelos es posible obtener dos tipos de salidas distintas de una imagen. Para ambas, se ha estudiado cómo representar a las personas que aparecen en esa imagen de cara a la clasificación de gestos posterior, que se detallan a continuación.

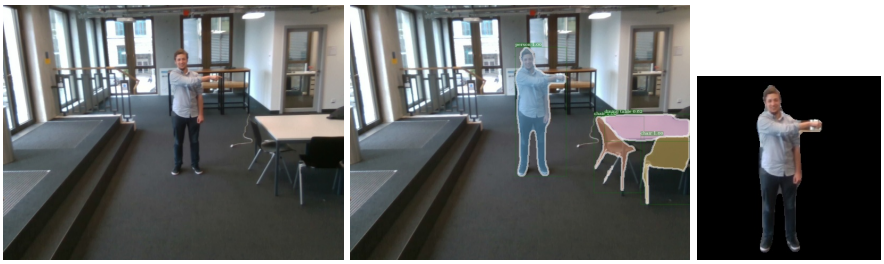
3.1.2. Representación de la persona

Los datos relativos a las personas detectadas, antes de poder ser utilizados como entrada del clasificador de gestos han de ser procesados para representar a la persona en un formato concreto. A continuación se detalla el proceso que

se propone aplicar a estos datos antes de pasar a la fase de clasificación. El tratamiento de los datos debe ser distinto dependiendo de la salida producida en la fase de detección de personas (**Reconocedor con Segmentación** o **Reconocedor con Esqueleto**).

Reconocedor con Segmentación. En esta primera opción para representar la información de personas detectadas, el modelo empleado divide la imagen en segmentos con los píxeles pertenecientes a cada objeto. Además, identifica a qué clase pertenece cada objeto, con lo que se sabe qué segmentos son de la clase "persona".

En el sistema diseñado en este trabajo, se propone seleccionar el segmento con mayor tamaño en la imagen como principal (es decir, se ha decidido que el gesto a reconocer sea el del usuario que ocupe más espacio en la imagen, que es el que es más visible). Con este segmento principal, se ha procedido a obtener un parche de la imagen con dimensiones cuadradas (porque en la siguiente fase se utiliza una CNN que requiere una entrada de tamaño fijo, y se ha decidido usar una forma cuadrada). Este parche es una región cuadrada con el lado igual a la mayor de las dimensiones de la persona en la imagen original. Además, al disponer de un segmento detallado, se ha hecho uso de él para descartar (marcar en negro) los píxeles del parche que no pertenezcan a la persona (como se puede ver en la figura 3.3). Con esto, se consigue que el fondo de la imagen no influya en la clasificación posterior (añade independencia con respecto al escenario en el que se esté utilizando el reconocedor de gestos).



(a) (b) (c)

Figura 3.3: Ejemplo de detección de personas con el modelo de detector del **Reconocedor con Segmentación**. (a) Imagen de entrada. (b) Resultado de la aplicación del modelo de segmentación semántica. (c) Representación escogida de la persona que aparece en la imagen.

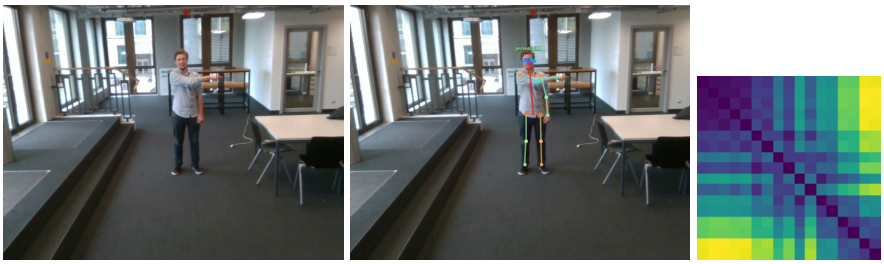
Reconocedor con Esqueleto. En esta segunda opción para representar las personas detectadas, el modelo produce listas con los 17 *keypoints* contenidos en la lista 3.1, representados con sus coordenadas vertical y horizontal en la imagen, para cada persona detectada. Una lista de pares coordenadas x, y no resulta una entrada homogénea adecuada para un sistema de clasificación basado en CNNs que necesita una entrada de al menos dos dimensiones y lo más densa posible, por lo que se han explorado otras posibilidades de representación para estos datos.

Inspirado en otros trabajos que identifican posturas de personas en 3D analizando las distancias entre las articulaciones [18] [19], se ha propuesto transformar los datos obtenidos con el modelo utilizado de estimación de la postura

en matrices de distancias euclídeas o *Euclidean Distance Matrices* (EDM). Una EDM es una matriz cuadrada simétrica cuyas dimensiones son el número de *keypoints*. Cada elemento (i, j) de la matriz es la distancia euclídea entre los *keypoints* i y j. La figura 3.4 muestra un ejemplo ilustrativo de cómo se codifica la información de la persona detectada con esta estrategia.

Una vez obtenida la EDM se normaliza por la mayor de las dimensiones de la imagen original (las distancias se calculan en píxeles de la imagen original, y con esta normalización se elimina esa dependencia de la resolución de la cámara). La EDM tiene dimensiones de 17x17x1 en la mayoría de los casos, pero en el caso de *MobileNets*, sus modelos requieren que la entrada tenga como mínimo dimensiones de 128x128x3. Para ello, en el caso de usar un clasificador basado en *MobileNets*, la EDM se redimensiona con interpolación bilineal para pasar de 17x17 a 128x128, y para tener tres canales, se hacen dos réplicas del canal original.

Para entrenar los modelos hay que almacenar los datos en ficheros, siendo la primera opción la de convertirlos en ficheros de imagen PNG. Sin embargo, para evitar la pérdida de precisión se ha tomado la decisión de modificar la API usada para el entrenamiento de modelos de CNNs conocida como *Keras*[5] para aceptar ficheros en el formato NPY (formato por defecto de la librería *Numpy* de *Python*).



(a) (b) (c)

Figura 3.4: Ejemplo de detección de personas con el modelo de detector del **Reconocedor con Esqueleto**. (a) Imagen de entrada. (b) Resultado de la aplicación del modelo de estimación de la postura. (c) Representación en EDM de la persona que aparece en la imagen, convertida en imagen para facilitar la visualización: el violeta son las distancias más pequeñas y el amarillo las más grandes. Las columnas y las filas corresponden a los *keypoints* en el mismo orden que en la lista 3.1

Ventajas de cada representación. En cuanto a la comparación entre las dos representaciones escogidas, hay diferencias muy importantes. Por un lado, la representación del **Reconocedor con Segmentación** es simple y mantiene las características de correlación espacial entre los píxeles. Por otro, la representación del **Reconocedor con Esqueleto** recoge los datos de manera invariante a la apariencia de la persona o su entorno (iluminación, etc.), y el número de arquitecturas de clasificadores distintas capaces de trabajar con ella es mayor.

Desventajas de cada representación. El éxito del **Reconocedor con Segmentación** depende mucho de la heterogeneidad del conjunto de imáge-

[5]https://keras.io/

nes utilizadas para el entrenamiento. Por ejemplo, si se entrenase un clasificador sólo con usuarios varones adultos, es probable que fallase si una mujer o un niño utilizasen un reconocedor con ese modelo entrenado. Por su parte, el **Reconocedor con Esqueleto** tienen muchos menos datos "reales" (es decir, sin contar posibles reescalados), lo que es un indicador de que posiblemente aparezca sobreajuste a los datos de entrenamiento con facilidad.

3.2. Clasificación

La fase de clasificación se encarga de tomar la representación de la persona detectada en la fase anterior, y decidir cuál de los gestos definidos en el modelo está realizando esa persona.

Las condiciones en las que el modelo de clasificador escogido va a ser utilizado imponen importantes requisitos esperados del mismo:

En primer lugar, el sistema final va a ser un reconocedor de gestos en tiempo real y con la GPU ya ocupada por el detector de personas, por lo que una inferencia no debe requerir excesivo tiempo de ejecución y debe poder ser ejecutada eficientemente en CPU.

En segundo lugar, el conjunto de datos etiquetados disponible para el entrenamiento (datasets explicados a continuación en la subsección 3.4.1) contiene 1393 imágenes con 8 clases/gestos y 2035 imágenes de una clase adicional desconocido ("unknown", todo movimiento que no pertenezca a ningún gesto definido). Estas cantidades son pequeñas en comparación con el tamaño en número de imágenes típico para el entrenamiento de modelos de CNNs. Por estos motivos, los tipos de clasificadores candidatos a ser usados en esta fase deben ser eficientes y capaces de aprender con pocos datos de entrenamiento (esto elimina la posibilidad de entrenar CNNs muy complejas desde cero).

3.2.1. Clasificadores considerados

Los clasificadores que han sido considerados para el sistema se dividen en dos grupos. El primero de ellos es el de las *Support Vector Machines* (SVM). Este tipo de clasificadores son simples en su uso y son capaces de aprender con pocos datos de entrenamiento si tienen suficiente dimensionalidad. La implementación de los mismos que se ha usado es la existente en la librería *Scikit-Learn* existente en *Python*. Este paquete contiene herramientas de aprendizaje automático como los SVM ya implementadas[6]. Concretamente, las SVM para clasificación se denominan *Support Vector Classifiers* y todos son capaces de trabajar con la representación del **Reconocedor con Esqueleto**:

- **LinearSVC.** Es una SVM en la que los hiperplanos generados para la clasificación se formulan de manera lineal, es decir, su *kernel function* es lineal.
- SVC. Es una SVM en la que la *kernel function* es un parámetro en el momento de su creación (no tiene por qué ser lineal). Es muy versátil pero a grandes dimensionalidades de datos necesita demasiada memoria para realizar los cálculos, y por eso **no se usa** en este trabajo.

[6] https://scikit-learn.org/stable/modules/svm.html

- **NuSVC con RBF kernel.** Similar al SVC pero con un parámetro (Nu, ν) que disminuye los cálculos realizados y la memoria necesaria para su uso. La *kernel function* empleada es una función de base radial (RBF).

- NuSVC con Polynomial kernel. El mismo tipo que en el caso anterior, pero con una función polinomial de grado 3 como *kernel function*. Se han realizado pruebas pero ha sido **descartada** rápidamente por producir resultados muy inferiores a las anteriormente mencionadas, incluso peor cuando se aumenta el grado de la función.

El segundo grupo de clasificadores candidatos es el de CNNs. Como se ha mencionado, el tamaño de las CNNs usadas en la clasificación no puede ser muy grande por las limitaciones en recursos y datos de entrenamiento. Además, como parte del trabajo está centrado en el aprendizaje sobre el uso de redes neuronales, se han realizado pruebas con pequeñas redes programadas *ad hoc*, además de otras redes de clasificación ya establecidas. Todas ellas son capaces de aprender de la representación del **Reconocedor con Esqueleto**:

- **1-Dense.** Esta red contiene únicamente una capa oculta del tipo *fully connected*.

- **4-Dense.** Una ampliación de la anterior, que contiene cuatro capas ocultas del tipo *fully connected* intercaladas con capas de regularización.

- **2-ConvNN.** Esta red contiene varios tipos de capas, siendo dos de ellas de convolución. La arquitectura ha sido extraída de un tutorial disponible en *Keras*[7]

- **6-ConvNN.** Una ampliación de la anterior, esta vez con seis capas de convolución.

- **27-ConvNN.** Red específicamente diseñada para comprobar si el número de capas de convolución es proporcional a su precisión, por lo que se usaron 27 capas de convolución.

- **MobileNets.** Arquitectura propuesta en [10]. Los modelos han sido escogidos con el parámetro $\alpha = 1{,}0$, y preentrenados en el dataset ImageNet[8], ya que como se ha comentado, no se dispone de datos suficientes para entrenar un modelo grande desde cero. Por ello es necesario inicializarlo con los pesos de un clasificador de objetos existente y realizar un *fine-tuning* para las clases objetivo del proyecto. Como se ha mencionado, este tipo de redes es capaz de clasificar no sólo las EDMs, sino que también puede aprender de la representación del **Reconocedor con Segmentación**.

Ya se han explicado todos los posibles modelos para los bloques internos del reconocedor de gestos, tanto para el **Reconocedor con Segmentación** como para el **Reconocedor con Esqueleto**. El siguiente paso es concretar las prestaciones adicionales que este reconocedor de gestos que se exploran en este trabajo y que no existían en el reconocedor previo.

[7] urlhttps://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py
[8] http://www.image-net.org/

3.3. Modificaciones propuestas respecto al reconocedor previo

Como se ha mencionado en el capítulo introductorio, uno de los principales objetivos de este trabajo es crear un prototipo de reconocimiento de gestos mejor que un prototipo existente en el repositorio del grupo de investigación[9]. Por ello se va a explicar la arquitectura interna de ese prototipo, en vistas de las mejoras que se proponen más adelante para el prototipo desarrollado en este trabajo.

3.3.1. Reconocedor previo

El reconocedor de gestos del prototipo previo no tiene una estructura interna tan modular como la utilizada al principio de este capítulo para definir el reconocedor de este trabajo en la figura 3.1 (el código del detector y el clasificador no está dividido en módulos independientes). Sin embargo, se va a seguir ese esquema para ver las similitudes y diferencias entre ambos:

1. Detector de personas: El sistema de detección de personas [12] presente en el prototipo previo también está basado en CNNs. Para detectar personas, la CNN recibe una imagen y genera dos tipos de salidas: mapas que contienen la situación de los *keypoints* (*Part Confidence Maps*) y campos que identifican miembros del cuerpo que unen los *keypoints* (*Part Affinity Fields*).

2. Representación: Esas dos salidas, a continuación reciben un procesado también explicado en la publicación. Se realiza un filtrado conjunto de ambas, que permite identificar qué *keypoints* une cada miembro para poder generar el "esqueleto" de las personas en la imagen (en forma de lista de coordenadas).

3. Clasificador: Este es el tramo en el que más difieren el reconocedor del prototipo previo y el reconocedor desarrollado en este proyecto. En lugar de una solución basada en el aprendizaje automático, se optó por una solución heurística sustentada en que la aplicación de ese sistema era reconocer gestos para generar comandos de movimiento. La heurística consiste en calcular el ángulo de orientación del **brazo derecho** (con los datos del "esqueleto" de la representación), y subdividir el espacio en intervalos de ángulos, con cada intervalo correspondiendo a un gesto. El paso final es buscar en qué intervalo se encuentra el ángulo calculado del brazo y tomar como salida del reconocedor el gesto asociado a ese intervalo.

Una característica relevante de este reconocedor, es que siempre que se ejecuta responde con uno de los gestos definidos. Es decir, requiere de una señal de activación que le indique cuándo debe actuar, porque sin ella, "reconocería" gestos incluso cuando el usuario no estuviese realizando ninguno.

3.3.2. Mejoras exploradas

Según las características que se buscaban en el reconocedor de gestos, se han realizado cambios en las condiciones en el entrenamiento con el objetivo

[9] `https://github.com/anacmurillo/unizar_interactive_robotics`

aumentar las prestaciones ofrecidas. En este trabajo se han implementado varias mejoras en el reconocedor para obtener un sistema más general y robusto, sobre todo en dos aspectos:

- **Automatización de activación del reconocedor:** En un principio se espera que el reconocedor sólo sea capaz de identificar gestos conocidos, es decir, no producir un gesto como salida de cada imagen sino sólo producirlo si ese gesto está definido en el modelo y el reconocedor está seguro de que es correcto. Esto contribuiría a la automatización del proceso, eliminando los mecanismos de activación necesarios en el prototipo previo.

  Para conseguir esto, una primera solución propuesta es hacer uso de una clase adicional "unknown" en el entrenamiento de los modelos para que sean capaces de discriminar las imágenes cuyo gesto no encaja en ninguna de las clases principales.

  En caso de que la clase "unknown" genere problemas en los modelos (como limitar su precisión media notablemente, por ejemplo), se ha formulado otra posible solución al problema: crear un sistema de votación a la salida del clasificador que guarde memoria sobre los últimos gestos reconocidos, y sólo acepte los gestos que han sido reconocidos un número determinado de imágenes seguidas, lo que asegura su validez.

- **Reconocer gestos con ambos brazos:** Todos los gestos contenidos en los datasets están realizados con el brazo derecho. Esto restringiría los modelos entrenados con estos datos a sólo reconocer gestos realizados con el brazo derecho.

  No obstante, gracias a la simetría del cuerpo humano, es posible voltear las imágenes horizontalmente y entrenar los modelos con las imágenes originales y las volteadas a la vez (modificando las etiquetas de clase acordemente). De este modo se cumplen dos objetivos: el reconocimiento de un gesto es independiente del brazo usado, y el número de datos de entrenamiento se duplica, lo que es beneficioso ya que el escaso número de imágenes de entrenamiento era una preocupación importante desde el inicio.

Una de las herramientas con las que se ha experimentado en estas pruebas es la de normalizar los pesos de las clases en el entrenamiento. Esta normalización de pesos de las clases hace que las diferencias entre cantidades de datos de entrenamiento de cada clase influya en que un modelo asigne una u otra con más frecuencia. Se entrenaron varios modelos sin realizar esta normalización, pero se observó (sobre todo cuando se hacía uso de la clase "unknown", puesto que esta tiene muchos más datos que las demás) que no hacer la normalización de pesos era siempre negativo en cuanto a la precisión del modelo.

3.4. Evaluación del reconocedor de gestos

Las opciones propuestas, tanto en diferentes modelos como en mejoras posibles, para el reconocedor de gestos que se está desarrollando tienen capacidad para cumplir en mayor o menor medida la tarea de reconocer gestos en imágenes. Sin embargo, se quiere elegir la opción que cumpla esta tarea de la forma

más certera, y por ello se ha hecho una evaluación de todos ellos en igualdad de condiciones.

3.4.1. Datos y métricas

Para realizar las pruebas de evaluación de las distintas opciones ha sido necesario escoger y definir los conjuntos de datos (*datasets*) disponibles para entrenamiento y evaluación.

Clases. Todos estos datos están etiquetados para reconocer el mismo número y tipo de clases (gestos): 8 clases de gestos, con una clase adicional de gesto desconocido ("unknown"). Ejemplos de imágenes pertenecientes a estas clases se pueden observar en la figura 3.5.

Figura 3.5: Ejemplos de imágenes que corresponden a cada tipo de clase. La clase asignada a una imagen (la etiqueta justo debajo de cada una) es la dirección en la que la persona está señalando. La clase "unknown" es la más heterogénea porque contiene todas las imágenes en las que el gesto no está claro.

Datasets. Los datasets disponibles (resumidos en la tabla 3.1, con ejemplos representativos en el apéndice B) son:

- **ETH:** Denominado así porque fue capturado en la universidad ETH Zurich, en el laboratorio V4RL (*Vision for Robotics Lab*). Procede de un proyecto anterior realizado en el grupo de investigación, y fue grabado

todo en la misma habitación. Para estas pruebas, se ha realizado una división de este dataset para entrenar varios (5) modelos, hacer validación cruzada en los datos y poder obtener promedios de su precisión. En cada una de las 5 variaciones se separan los datos en dos grupos:

- **ETH-1 (*train-fold*):** Las imágenes pertenecientes a cuatro de los cinco usuarios, usadas para entrenar un modelo.
- **ETH-2 (*test-fold*):** Las imágenes pertenecientes al usuario restante, usadas para realizar la validación cruzada de ese modelo.

Esta estrategia asegura que todos los datos han sido utilizados en alguna prueba para entrenar y en alguna otra para evaluar, pero nunca entrenando y evaluando con datos demasiado parecidos (es decir, provenientes del mismo usuario).

- **I3A:** Denominado así porque fue capturado en las instalaciones del Instituto de Ingeniería e Investigación de Aragón (I3A). Ha sido grabado, procesado y etiquetado dentro de este proyecto, y contiene imágenes en entornos distintos. En estas pruebas se ha utilizado este dataset como datos de test para conseguir una estimación final de la precisión que se puede esperar de los clasificadores, y se utiliza entero para cada uno de los (5) modelos entrenados en la validación cruzada mencionada.

	Dataset **ETH**	Dataset **I3A**
Nº usuarios	5	3
Nº vídeos	5	5
Indoors / Outdoors	5/0	3/2
Usuario apunta con brazo Izq./Dch.	Dch.	Dch.
Nº clases	8+"unknown"	8+"unknown"
Nº imágenes de las 8 clases (repartidas equitativamente)	1393	3212
Nº imágenes del tipo "unknown"	2035	2093

Tabla 3.1: Tabla de resumen del contenido de los dos datasets empleados en este trabajo.

La métrica utilizada para la evaluación de cada arquitectura es la precisión media por clase de todas las inferencias realizadas con los modelos de esa arquitectura, como se muestra en la ecuación 3.2.

$$AP_{\text{per class}} = \frac{\sum_{i=1}^{m} \frac{s_i}{n_i}}{m} \tag{3.2}$$

Donde:

m es el número de clases

s_i es el número de imágenes correctamente reconocidas de la clase i

n_i es el número total de imágenes reconocidas de la clase i

3.4.2. Experimentos

Dados los detalles de los datos y métricas a utilizar detallados en la sección anterior, se ha realizado el entrenamiento y las pruebas de evaluación de las arquitecturas candidatas. En las Tablas 3.2 y 3.3 se muestra un resumen de los datos recogidos durante estas pruebas. El sufijo "-A" significa que han sido entrenados y evaluados con imágenes **tanto del brazo derecho como del izquierdo**. Si no aparece este sufijo, es porque esos modelos sólo han sido entrenados y evaluados con imágenes del brazo derecho. El sufijo "-U" significa que los modelos han sido entrenados **sin normalizar los pesos** de las clases según el número de datos de entrenamiento. Resultados más completos se encuentran recogidos en el anexo C.

	Entr:ETH-1—Eval:**ETH-2**		Entr:ETH-1—Eval:**I3A**	
	8 clases	8 clases + "unknown"	8 clases	8 clases + "unknown"
LinearSVC-U	92.75 (6.50)	68.11 (25.19)	70.62 (21.10)	61.22 (25.75)
LinearSVC	92.75 (6.24)	**80.11** (15.98)	**71.25** (19.77)	**62.89** (21.47)
LinearSVC-A	93.12 (5.18)	79.89 (11.34)	58.00 (19.12)	51.33 (20.10)
NuSVC	**93.75** (9.23)	72.44 (16.90)	65.75 (18.61)	55.89 (20.05)
NuSVC-A	84.75 (12.53)	69.44 (11.60)	53.12 (13.90)	52.44 (15.20)
1-Dense-A	86.87 (13.17)	71.89 (12.49)	61.37 (21.36)	53.44 (15.19)
4-Dense-A	84.50 (6.40)	69.22 (10.59)	69.87 (13.09)	48.44 (16.96)
2-ConvNN-A	91.62 (7.19)	**83.44** (11.35)	**72.25** (11.07)	**66.22** (15.52)
6-ConvNN-A	**95.62** (3.35)	78.78 (13.33)	66.00 (16.08)	56.22 (17.85)
27-ConvNN-A	86.87 (11.16)	78.89 (15.72)	48.50 (21.10)	41.44 (17.71)
MobileNet-A	91.00 (9.87)	68.00 (20.72)	70.62 (14.28)	38.67 (27.58)

Tabla 3.2: Precisión media por clase y desviación típica entre paréntesis entre 5 modelos, de los modelos entrenados para el **Reconocedor con Esqueleto**

El primer conjunto de experimentos ha sido realizado con los modelos posibles para el **Reconocedor con Esqueleto**. De los experimentos recogidos en la tabla 3.2 se han extraído algunas observaciones:

Experimento 1: Normalización: ¿sí o no? Como se ha mencionado antes, se ha intentado comprobar si la normalización en los pesos de las clases de entrenamiento es necesaria. En el caso de que haya clases con muchos más datos que las demás (la columna de *8 clases + "unknown"*), se observa mucha diferencia entre *LinearSVC-U* y *LinearSVC*. Esto que indica que, en efecto, la normalización de los pesos de las clases es necesaria en el entrenamiento de modelos.

Experimento 2: Generalización de un modelo a imágenes de prueba de otro dataset. La precisión entre la evaluación con datos cuyo dataset de procedencia es el mismo que los de entrenamiento (ETH) y la evaluación con datos de un dataset distinto (I3A) es muy distinta. Concretamente estos modelos disminuyen entre un 20 % y 30 % su precisión. Esta diferencia es normal, ya que la generalización a otro entorno o cualquier tipo de escenario es muy compleja. Hay diferencia notable en este aspecto entre los modelos *LinearSVC* y *LinearSVC-A*. Esto indica que la arquitectura SVM con un kernel lineal no

generaliza bien el uso de ambos brazos, y cuando se pasa a datos de test, esto repercute en su precisión. Este hecho también corrobora que los datos de test no han sido utilizados en la fase de entrenamiento y que son distintos, por lo que los modelos cometen más errores.

Experimento 3: Adición de la clase "unknown". La diferencia entre utilizar la clase "unknown" y no usarla influye notablemente en la precisión de los modelos, pero obviamente, considerar dicha clase implica la automatización del sistema a la hora de decidir cuándo detectar gestos y cuándo no, por lo que es de interés su estudio. Comparando las columnas *8 clases* con sus respectivas columnas *8 clases + "unknown"*, se observa que se pierde en torno a un 10 % de precisión al incluir esta clase adicional. No es sorprendente, ya que la clase "unknown" es una clase muy heterogénea e incluye imágenes que son cercanas a las que aparecen en todas las demás clases. Por lo tanto estos casos (a veces difíciles hasta para un humano) repercuten en los resultados.

Experimento 4: Mejores modelos en entrenamiento contra mejores modelos en test. En el entrenamiento con *8 clases*, los mejores clasificadores son *NuSVC* entre las SVMs y *6-ConvNN-A* entre las CNNs. Sin embargo, en las otras tres mediciones, el clasificador SVM *LinearSVC* y el clasificador CNN *2-ConvNN-A* funcionan notablemente mejor que ellos en media. Una de las posibles razones por las que esta disparidad entre precisión en entrenamiento y test ocurre, es la de que la complejidad de un modelo es un factor en la aparición de sobreajuste a los datos. Si un modelo es demasiado complejo (tiene demasiados parámetros a entrenar en relación a la complejidad de los datos), es posible que aprenda con demasiada exactitud de los datos de entrenamiento, y esto tiene un impacto negativo cuando se evalúa con otros datos distintos. Por esto es tan importante tener siempre un buen conjunto de datos independientes de test.

Experimento 5: ¿Cuantas más capas convolucionales mejor? Como se han ordenado las CNNs en función de su complejidad en término de número de capas, se observa que el tamaño no posee una relación proporcional a la precisión en los datos. Concretamente, en los resultados del clasificador *27-ConvNN-A* (que había sido incluído para comprobar si un gran número de capas de convolución era mejor sin tener en cuenta nada más de la arquitectura) se puede observar un ejemplo evidente de sobreajuste a los datos, en el que la precisión se reduce a casi la mitad cuando se pasa de datos de validación en entrenamiento a datos de test.

Experimento 6: Modelos más precisos para el Reconocedor con Esqueleto Según los datos de este primer conjunto de experimentos, tanto el clasificador *LinearSVC* como el clasificador *2-ConvNN-A* tienen las mejores características para ser el clasificador escogido en el sistema final.

A continuación se recoge el segundo conjunto de pruebas realizadas, en este caso modelos para el **Reconocedor con Segmentación**. Estos datos, resumidos en la tabla 3.3, y las observaciones previas permiten sacar conclusiones sobre las pruebas de evaluación:

	Entr:ETH-1—Eval:**ETH-2**		Entr:ETH-1—Eval:**I3A**	
	8 clases	8 clases + "unknown"	8 clases	8 clases + "unknown"
MobileNet-U	-	60.00 (27.33)	-	52.78 (28.34)
MobileNet-UA	-	58.67 (19.69)	-	63.67 (24.21)
MobileNet	87.12 (11.88)	**69.11** (29.60)	78.12 (17.64)	60.33 (24.39)
MobileNet-A	**87.62** (6.20)	66.33 (20.01)	**86.37** (10.71)	**75.11** (23.23)

Tabla 3.3: Precisión media por clase y desviación típica entre paréntesis entre 5 modelos, de los modelos entrenados para el **Reconocedor con Segmentación**

Experimento 7: ¿Se mantienen las diferencias entre entrenamiento y test? La representación de los datos del **Reconocedor con Segmentación** y la arquitectura de las *MobileNets* consiguen una mayor generalización de los datos: las diferencias entre precisión de evaluación en entrenamiento y precisión de test no es tan grande (en torno a 10 %) en comparación con las diferencias de 20 % y 30 % observadas en las pruebas con la representación del **Reconocedor con Esqueleto**.

Experimento 8: ¿Es beneficioso aprender con ambos brazos (es decir, realizar volteo de las imágenes)? El volteo de las imágenes para aumentar el número de datos de entrenamiento es muy positivo en comparación a su precisión sin aplicar esta mejora. Se comprueba en que los reconocedores con el sufijo "-A" tienen mayor precisión media y menor desviación típica que los correspondientes sin el sufijo. Esto es un reflejo de la naturaleza de las CNNs, que se benefician mucho de la abundancia de datos de entrenamiento, y su escasez hace que el entrenamiento quede incompleto. Esto tiene un impacto importante en el sistema final, ya que esta mejora permite quitar la restricción de usar sólo el brazo derecho que habría si no se hubiese realizado este cambio.

Experimento 9: Elección del reconocedor más certero. Observando las precisiones medias obtenidas en datos de test (I3A) tanto con modelos de *MobileNets* en el **Reconocedor con Segmentación** como en los clasificadores entrenados con la representación del **Reconocedor con Esqueleto**, se observa que el clasificador *MobileNet-A* tiene las mejores características para la aplicación que se está buscando.

La conclusión de los experimentos de evaluación es que se ha comprobado que de los dos tipos de reconocedores propuestos, el **Reconocedor con Segmentación** produce mejores resultados. Dentro de este, se encuentra el detector de personas con un modelo de segmentación semántica de *Mask R-CNN*; el procesamiento de su salida que elimina el fondo de la imagen original para representar a la persona; y el clasificador de gestos, con el modelo *MobileNet-A* escogido de entre los evaluados en esta sección.

Capítulo 4

Prototipo desarrollado

Este capítulo contiene un visión completa del prototipo desarrollado, incluyendo el reconocedor de gestos con los modelos escogidos en el capítulo 3 que se introducen en un simulador para realizar pruebas de integración del sistema.

4.1. Visión general

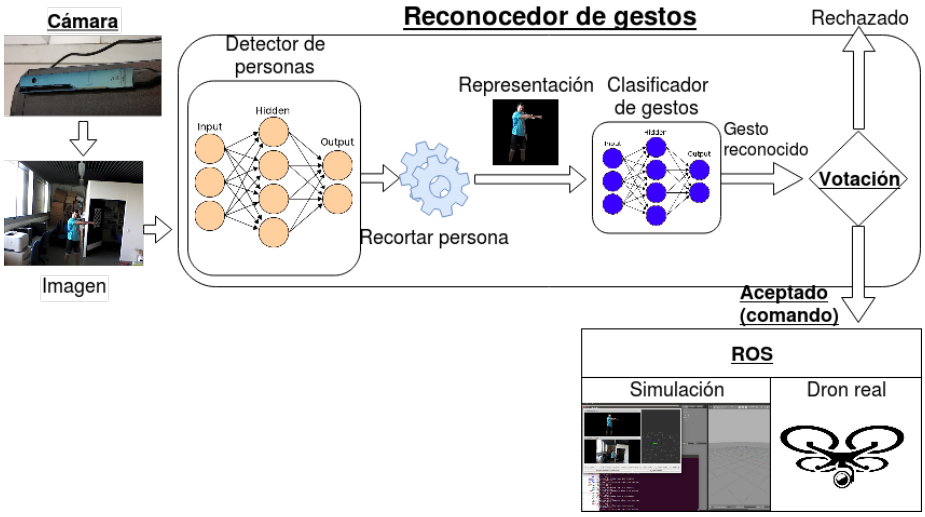


Figura 4.1: Diagrama del prototipo desarrollado. Las imágenes de la cámara se pasan al reconocedor de gestos que decide qué gesto es el realizado y con el sistema de votación se acepta o rechaza el gesto. Si se rechaza, se acaba ese ciclo de ejecución. Si se acepta, se envía como comando a ROS para que lo use en la simulación o en el dron real.

El sistema desarrollado utiliza como punto de partida un prototipo de guiado de drones con reconocimiento visual de gestos existente en el repositorio del grupo de investigación[1]. La razón por la que se ha decidido usar este prototipo como base, es porque ya tiene programada la comunicación entre procesos

[1] `https://github.com/anacmurillo/unizar_interactive_robotics`

haciendo uso del simulador ROS[2] y una interfaz gráfica que permite probar el sistema.

La figura 4.1 muestra un diagrama del sistema completo desarrollado en este proyecto con sus componentes principales, que son los siguientes:

Cámara. La cámara utilizada para capturar las imágenes del prototipo es el modelo Intel® RealSense™Camera R200[3]. Esta cámara es una de las disponibles en el laboratorio del grupo de investigación y tiene como ventaja de que existen drivers en ROS[4] que facilitan la integración de esta cámara con el prototipo.

Reconocedor de gestos. Como se ha explicado en el capítulo 3, el reconocedor de gestos escogido sigue el esquema que se ha denominado **Reconocedor con Segmentación**:

- Detector de personas: utiliza un modelo de red neuronal profunda de segmentación semántica. La arquitectura de la red se llama *Mask R-CNN* [14]. El modelo entrenado se ha obtenido del repositorio oficial[5], implementado en el *framework Caffe2*. Las inferencias del detector de personas se ejecutan en la GPU del equipo. Este modelo obtiene los segmentos de la imagen de entrada, y después se utilizan estos segmentos para recortar la persona y usar esa parte de la imagen como representación.
- Clasificador de gestos: utiliza otro modelo de red neuronal profunda, esta vez de clasificación. La arquitectura pertenece al tipo de redes llamado *MobileNets* [10]. Como la GPU está ocupada por el detector de personas, la clasificación se realiza en la CPU del equipo, ya que este modelo es mucho más ligero computacionalmente. El modelo utilizado (en la API *Keras* sobre el *framework Tensorflow*) está pre-entrenado en el dataset *ImageNet*, y el *fine-tuning* se ha realizado con el dataset ETH. Este modelo se ha denominado en el trabajo *MobileNet-A*. El modelo recibe como entrada la imagen de la persona segmentada y produce como salida una etiqueta y un valor de confianza entre 0 y 1 (e.g. {"arriba-derecha", 0,89}) que representan un gesto.

Votación. Mencionado en la subsección 3.3.2. Las prestaciones del reconocedor disminuyen mucho usando la clase adicional "unknown" para automatización del descarte de imágenes que no corresponden con ningún gesto conocido. Esta disminución se debe a que la clase "unknown" es mucho más difícil de aprender que una clase normal, pues cubre un espectro de "todo lo demás" que es difícil de recoger en datos para entrenar modelos. Por lo tanto, se ha optado por conseguir la automatización del proceso añadiendo este bloque final de votación en el sistema. Este bloque recibe el gesto reconocido por el clasificador, comprueba si su valor de confianza está por encima de un umbral predefinido (0,5), y guarda información sobre los últimos gestos reconocidos. En un inicio

[2] `http://www.ros.org/`

[3] `https://software.intel.com/sites/default/files/managed/d7/a9/realsense-camera-r200-product-datasheet.pdf`

[4] `http://wiki.ros.org/RealSense`

[5] En el conjunto 12_2017_baselines, el modelo e2e_mask_rcnn_R-101-FPN_2x [17]

rechaza todos los gestos (lo que finaliza esos ciclos de ejecución), hasta que ha recibido varias veces seguidas el mismo gesto (3 gestos seguidos iguales, de los 4 últimos) con alto valor confianza, y entonces lo acepta.

Comando. Un comando es el mensaje enviado a ROS correspondiente un gesto que ha sido aceptado por el sistema de votación. El mensaje es un trío de coordenadas x, y y z, de la posición relativa a la que el dron se debe mover, es decir, contando su posición actual como el origen de coordenadas. Por simplicidad y para hacer comparaciones con el prototipo previo, en este trabajo se ha optado por hacer un mapeo de los gestos al plano horizontal (es decir, el dron se mueve en su plano horizontal, no hacia arriba o hacia abajo).

ROS. Este bloque es en el que se hace uso de las ventajas que proporciona ROS, porque ejerce la función de *wrapper* independientemente de si el sistema se ha lanzado con el objetivo de realizar una simulación o para dirigir un dron real. El comando se ejecuta y el ciclo finaliza con éxito.

4.2. Pruebas de integración del prototipo

Para comprobar el funcionamiento correcto del sistema y poder hacer comparaciones con el prototipo previo, se han diseñado pruebas de integración que son realizadas en el laboratorio del grupo de investigación, en el I3A.

El **equipamiento** utilizado para realizar estas pruebas es:

- Procesador: Intel® Core™i7-6700 CPU @ 3.40GHz x 8
- RAM: 32GB
- GPU: GeForce GTX 1070/PCIe/SSE2

El entorno de **monitorización** utilizado se muestran en la figura 4.2. Detalles y pasos para su lanzamiento se encuentran recogidos en el anexo D.

Los **experimentos** realizados se encuentran resumidos en la tabla 4.1. El procedimiento seguido para realizar estos experimentos es el siguiente:

- Se realiza el mismo experimento, en las mismas condiciones con los dos prototipos: el previo y el desarrollado en este trabajo.
- El experimento consiste en reconocer 20 veces cada gesto (10 con cada brazo) de los 8 descritos en la sección 3.4.1, sin contar la clase "unknown" ya que ninguno de los prototipos está preparado para reconocerla.
- Los gestos los realizarán 2 usuarios distintos, y ninguno de ellos ha aparecido en el entrenamiento de ninguno de los modelos usados en los prototipos.

Las figuras 4.3, 4.4 y 4.5 muestran unos cuantos ejemplos representativos que ayudan a ilustrar las observaciones que se discuten a continuación.

El prototipo previo reconoce correctamente gestos claros (ver ejemplos (a) y (b)), pero falla al complicarse la postura o la visibilidad del brazo derecho en la imagen de entrada (ver ejemplo (c)). Además, al reconocer siempre el

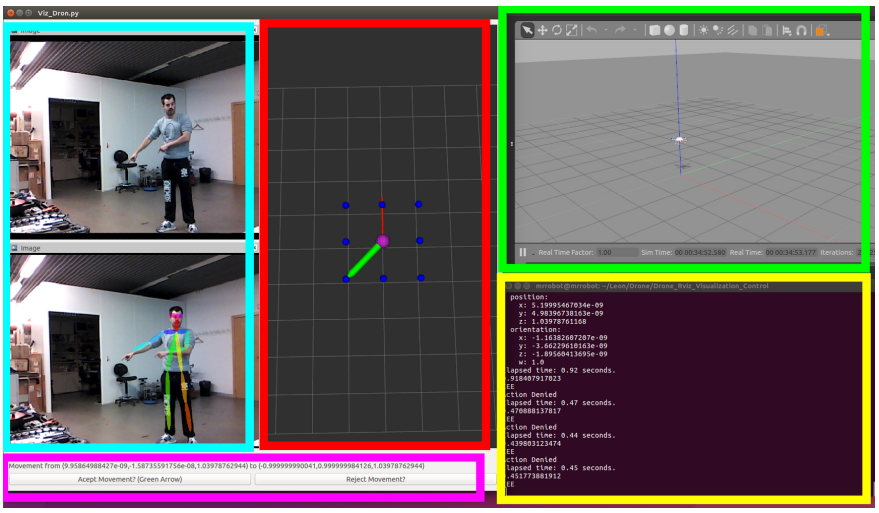


Figura 4.2: *Setup* de la simulación, con las partes importantes recuadradas con colores. **Azul:** imagen de entrada e imagen procesada. **Rojo:** Comando reconocido. Si tiene la misma orientación que el brazo del usuario en la foto, el reconocimiento ha sido correcto. **Rosa:** Botones para aceptar o rechazar el comando desde ROS. **Verde:** Dron simulado. **Amarillo:** Salida del programa, con datos útiles para el programador.

Clase de gesto	Prototipo previo	Prototipo de este trabajo
"arriba"	$(10+0)/20$ (50 %)	$(6+5)/20$ (55 %)
"arriba-derecha"	$(5+0)/20$ (25 %)	$(8+10)/20$ (90 %)
"derecha"	$(10+0)/20$ (50 %)	$(8+6)/20$ (70 %)
"abajo-derecha"	$(7+0)/20$ (35 %)	$(10+8)/20$ (90 %)
"abajo"	$(8+10)/20$ (90 %)	$(10+10)/20$ (100 %)
"abajo-izquierda"	$(7+0)/20$ (35 %)	$(8+10)/20$ (90 %)
"izquierda"	$(9+0)/20$ (45 %)	$(9+10)/20$ (95 %)
"arriba-izquierda"	$(10+0)/20$ (50 %)	$(9+7)/20$ (80 %)
Precisión Media	**47.5 %**	**84.12 %**

Tabla 4.1: **Precisión media en el reconocimiento de cada clase.** Resumen de las pruebas de integración y evaluación. Se compara el prototipo existente en el grupo de investigación y el desarrollado en este trabajo. Se mide su precisión para reconocer 8 gestos con ambos brazos.
Notación: (*aciertos con brazo derecho* + *aciertos con brazo izquierdo*) / *total de intentos*

brazo derecho en los ejemplos de entrenamiento, si se usa el brazo izquierdo (ver ejemplo (d)) el reconocedor falla en prácticamente todos los casos (a no ser que ambos brazos apunten en la misma dirección, lo cual sería lo mismo que usar sólo el derecho).

En comparación, el prototipo desarrollado en este trabajo reconoce bien tanto gestos claros como otros más complicados (ver ejemplos (e), (f) y (g)), ya se hayan realizado con el brazo derecho o con el izquierdo, con lo cual verificamos una de las mejoras deseadas en el nuevo sistema, de un comportamiento más robusto y fiable. Sin embargo, no es un sistema perfecto ya que tiene problemas para reconocer la clase "arriba" (55 %) (ver ejemplo (h)). Una posible explica-

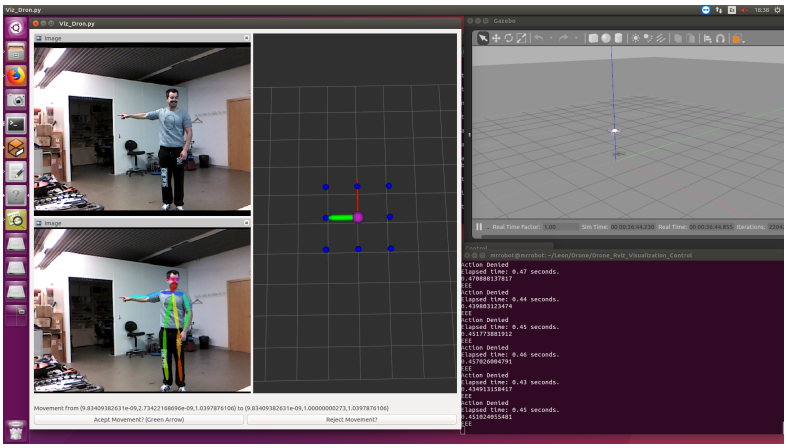
ción de este fenómeno es que cuando el clasificador de gestos recibe la imagen de la persona recortada apuntando hacia arriba, se fije demasiado en la silueta y no sepa si es una persona de alta estatura señalando hacia abajo, o una persona de estatura normal señalando hacia arriba. Esto provoca que el sistema de votación no acepte el comando durante un largo periodo de tiempo y al final reconozca otro gesto distinto. Esta inconveniencia tiene muchas posibles soluciones: desde más datos de entrenamiento hasta equilibrar pesos en el entrenamiento o mejorar el sistema de votación para valorar mejor los gestos reconocidos a lo largo del tiempo.

Además, se ha calculado el tiempo medio de procesamiento de una imagen por cada prototipo, y se muestra en la tabla 4.2. Se han calculado los tiempos medios de cada tramo, así como la suma total. Hay que tener en cuenta que el modelo usado en una fase condiciona a las siguientes. Por ello, se observa que en el prototipo previo aunque parece que la detección es más rápida, el procesamiento necesario posterior es mucho más costoso. Se comprueba que el prototipo creado en este trabajo es el doble de rápido en su ejecución, llegando a alcanzar los 4 o 5 *fps*.

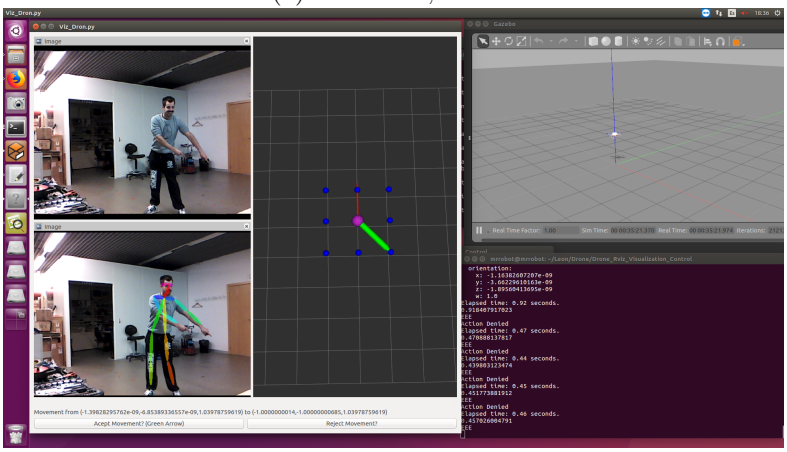
Acción medida	Prototipo previo	Prototipo de este trabajo
Detectar a la persona	146,36ms	201,81ms
Representación	278,09ms	11ms
Clasificar el gesto	1ms	13,54ms
Total por imagen	481**ms**	227**ms**

Tabla 4.2: **Tiempo medio de ejecución en el reconocimiento de gestos.** Se comparan el prototipo previo y el desarrollado en este trabajo. Los valores corresponden al tiempo medio de realizar esa parte sobre una única imagen, con el sistema en caliente (es decir, descartando la primera ejecución porque tarda más que las demás ya que es la que carga los modelos).

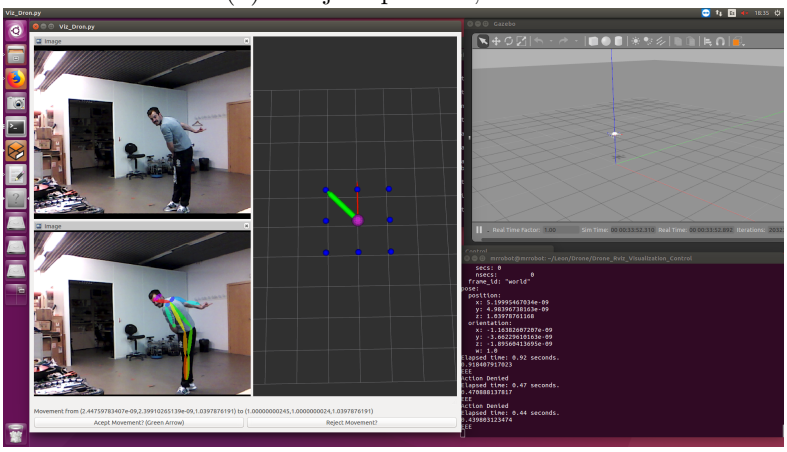
En cuanto al funcionamiento del sistema completo, el prototipo previo tiene el problema de que sólo tiene en cuenta la imagen de entrada actual para reconocer un gesto, lo que obliga a que en el momento de su lanzamiento, la persona debe estar colocada en su posición realizando el gesto. Esto no ocurre en el prototipo desarrollado porque el sistema de votación se asegura de que no sólo la imagen actual produzca un gesto válido, sino que varias imágenes anteriores contengan ese gesto antes de aceptarlo como un comando. Esta característica mejora la fluidez de su uso y la automatización en su uso.



(a):"derecha", **Éxito**

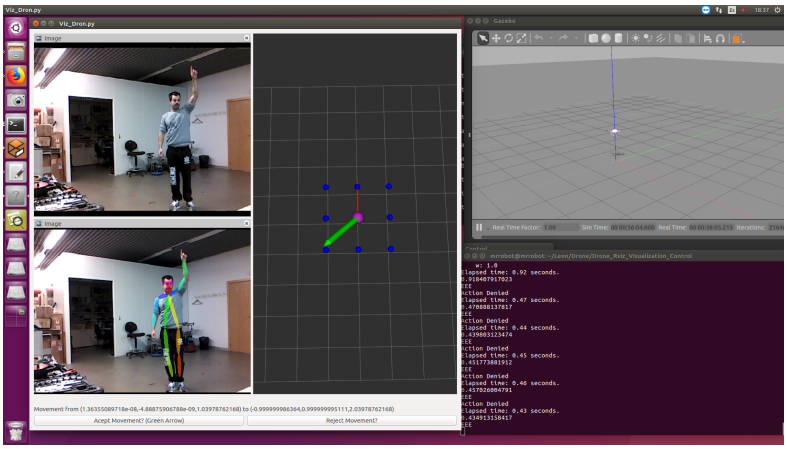


(b):"abajo-izquierda", **Éxito**

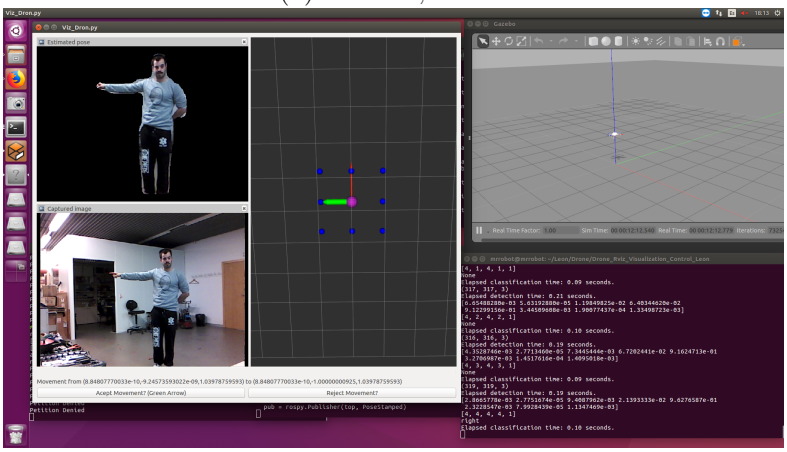


(c):"izquierda", **Error**

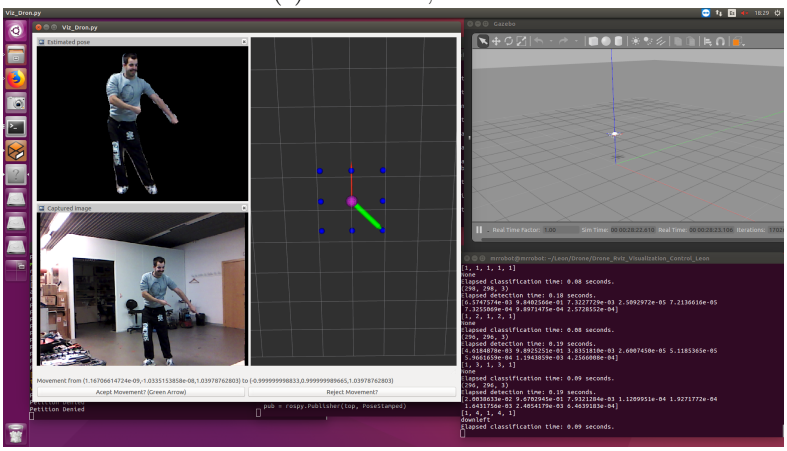
Figura 4.3: Ejemplos de gestos reconocidos en las pruebas de integración y comparación del prototipo previo (imágenes (a),(b) y (c)). Para saber si el reconocimiento ha sido un éxito, hay que comprobar que la flecha verde (en la parte central de cada imagen) apunta en la misma dirección que la persona en la imagen (en la parte izquierda de cada imagen).



(d):"arriba", **Error**

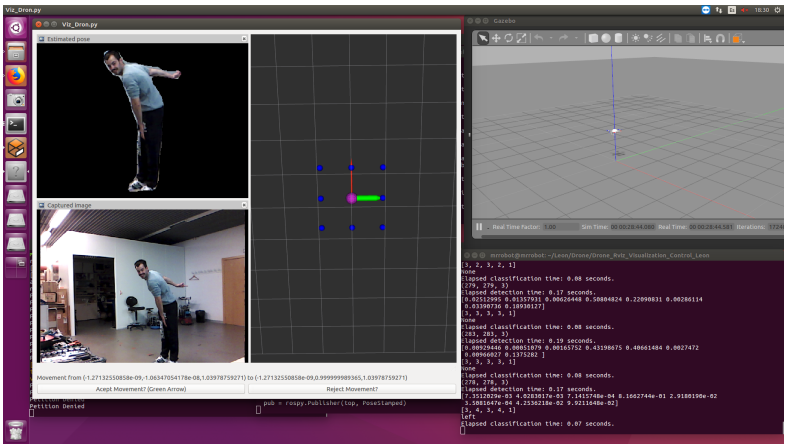


(e):"derecha", **Éxito**

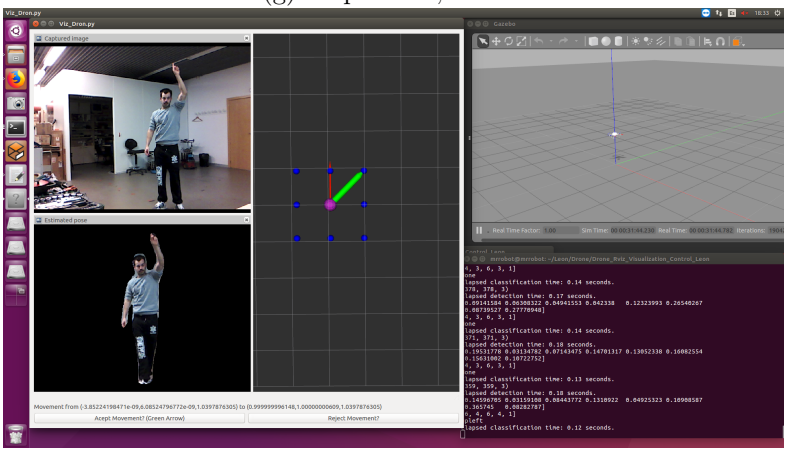


(f):"abajo-izquierda", **Éxito**

Figura 4.4: Ejemplos de las pruebas de integración y comparación del prototipo previo (imagen (d)) y del de este trabajo (imágenes (e) y (f)). Para saber si el reconocimiento ha sido un éxito, hay que comprobar que la flecha verde apunta en la misma dirección que la persona en la imagen.



(g):"izquierda", **Éxito**



(h):"arriba", **Error**

Figura 4.5: Ejemplos de gestos reconocidos en las pruebas de integración y comparación del prototipo de este trabajo (imágenes (g) y (h)). Para saber si el reconocimiento ha sido un éxito, hay que comprobar que la flecha verde (en la parte central de cada imagen) apunta en la misma dirección que la persona en la imagen (en la parte izquierda de cada imagen).

Capítulo 5

Conclusiones

En este capítulo se recogen diversas conclusiones sacadas en el transcurso del proyecto, tanto a nivel técnico como personal. Además se incluyen algunos de los mayores obstáculos encontrados en el desarrollo de las tareas, junto con algunos posibles enfoques que se podrían tomar con el objetivo de extender en un futuro el trabajo aquí realizado.

5.1. Conclusiones técnicas

El campo del aprendizaje automático es una herramienta potente para mejorar sistemas y hacerlos más intuitivos y convenientes. Dentro de él, el *deep learning* ha permitido avances enormes en ámbitos de visión por computador gracias a la aparición de las CNNs. En este proyecto, estas técnicas han sido la base para crear un sistema que evite el uso de un mando o control remoto a la hora de dirigir un dron, con el objetivo de quitar restricciones sobre el piloto.

Se ha estudiado la literatura referente a las CNNs, así como algunas de las arquitecturas más renombradas actualmente. Además, se ha trabajado con software específico del diseño e implementación de redes neuronales profundas, pudiendo así poner en funcionamiento las arquitecturas mencionadas e incluso implementando redes diseñadas *ad hoc*. También se han comparado gran cantidad de modelos entrenados que han permitido sacar diversas conclusiones antes de tomar decisiones sobre los modelos más aptos para la aplicación objetivo.

Se ha propuesto la mejor arquitectura desarrollada para el sistema, y se ha evaluado comparando el prototipo desarrollado con un prototipo previo ya existente en el grupo de investigación. El resultado es que el nuevo prototipo consigue reconocer en torno a un 84 % de los gestos definidos con ambos brazos, en comparación con el 47 % que el anterior reconocía. El reconocimiento también se realiza de una forma más flexible ya que lo hace independientemente de restricciones como qué brazo se está usando, además de que el sistema es capaz de reconocer automáticamente cuándo el usuario está realizando un gesto y cuándo no.

5.2. Conclusiones personales

Personalmente me siento muy satisfecho con la experiencia obtenida de este trabajo. Poder trabajar en un proyecto propio y a la vez estar rodeado de expertos en la materia ha sido positivo en todos los aspectos porque me ha permitido hacer uso de su consejo cuando lo he necesitado. Además, ha aumentado mi perspectiva sobre la cantidad de conocimiento compartido y creciente de la comunidad científica a una escala global. Una de las razones por las que escogí realizar este trabajo era mi curiosidad por el campo del *deep learning*, y esta decisión ha resultado muy fructífera en ese aspecto.

Reflexionando sobre mi organización del trabajo, pienso que la irregularidad de mi dedicación ha prolongado su duración más de lo estimado inicialmente. Sin embargo, ahora que ya está finalizado creo que los resultados obtenidos son buenos, y pienso que ha sido un buen paso adelante en mi madurez a muchos niveles.

5.3. Problemas encontrados

En el transcurso de este proyecto se han tenido que superar muchos obstáculos. Uno de los primeros fue la toma de contacto con el mundo de la investigación académica: utilizar herramientas para encontrar publicaciones científicas y ser capaz de entender y analizar el contenido de las mismas.

Una vez realizada la investigación, el paso a la parte de desarrollo puso en evidencia la falta de experiencia en el campo del aprendizaje automático más allá de los ejemplos utilizados en clase: la preparación de datasets y el entrenamiento de modelos supone una gran cantidad de decisiones que en algunos casos ha habido que rectificar. Además, los problemas técnicos derivados de la utilización de código ajeno ha sido abundantes: aprender a utilizar varios *frameworks* de *deep learning* (por ejemplo, hubo que reinstalar *Caffe2* tras varios problemas críticos en su uso), modificar la API *Keras* para usar formatos de archivo distintos, y gran parte de reestructuración del código del prototipo previo usado como base del actual.

5.4. Trabajo Futuro

La realización de este trabajo abre posibilidades de aplicación en futuros proyectos. Para mejorar la flexibilidad y precisión de los modelos, la mejor forma de hacerlo sería aumentar el número de datos: ampliar los datasets con más usuarios o en más entornos diferentes haría que los modelos entrenados fuesen más robustos. En una línea similar se podría incluir más gestos diferentes, lo que aumentaría las posibilidades de aplicación del sistema.

Quitando las partes que conciernen la entrada/salida del programa y la comunicación dentro de ROS, el resto del programa (la parte que realiza el procesado de imágenes y reconocimiento de gestos) ha sido reestructurada para conseguir una estructura más modular. Esto permite que en un futuro se puedan usar modelos más nuevos de detectores o clasificadores sin gran esfuerzo de recodificación.

Por limitaciones en recursos, el sistema no ha podido ser probado con un dron real (el uso de ROS da algunas facilidades en este aspecto), por ello esa

tarea es una de las que se pueden realizar. Además, gracias a innovaciones en *hardware* para sistemas embebidos[1], se podría llegar a usar este sistema en drones de manera autónoma (es decir, sin ejecutar los modelos en un ordenador conectado de manera remota al dron), utilizando una GPU integrada en él.

[1] https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/?section=jetsonTX2

Apéndice A

Conceptos básicos sobre CNNs

Este anexo recoge algunos de los conceptos básicos de las redes neuronales convolucionales (CNN).

La rama del *deep learning* se dedica al uso de redes neuronales profundas para encontrar soluciones a problemas relacionados con el aprendizaje automático. Las redes neuronales profundas son un tipo de redes neuronales que tienen, además de la capa de entrada y la de salida, varias capas intermedias (ocultas) con operaciones y conexiones diferentes. Estas capas ocultas, realizan cálculos sobre los datos de entrada en cada capa, hasta producir la salida en la última capa. Matemáticamente, suponen una cadena de funciones ejecutada una detrás de otra, que transforman la entrada numérica en el resultado también numérico, con las dimensiones de la última capa.

Hay tantos tipos de capas distintas con las que formar una red como funciones matemáticas distintas. Sin embargo, algunas de ellas son mucho más populares en ciertas aplicaciones por algunas de sus características como la rapidez de su procesamiento o su capacidad para representar los datos de una manera más compacta.

En materia de visión por computador, existe un tipo de capa cuya aparición ha sido crucial para su avance en los últimos años: la capa de convolución. Esta capa, como su nombre indica, calcula una operación de **convolución** sobre los datos de la capa previa. La convolución utiliza un *kernel* (una matriz de números) que se utiliza para sumar ponderadamente una ventana de números de la entrada con las mismas dimensiones que el *kernel*. Así, la ventana de números se va desplazando por toda la imagen para calcular los valores de salida de la capa de convolución. Las convoluciones son operaciones muy comunes y básicas en el procesamiento de imagen en general (fuera del ámbito de *deep learning*), siendo la base de numerosísimos filtros de imagen, desde cálculos de contornos, hasta filtros de *blur* o *sharp* de imágenes. Por lo tanto, por su cercanía y conveniencia para procesamiento de imágenes, la capa de convolución da nombre a la categoría de **Redes Neuronales Convolucionales** (CNN).

Se ha investigado mucho en los últimos años sobre las mejores arquitecturas para CNNs. Estas arquitecturas se centran en seleccionar qué capas incluir en la

red y el orden de ellas. Algunas de las capas más importantes en la actualidad son:

- ***Fully Connected.*** Esta capa tiene la estructura más evidente ya que su funcionamiento se basa en que cada nodo de esta capa recibe como entrada las salidas de cada uno de los nodos de la capa anterior. Típicamente es usada al final de un bloque o de la red para obtener la salida.

- **Convolución.** Como se ha explicado antes, es una capa muy importante en redes de este tipo. Esta capa contiene filtros, que son distintas convoluciones realizadas sobre la entrada de la capa.

- **Normalización.** Como su nombre indica, este tipo de capas están destinadas a normalizar los datos de entrada, para estabilizar los valores manejados dentro de la red y, por tanto, ayudar a generalizar el entrenamiento. Un ejemplo muy usado es el de *Batch Normalization* o normalización por lotes, que normaliza según la media y la varianza del lote de datos de entrada.

- ***Pooling.*** Este tipo de entrada tiene como objetivo la regularización del modelo. Mediante una reducción de dimensión de los datos de entrada (por ejemplo seleccionar sólo el máximo de cada cinco valores, o quedarse con la mitad de ellos aleatoriamente) se consigue que el modelo aprenda a manejar datos que no sean exactamente iguales a los del entrenamiento. Gracias a este tipo de capas, los modelos son capaces de generalizar mejor y más rápido, haciendo que sean más flexibles a cambios en los datos de entrada.

Apéndice B

Ejemplos de imágenes en los datasets

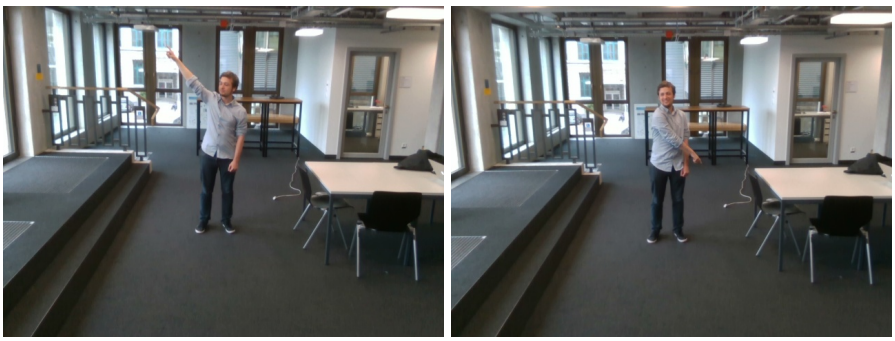
En este anexo se recogen algunos ejemplos representativos del tipo de imágenes contenidas en los datasets utilizados en este trabajo.

B.1. Dataset ETH

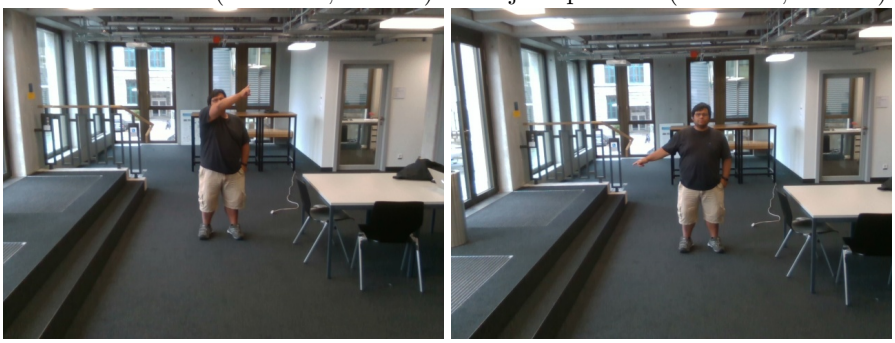
"arriba" (Usuario1, Vídeo1) "abajo" (Usuario1, Vídeo1)

"arriba" (Usuario2, Vídeo2) "derecha" (Usuario2, Vídeo2)

Figura B.1: Ejemplos del dataset ETH.



"arriba-derecha" (Usuario3, Vídeo3) "abajo-izquierda" (Usuario3, Vídeo3)



"arriba-izquierda" (Usuario4, Vídeo4) "abajo-derecha" (Usuario4, Vídeo4)

"izquierda" (Usuario5, Vídeo5) "unknown" (Usuario5, Vídeo5)

Figura B.2: Más ejemplos del dataset ETH.

B.2. Dataset I3A

"arriba" (Usuario1, Vídeo1) "abajo" (Usuario1, Vídeo1)

"arriba" (Usuario2, Vídeo2) "derecha" (Usuario2, Vídeo2)

"arriba-derecha" (Usuario1, Vídeo3) "abajo-izquierda" (Usuario1, Vídeo3)

"arriba-izquierda" (Usuario2, Vídeo4) "abajo-derecha" (Usuario2, Vídeo4)

Figura B.3: Ejemplos del dataset I3A.

"izquierda" (Usuario3, Vídeo5) "unknown" (Usuario3, Vídeo5)

Figura B.4: Más ejemplos del dataset I3A.

Apéndice C

Resultados de los experimentos

En este anexo se recogen los datos completos de los experimentos realizados. Por cada arquitectura se muestra: su nombre, la precisión total de cada uno de los 5 modelos, la precisión y desviación típica de esas precisiones, la matriz de confusión acumulada de los 5 modelos, las precisiones por clase de la acumulación y la **media y desviación típica** de esas precisiones por clase. Estas últimas media y desviación típica son las utilizadas en las tablas resumen 3.2 y 3.3.

C.1. Reconocedor con Esqueleto

C.1.1. Resultados de validación cruzada

```
## Using Euclidean Distance Matrix
# LinearSVC-U
Model1 val accuracy: 100.0 %
Model2 val accuracy: 95.5555555556 %
Model3 val accuracy: 92.3913043478 %
Model4 val accuracy: 88.0341880342 %
Model5 val accuracy: 87.3646209386 %

mean= 92.6691337752 %
std= 4.72801856905 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[180, 0, 0, 0, 0, 13, 0, 0]
[0, 145, 0, 0, 0, 0, 1, 0]
[0, 0, 144, 0, 26, 0, 0, 0]
[0, 0, 0, 151, 0, 1, 0, 20]
[0, 0, 32, 0, 187, 0, 0, 0]
[0, 0, 0, 0, 0, 158, 0, 0]
```

```
[0, 0, 0, 0, 0, 0, 150, 0]
[0, 7, 0, 0, 0, 0, 0, 178]
Accuracy per class:  [100  95  81 100  87  91  99  89]
Mean accuracy:  92.75
Std accuracy:  6.49519052838


# LinearSVC
Model1 val accuracy: 100.0 %
Model2 val accuracy: 95.2777777778 %
Model3 val accuracy: 91.3043478261 %
Model4 val accuracy: 86.7521367521 %
Model5 val accuracy: 88.8086642599 %

mean= 92.4285853232 %
std= 4.73358267458 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[180, 0, 0, 0, 0, 13, 0, 0]
[0, 145, 0, 0, 0, 0, 1, 0]
[0, 0, 148, 0, 30, 0, 0, 0]
[0, 0, 0, 151, 0, 1, 0, 23]
[0, 0, 28, 0, 183, 0, 0, 0]
[0, 0, 0, 0, 0, 158, 0, 0]
[0, 0, 0, 0, 0, 0, 150, 0]
[0, 7, 0, 0, 0, 0, 0, 175]
Accuracy per class:  [100  95  84 100  85  91  99  88]
Mean accuracy:  92.75
Std accuracy:  6.23999198717


# LinearSVC-A
Model1 val accuracy: 100.0 %
Model2 val accuracy: 95.2777777778 %
Model3 val accuracy: 89.8550724638 %
Model4 val accuracy: 97.4358974359 %
Model5 val accuracy: 84.8375451264 %

mean= 93.4812585608 %
std= 5.46303299327 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[360, 0, 0, 0, 4, 4, 0, 0]
[0, 296, 1, 1, 0, 0, 4, 4]
[0, 0, 288, 0, 40, 0, 2, 0]
[0, 0, 0, 288, 0, 40, 0, 2]
```

```
[0, 0, 38, 0, 341, 0, 0, 0]
[0, 0, 0, 38, 0, 341, 0, 0]
[0, 4, 0, 0, 0, 0, 343, 0]
[0, 4, 0, 0, 0, 0, 0, 343]
Accuracy per class:  [100  97  88  88  88  88  98  98]
Mean accuracy:  93.125
Std accuracy:  5.18260311041


# LinearSVC-U with additional 'unknown' class
Model1 val accuracy: 88.2051282051 %
Model2 val accuracy: 73.5182849937 %
Model3 val accuracy: 88.5817307692 %
Model4 val accuracy: 76.8456375839 %
Model5 val accuracy: 84.0836012862 %

mean= 82.2468765676 %
std= 6.07233713568 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[176, 0, 0, 0, 0, 10, 0, 0, 16]
[0, 27, 0, 0, 0, 0, 0, 0, 24]
[0, 0, 71, 0, 6, 0, 0, 0, 12]
[0, 0, 0, 128, 0, 0, 0, 0, 21]
[0, 0, 30, 0, 165, 0, 0, 0, 13]
[0, 0, 0, 0, 0, 152, 0, 0, 9]
[0, 0, 0, 0, 0, 0, 87, 0, 23]
[0, 0, 0, 1, 0, 0, 0, 121, 27]
[4, 125, 75, 22, 42, 10, 64, 77, 1890]
Accuracy per class:  [97 17 40 84 77 88 57 61 92]
Mean accuracy:  68.1111111111
Std accuracy:  25.1857298416


# LinearSVC with additional 'unknown' class
Model1 val accuracy: 88.2051282051 %
Model2 val accuracy: 45.5233291299 %
Model3 val accuracy: 93.1490384615 %
Model4 val accuracy: 76.5100671141 %
Model5 val accuracy: 91.8006430868 %

mean= 79.0376411995 %
std= 17.7519931958 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[177, 0, 0, 0, 0, 9, 0, 0, 25]
```

```
[0, 82, 0, 0, 0, 0, 0, 0, 299]
[0, 0, 130, 0, 15, 0, 0, 0, 23]
[0, 0, 0, 151, 0, 1, 0, 21, 56]
[0, 0, 11, 0, 185, 0, 0, 0, 15]
[0, 0, 0, 0, 0, 160, 0, 0, 13]
[0, 0, 0, 0, 0, 0, 87, 0, 34]
[0, 0, 0, 0, 0, 0, 0, 171, 34]
[3, 70, 35, 0, 13, 2, 64, 6, 1536]
Accuracy per class:  [ 98  53  73 100  86  93  57  86  75]
Mean accuracy:  80.1111111111
Std accuracy:  15.9752895605


# LinearSVC-A with additional 'unknown' class
Model1 val accuracy: 86.8376068376 %
Model2 val accuracy: 46.658259773 %
Model3 val accuracy: 84.375 %
Model4 val accuracy: 81.0402684564 %
Model5 val accuracy: 83.4405144695 %

mean= 76.4703299073 %
std= 15.0212621113 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[356, 0, 0, 0, 2, 2, 0, 0, 50]
[0, 168, 0, 0, 0, 0, 3, 3, 820]
[0, 0, 270, 0, 35, 0, 0, 0, 52]
[0, 0, 0, 270, 0, 35, 0, 0, 52]
[0, 0, 32, 0, 337, 0, 0, 0, 30]
[0, 0, 0, 32, 0, 337, 0, 0, 30]
[0, 0, 1, 0, 0, 0, 278, 0, 83]
[0, 0, 0, 1, 0, 0, 0, 278, 83]
[4, 136, 24, 24, 11, 11, 68, 68, 2870]
Accuracy per class:  [98 55 82 82 87 87 79 79 70]
Mean accuracy:  79.8888888889
Std accuracy:  11.3376897945


# NuSVC   NU=0.4
Model1 val accuracy: 100.0 %
Model2 val accuracy: 79.4444444444 %
Model3 val accuracy: 99.2753623188 %
Model4 val accuracy: 94.8717948718 %
Model5 val accuracy: 98.916967509 %

mean= 94.5017138288 %
std= 7.7378505682 %
```

```
Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[180, 0, 0, 0, 0, 9, 0, 0]
[0, 144, 0, 0, 0, 0, 4, 0]
[0, 0, 172, 0, 62, 0, 0, 0]
[0, 0, 0, 148, 0, 1, 0, 0]
[0, 0, 1, 0, 151, 0, 0, 0]
[0, 0, 0, 2, 0, 162, 0, 0]
[0, 0, 3, 0, 0, 0, 147, 0]
[0, 8, 0, 1, 0, 0, 0, 198]
Accuracy per class:  [100  94  97  98  70  94  97 100]
Mean accuracy:  93.75
Std accuracy:  9.22970747099


# NuSVC-A   NU=0.3
Model1 val accuracy: 100.0 %
Model2 val accuracy: 62.9166666667 %
Model3 val accuracy: 89.3115942029 %
Model4 val accuracy: 97.8632478632 %
Model5 val accuracy: 84.476534296 %

mean= 86.9136086058 %
std= 13.2566294972 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[359, 0, 0, 0, 6, 6, 0, 0]
[0, 294, 1, 1, 0, 0, 10, 10]
[0, 0, 236, 0, 98, 0, 0, 0]
[0, 0, 0, 236, 0, 93, 0, 0]
[1, 0, 37, 0, 278, 0, 0, 0]
[0, 0, 0, 37, 0, 283, 0, 0]
[0, 5, 53, 0, 3, 0, 339, 0]
[0, 5, 0, 53, 0, 3, 0, 339]
Accuracy per class:  [99 96 72 72 72 73 97 97]
Mean accuracy:  84.75
Std accuracy:  12.5274698164


# NuSVC with additional 'unknown' class   NU=0.115
Model1 val accuracy: 88.2051282051 %
Model2 val accuracy: 28.499369483 %
Model3 val accuracy: 95.3125 %
Model4 val accuracy: 78.1879194631 %
Model5 val accuracy: 90.0321543408 %

mean= 76.0474142984 %
```

```
std= 24.4128032668 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[176, 0, 0, 0, 0, 17, 0, 0, 27]
[0, 84, 0, 0, 0, 0, 5, 0, 365]
[0, 0, 89, 0, 0, 0, 0, 0, 15]
[0, 0, 0, 100, 0, 0, 0, 0, 27]
[0, 0, 12, 0, 198, 0, 0, 0, 11]
[0, 0, 0, 0, 0, 116, 0, 0, 13]
[0, 0, 0, 0, 0, 0, 87, 0, 22]
[0, 0, 0, 1, 0, 0, 0, 188, 32]
[4, 68, 75, 50, 15, 39, 59, 10, 1523]
Accuracy per class:  [97 55 50 66 92 67 57 94 74]
Mean accuracy:  72.4444444444
Std accuracy:  16.8991197082


# NuSVC-A with additional 'unknown' class   NU=0.125
Model1 val accuracy: 88.547008547 %
Model2 val accuracy: 29.5081967213 %
Model3 val accuracy: 71.3942307692 %
Model4 val accuracy: 76.677852349 %
Model5 val accuracy: 87.7813504823 %

mean= 70.7817277738 %
std= 21.6482617485 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[344, 0, 0, 0, 2, 12, 0, 0, 54]
[0, 168, 2, 2, 0, 0, 6, 4, 1023]
[0, 0, 210, 0, 69, 0, 0, 0, 29]
[0, 0, 0, 231, 0, 80, 0, 0, 36]
[7, 0, 21, 0, 243, 0, 0, 0, 23]
[0, 0, 0, 3, 0, 222, 0, 0, 21]
[0, 0, 58, 0, 1, 0, 274, 0, 74]
[0, 0, 0, 54, 0, 1, 0, 268, 72]
[9, 136, 36, 37, 70, 70, 69, 77, 2738]
Accuracy per class:  [95 55 64 70 63 57 78 76 67]
Mean accuracy:  69.4444444444
Std accuracy:  11.5960827784


# 1-Dense-A
Model1 val accuracy: 99.79 %
Model2 val accuracy: 82.50 %
Model3 val accuracy: 96.32 %
```

```
Model4 val accuracy: 83.62 %
Model5 val accuracy: 73.35 %

mean= 87.12 %
std= 9.6786 %
- confusion
mean= 87.1011533329 %
std= 9.71492792414 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[360, 0, 0, 0, 83, 75, 0, 0]
[0, 290, 0, 0, 0, 0, 9, 5]
[0, 0, 286, 0, 64, 0, 1, 0]
[0, 0, 0, 283, 0, 36, 0, 0]
[0, 0, 39, 0, 238, 0, 0, 0]
[0, 0, 0, 42, 0, 274, 0, 0]
[0, 7, 2, 0, 0, 0, 339, 0]
[0, 7, 0, 2, 0, 0, 0, 344]
Accuracy per class:  [100  95  87  86  61  71  97  98]
Mean accuracy:  86.875 %
Std accuracy:  13.1666007382 %

# 1-Dense-A with additional 'unknown' class
Model1 val accuracy: 84.93 %
Model2 val accuracy: 33.90 %
Model3 val accuracy: 50.72 %
Model4 val accuracy: 85.81 %
Model5 val accuracy: 79.38 %

mean= 66.84 %
std= 20.8327 %
- confusion
mean= 66.9201738368 %
std= 20.9013330304 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[294, 0, 0, 0, 0, 0, 0, 0, 38]
[0, 169, 0, 0, 0, 0, 1, 0, 1260]
[0, 0, 212, 0, 114, 0, 0, 0, 56]
[0, 0, 0, 239, 0, 106, 0, 0, 66]
[19, 0, 54, 0, 259, 0, 0, 0, 36]
[37, 0, 0, 57, 0, 272, 0, 0, 39]
[0, 0, 49, 0, 0, 0, 316, 0, 117]
[0, 0, 0, 24, 0, 0, 0, 320, 155]
[10, 135, 12, 7, 12, 7, 32, 29, 2303]
Accuracy per class:  [81 55 64 73 67 70 90 91 56]
```

```
Mean accuracy:  71.8888888889 %
Std accuracy:  12.4939491528 %

# 4-Dense-A
Model1 val accuracy: 100.00 %
Model2 val accuracy: 76.81 %
Model3 val accuracy: 84.38 %
Model4 val accuracy: 87.50 %
Model5 val accuracy: 79.78 %

mean= 85.69 %
std= 8.0443 %
- confusion
mean= 85.6869834198 %
std= 8.05452291347 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[355, 0, 0, 0, 25, 19, 0, 0]
[0, 248, 0, 0, 0, 0, 41, 44]
[0, 0, 275, 0, 65, 0, 0, 0]
[0, 0, 0, 275, 0, 65, 0, 0]
[3, 0, 34, 0, 295, 0, 0, 0]
[2, 0, 0, 34, 0, 301, 0, 0]
[0, 25, 18, 0, 0, 0, 308, 0]
[0, 31, 0, 18, 0, 0, 0, 305]
Accuracy per class:  [98 81 84 84 76 78 88 87]
Mean accuracy:  84.5 %
Std accuracy:  6.40312423743 %

# 4-Dense-A with additional 'unknown' class
Model1 val accuracy: 86.13 %
Model2 val accuracy: 34.34 %
Model3 val accuracy: 43.81 %
Model4 val accuracy: 77.36 %
Model5 val accuracy: 84.25 %

mean= 65.18 %
std= 21.7196 %
- confusion
mean= 65.1882654615 %
std= 21.7426990675 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[294, 0, 0, 0, 0, 0, 0, 0, 29]
[0, 212, 0, 0, 0, 0, 22, 26, 1249]
[0, 0, 229, 0, 119, 0, 0, 0, 48]
```

```
[0, 0, 0, 189, 0, 84, 0, 0, 48]
[39, 0, 27, 0, 244, 0, 0, 0, 30]
[10, 0, 0, 24, 0, 237, 0, 0, 27]
[0, 0, 21, 0, 0, 0, 298, 0, 130]
[0, 0, 0, 65, 0, 1, 0, 287, 247]
[17, 92, 50, 49, 22, 63, 29, 36, 2262]
Accuracy per class:  [81 69 70 57 63 61 85 82 55]
Mean accuracy:  69.2222222222 %
Std accuracy:  10.5911679954 %

# 2-ConvNN-A
Model1 val accuracy: 100.00 %
Model2 val accuracy: 95.00 %
Model3 val accuracy: 84.01 %
Model4 val accuracy: 82.76 %
Model5 val accuracy: 97.43 %

mean= 91.84 %
std= 7.0933 %
- confusion
mean= 91.8873756216 %
std= 7.05218333886 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[358, 0, 0, 0, 0, 0, 0, 0]
[0, 232, 0, 0, 0, 0, 18, 40]
[0, 0, 324, 0, 25, 0, 0, 0]
[0, 0, 0, 315, 0, 38, 0, 8]
[0, 0, 1, 0, 360, 0, 0, 0]
[2, 0, 0, 7, 0, 347, 0, 0]
[0, 35, 2, 0, 0, 0, 331, 0]
[0, 37, 0, 5, 0, 0, 0, 301]
Accuracy per class:  [99 76 99 96 93 90 94 86]
Mean accuracy:  91.625 %
Std accuracy:  7.19266119041 %

# 2-ConvNN-A with additional 'unknown' class
Model1 val accuracy: 83.99 %
Model2 val accuracy: 44.13 %
Model3 val accuracy: 82.21 %
Model4 val accuracy: 83.02 %
Model5 val accuracy: 90.50 %

mean= 76.77 %
std= 16.5811 %
- confusion
mean= 76.7656502704 %
std= 16.5487132551 %
```

```
Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[354, 0, 0, 0, 0, 0, 0, 0, 58]
[0, 193, 0, 0, 0, 0, 0, 0, 862]
[0, 0, 315, 0, 35, 0, 0, 0, 75]
[0, 0, 0, 295, 0, 41, 0, 0, 62]
[0, 0, 0, 0, 340, 0, 0, 0, 33]
[0, 0, 0, 8, 0, 339, 0, 0, 35]
[0, 1, 0, 0, 0, 0, 282, 0, 91]
[0, 3, 0, 0, 0, 0, 0, 285, 91]
[6, 107, 12, 24, 10, 5, 67, 64, 2763]
Accuracy per class:  [98 63 96 90 88 88 80 81 67]
Mean accuracy:  83.4444444444 %
Std accuracy:  11.3540116367 %

# 6-ConvNN-A
Model1 val accuracy: 100.00 %
Model2 val accuracy: 96.53 %
Model3 val accuracy: 100.00 %
Model4 val accuracy: 97.41 %
Model5 val accuracy: 87.68 %

mean= 96.32 %
std= 4.5380 %
- confusion
mean= 96.3739624796 %
std= 4.45312758476 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[354, 0, 0, 0, 0, 1, 0, 0]
[0, 277, 0, 0, 0, 0, 5, 12]
[0, 0, 322, 0, 7, 0, 0, 0]
[0, 0, 0, 293, 0, 7, 0, 0]
[0, 0, 3, 0, 378, 0, 0, 0]
[6, 0, 0, 31, 0, 377, 0, 0]
[0, 23, 2, 0, 0, 0, 344, 0]
[0, 4, 0, 3, 0, 0, 0, 337]
Accuracy per class:  [98 91 98 89 98 97 98 96]
Mean accuracy:  95.625 %
Std accuracy:  3.35177191945 %

# 6-ConvNN-A with additional 'unknown' class
Model1 val accuracy: 89.21 %
Model2 val accuracy: 73.67 %
Model3 val accuracy: 92.61 %
Model4 val accuracy: 81.93 %
```

```
Model5 val accuracy: 84.42 %

mean= 84.37 %
std= 6.5080 %
- confusion
mean= 84.4150772111 %
std= 6.49624828985 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[342, 0, 0, 0, 1, 3, 0, 0, 48]
[0, 149, 0, 0, 0, 0, 3, 0, 72]
[0, 8, 237, 0, 0, 0, 0, 0, 40]
[0, 0, 0, 218, 0, 2, 0, 0, 55]
[9, 0, 18, 0, 342, 0, 0, 0, 44]
[2, 0, 0, 28, 0, 334, 0, 0, 44]
[0, 13, 0, 0, 0, 0, 292, 0, 84]
[0, 0, 0, 0, 0, 0, 0, 293, 107]
[7, 134, 72, 81, 42, 46, 54, 56, 3576]
Accuracy per class:  [95 49 72 66 88 86 83 83 87]
Mean accuracy:  78.7777777778 %
Std accuracy:  13.3314813529 %

27-ConvNN-A
Model1 val accuracy: 100.00 %
Model2 val accuracy: 73.19 %
Model3 val accuracy: 93.57 %
Model4 val accuracy: 78.23 %
Model5 val accuracy: 97.98 %

mean= 84.49 %
std= 10.7561 %
- confusion
mean= 88.6146866745 %
std= 10.8588475029 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[359, 0, 0, 0, 17, 38, 0, 0]
[0, 252, 2, 1, 0, 0, 108, 12]
[0, 0, 323, 0, 4, 0, 23, 0]
[0, 0, 0, 271, 0, 2, 0, 23]
[1, 0, 0, 0, 364, 1, 0, 0]
[0, 0, 0, 11, 0, 344, 0, 0]
[0, 20, 2, 0, 0, 0, 218, 0]
[0, 32, 0, 44, 0, 0, 0, 314]
Accuracy per class:  [99 82 98 82 94 89 62 89]
Mean accuracy:  86.875 %
```

```
Std accuracy:  11.1628569372 %

27-ConvNN-A with additional 'unknown' class
Model1 val accuracy: 78.17 %
Model2 val accuracy: 33.40 %
Model3 val accuracy: 58.05 %
Model4 val accuracy: 71.20 %
Model5 val accuracy: 91.72 %

mean= 66.51 %
std= 19.8058 %
- confusion
mean= 66.5189158783 %
std= 19.7905012818 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[354, 0, 0, 0, 1, 5, 0, 0, 48]
[0, 200, 10, 0, 0, 0, 57, 30, 1460]
[0, 0, 212, 0, 2, 0, 8, 0, 73]
[0, 0, 0, 250, 0, 9, 0, 7, 76]
[0, 0, 39, 0, 374, 0, 0, 0, 40]
[0, 0, 0, 8, 0, 358, 0, 0, 27]
[0, 10, 53, 0, 1, 0, 277, 0, 127]
[0, 9, 0, 59, 0, 1, 0, 312, 155]
[6, 85, 13, 10, 7, 12, 7, 0, 2064]
Accuracy per class:  [98 65 64 76 97 92 79 89 50]
Mean accuracy:  78.8888888889 %
Std accuracy:  15.7229092901 %


## Using Euclidean Distance Matrix resized to 128x128
# MobileNet-A
Model1 val accuracy: 100.00 %
Model2 val accuracy: 98.61 %
Model3 val accuracy: 100.00 %
Model4 val accuracy: 75.43 %
Model5 val accuracy: 82.54 %

mean= 91.32 %
std= 10.33 %
- confusion
mean= 91.3847233793 %
std= 10.2415944881 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[360, 0, 0, 0, 0, 0, 0, 0]
```

```
[0, 211, 0, 0, 0, 0, 0, 9]
[0, 22, 276, 0, 4, 0, 1, 14]
[0, 0, 0, 298, 0, 4, 0, 1]
[0, 1, 49, 3, 381, 2, 0, 12]
[0, 0, 0, 26, 0, 379, 0, 0]
[0, 2, 2, 0, 0, 0, 348, 0]
[0, 68, 0, 0, 0, 0, 0, 313]
Accuracy per class:  [100  69  84  91  98  98  99  89]
Mean accuracy:  91.0 %
Std accuracy:  9.87420882907 %

# MobileNet-A with additional 'unknown' class
Model1 val accuracy: 85.10 %
Model2 val accuracy: 75.19 %
Model3 val accuracy: 92.31 %
Model4 val accuracy: 75.84 %
Model5 val accuracy: 81.41 %

mean= 81.97 %
std= 6.3336 %
- confusion
mean= 81.9234158488 %
std= 6.33698759419 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[350, 0, 0, 0, 0, 0, 0, 0, 22]
[0, 103, 0, 0, 0, 0, 0, 0, 27]
[0, 1, 179, 0, 0, 0, 0, 0, 37]
[0, 0, 0, 186, 0, 0, 0, 1, 36]
[2, 0, 40, 0, 308, 0, 0, 0, 33]
[0, 0, 0, 9, 0, 339, 0, 0, 33]
[0, 0, 0, 0, 0, 0, 215, 0, 44]
[0, 0, 0, 0, 0, 0, 0, 178, 58]
[8, 200, 108, 132, 77, 46, 134, 170, 3780]
Accuracy per class:  [97 33 54 56 80 88 61 51 92]
Mean accuracy:  68.0 %
Std accuracy:  20.7203603572 %
```

C.1.2. Resultados de test

```
## Using Euclidean Distance Matrix
# LinearSVC-U
Model1 val accuracy: 76.2764632628 %
Model2 val accuracy: 71.2640099626 %
Model3 val accuracy: 69.7384806974 %
Model4 val accuracy: 70.8592777086 %
Model5 val accuracy: 69.1158156912 %
```

```
mean= 71.4508094645 %
std= 2.53203492002 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[1704, 0, 0, 29, 200, 288, 0, 0]
[0, 1578, 0, 15, 0, 9, 35, 44]
[0, 0, 1046, 0, 258, 0, 0, 0]
[0, 0, 4, 802, 0, 16, 0, 220]
[36, 0, 91, 1, 914, 12, 0, 0]
[21, 0, 142, 890, 414, 1197, 0, 0]
[0, 1, 318, 5, 295, 0, 2073, 0]
[44, 456, 119, 278, 99, 203, 42, 2161]
Accuracy per class:  [94 77 60 39 41 69 96 89]
Mean accuracy:  70.625
Std accuracy:  21.1005775987


# LinearSVC
Model1 val accuracy: 76.4632627646 %
Model2 val accuracy: 71.3574097136 %
Model3 val accuracy: 71.4196762142 %
Model4 val accuracy: 71.4508094645 %
Model5 val accuracy: 68.5242839352 %

mean= 71.8430884184 %
std= 2.56630156699 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[1708, 0, 0, 31, 207, 282, 0, 0]
[0, 1611, 0, 15, 0, 9, 34, 45]
[0, 0, 1050, 0, 267, 0, 0, 0]
[0, 0, 5, 916, 0, 19, 0, 289]
[31, 0, 84, 1, 899, 12, 0, 0]
[22, 0, 144, 778, 417, 1190, 0, 0]
[0, 1, 321, 5, 293, 0, 2073, 0]
[44, 423, 116, 274, 97, 213, 43, 2091]
Accuracy per class:  [94 79 61 45 41 68 96 86]
Mean accuracy:  71.25
Std accuracy:  19.7721394897


# LinearSVC-A
Model1 val accuracy: 62.9202988792 %
Model2 val accuracy: 57.4097135741 %
Model3 val accuracy: 55.0747198007 %
Model4 val accuracy: 60.6475716065 %
```

```
Model5 val accuracy: 59.8069738481 %

mean= 59.1718555417 %
std= 2.70272478122 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[3240, 4, 10, 10, 749, 749, 0, 0]
[0, 2344, 17, 17, 0, 0, 113, 113]
[0, 0, 1275, 40, 193, 9, 483, 8]
[0, 0, 40, 1275, 9, 193, 8, 483]
[164, 3, 778, 22, 1936, 562, 0, 46]
[164, 3, 22, 778, 562, 1936, 46, 0]
[21, 858, 1355, 243, 275, 181, 3500, 425]
[21, 858, 243, 1355, 181, 275, 425, 3500]
Accuracy per class:  [89 57 34 34 49 49 76 76]
Mean accuracy:  58.0
Std accuracy:  19.1180542943


# LinearSVC-U with additional 'unknown' class
Model1 val accuracy: 71.8567389255 %
Model2 val accuracy: 63.7700282752 %
Model3 val accuracy: 66.9557021678 %
Model4 val accuracy: 72.1206409048 %
Model5 val accuracy: 68.8407163054 %

mean= 68.7087653157 %
std= 3.13152991055 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[1636, 0, 0, 1, 190, 190, 0, 0, 5]
[0, 558, 0, 2, 46, 12, 75, 24, 574]
[0, 0, 974, 0, 272, 0, 0, 0, 97]
[0, 0, 0, 504, 0, 4, 0, 105, 2]
[17, 0, 71, 1, 763, 4, 0, 0, 34]
[26, 0, 0, 646, 320, 1087, 0, 0, 1]
[7, 31, 395, 15, 470, 2, 2038, 13, 460]
[50, 359, 12, 324, 108, 332, 9, 1929, 556]
[69, 1087, 268, 527, 11, 94, 28, 354, 8736]
Accuracy per class:  [90 27 56 24 35 63 94 79 83]
Mean accuracy:  61.2222222222
Std accuracy:  25.7499850174


# LinearSVC with additional 'unknown' class
Model1 val accuracy: 60.5278039585 %
```

```
Model2 val accuracy: 55.438265787 %
Model3 val accuracy: 59.8303487276 %
Model4 val accuracy: 69.7455230914 %
Model5 val accuracy: 59.1140433553 %

mean= 60.931196984 %
std= 4.74428662688 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[1643, 0, 0, 25, 235, 166, 0, 0, 14]
[0, 856, 0, 9, 97, 14, 75, 35, 1765]
[0, 0, 1025, 0, 277, 0, 0, 0, 135]
[0, 0, 17, 811, 0, 12, 0, 314, 16]
[19, 0, 51, 7, 759, 3, 0, 0, 51]
[49, 0, 138, 585, 367, 1143, 0, 0, 8]
[0, 18, 377, 10, 340, 0, 2055, 8, 578]
[51, 389, 49, 380, 103, 332, 13, 2043, 2071]
[43, 772, 63, 193, 2, 55, 7, 25, 5827]
Accuracy per class:  [91 42 59 40 34 66 95 84 55]
Mean accuracy:  62.8888888889
Std accuracy:  21.4706230539


# LinearSVC-A with additional 'unknown' class
Model1 val accuracy: 56.4184731385 %
Model2 val accuracy: 49.1423185674 %
Model3 val accuracy: 44.0716305372 %
Model4 val accuracy: 61.6776625825 %
Model5 val accuracy: 45.7681432611 %

mean= 51.4156456173 %
std= 6.65333197182 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[3332, 0, 0, 0, 519, 519, 1, 1, 50]
[0, 1936, 70, 70, 56, 56, 152, 152, 5216]
[0, 0, 959, 39, 164, 3, 415, 3, 71]
[0, 0, 39, 959, 3, 164, 3, 415, 71]
[58, 0, 441, 1, 1768, 579, 0, 40, 83]
[58, 0, 1, 441, 579, 1768, 40, 0, 83]
[52, 205, 1896, 269, 555, 217, 3087, 423, 2488]
[52, 205, 269, 1896, 217, 555, 423, 3087, 2488]
[58, 1724, 65, 65, 44, 44, 454, 454, 10380]
Accuracy per class:  [92 47 25 25 45 45 67 67 49]
Mean accuracy:  51.3333333333
Std accuracy:  20.0997512422
```

```
# NuSVC   NU=0.4
Model1 val accuracy: 65.0684931507 %
Model2 val accuracy: 58.2191780822 %
Model3 val accuracy: 68.9601494396 %
Model4 val accuracy: 65.4420921544 %
Model5 val accuracy: 68.0572851806 %

mean= 65.1494396015 %
std= 3.77088313138 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[1724, 0, 0, 10, 461, 324, 0, 0]
[0, 1016, 0, 33, 4, 4, 235, 172]
[0, 0, 1242, 0, 457, 4, 0, 0]
[40, 0, 228, 1053, 48, 26, 0, 842]
[30, 0, 58, 0, 814, 4, 0, 0]
[11, 0, 0, 832, 317, 1361, 0, 0]
[0, 4, 140, 2, 25, 2, 1842, 0]
[0, 1015, 52, 90, 54, 0, 73, 1411]
Accuracy per class:  [95 49 72 52 37 78 85 58]
Mean accuracy:  65.75
Std accuracy:  18.6128315954


# NuSVC-A   NU=0.3
Model1 val accuracy: 54.9190535492 %
Model2 val accuracy: 37.4844333748 %
Model3 val accuracy: 53.6892901619 %
Model4 val accuracy: 64.0877957659 %
Model5 val accuracy: 55.9775840598 %

mean= 53.2316313823 %
std= 8.67672347657 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[3216, 0, 17, 10, 1152, 1143, 3, 3]
[0, 1826, 204, 195, 10, 10, 1063, 1083]
[0, 118, 1898, 163, 451, 32, 970, 72]
[0, 108, 176, 1949, 30, 457, 77, 956]
[199, 9, 1095, 66, 1781, 435, 19, 56]
[195, 9, 42, 1042, 397, 1746, 50, 22]
[0, 1017, 212, 93, 26, 60, 2352, 53]
[0, 983, 96, 222, 58, 22, 41, 2330]
Accuracy per class:  [89 44 50 52 45 44 51 50]
```

```
Mean accuracy:  53.125
Std accuracy:  13.8963799243


# NuSVC with additional 'unknown' class   NU=0.115
Model1 val accuracy: 57.8133836004 %
Model2 val accuracy: 43.5815268615 %
Model3 val accuracy: 52.4033930254 %
Model4 val accuracy: 53.8548539114 %
Model5 val accuracy: 47.6908576814 %

mean= 51.068803016 %
std= 4.94979587947 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[1666, 0, 0, 0, 655, 356, 0, 0, 12]
[0, 921, 0, 15, 11, 8, 160, 88, 2706]
[0, 0, 710, 0, 360, 1, 0, 0, 83]
[40, 0, 0, 977, 3, 60, 0, 339, 54]
[19, 0, 97, 0, 643, 0, 0, 0, 46]
[21, 0, 0, 317, 216, 1217, 0, 0, 2]
[0, 0, 112, 4, 15, 0, 1796, 0, 357]
[0, 724, 10, 31, 69, 3, 64, 1356, 2945]
[59, 390, 791, 676, 208, 80, 130, 642, 4260]
Accuracy per class:  [92 45 41 48 29 70 83 55 40]
Mean accuracy:  55.8888888889
Std accuracy:  20.0468586868


# NuSVC-A with additional 'unknown' class   NU=0.125
Model1 val accuracy: 62.629594722 %
Model2 val accuracy: 48.9349670123 %
Model3 val accuracy: 44.4297832234 %
Model4 val accuracy: 63.7511781338 %
Model5 val accuracy: 48.1244109331 %

mean= 53.5739868049 %
std= 8.00517456465 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[3237, 0, 10, 2, 873, 975, 1, 1, 41]
[0, 1311, 156, 106, 6, 10, 759, 762, 5881]
[1, 61, 1846, 125, 592, 82, 188, 46, 114]
[0, 62, 163, 1933, 81, 562, 62, 193, 122]
[110, 1, 716, 102, 1644, 477, 0, 56, 63]
[103, 3, 47, 626, 497, 1585, 42, 0, 60]
```

```
[0, 335, 268, 127, 23, 71, 2630, 176, 1782]
[0, 264, 117, 299, 79, 28, 192, 2618, 1250]
[159, 2033, 417, 420, 110, 115, 701, 723, 11617]
Accuracy per class:  [89 32 49 51 42 40 57 57 55]
Mean accuracy:  52.4444444444
Std accuracy:  15.2031510704


# 1-Dense-A
Model1 val accuracy: 61.9863013699 %
Model2 val accuracy: 64.3368617684 %
Model3 val accuracy: 64.897260274 %
Model4 val accuracy: 61.2391033624 %
Model5 val accuracy: 60.7098381071 %

mean= 62.6338729763 %
std= 1.67867413898 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[3421, 0, 282, 298, 1900, 1897, 19, 24]
[0, 2970, 20, 20, 0, 0, 107, 95]
[0, 120, 1951, 274, 117, 15, 102, 73]
[0, 116, 274, 1867, 16, 131, 70, 97]
[94, 13, 1050, 40, 1296, 553, 63, 57]
[95, 15, 35, 1089, 574, 1309, 57, 68]
[0, 412, 103, 35, 0, 0, 3649, 506]
[0, 424, 25, 117, 2, 0, 508, 3655]
Accuracy per class:  [94 72 52 49 33 33 79 79]
Mean accuracy:  61.375 %
Std accuracy:  21.3596436066 %

# 1-Dense-A with additional 'unknown' class
Model1 val accuracy: 55.6361922714 %
Model2 val accuracy: 46.8614514609 %
Model3 val accuracy: 39.1423185674 %
Model4 val accuracy: 61.6211121583 %
Model5 val accuracy: 50.2639019793 %

mean= 50.7049952875 %
std= 7.64383668873 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[3183, 0, 1, 4, 590, 597, 5, 3, 42]
[41, 1739, 72, 61, 140, 161, 289, 259, 6797]
[1, 27, 1661, 271, 596, 76, 200, 4, 98]
[4, 30, 271, 1772, 83, 609, 4, 230, 114]
```

```
[89, 0, 922, 20, 1655, 374, 13, 30, 115]
[145, 0, 24, 940, 448, 1726, 30, 14, 133]
[17, 359, 467, 6, 70, 16, 2975, 345, 1807]
[20, 426, 5, 450, 22, 96, 379, 3029, 2665]
[110, 1489, 317, 216, 301, 250, 680, 661, 9159]
Accuracy per class:  [88 42 44 47 42 44 65 66 43]
Mean accuracy:  53.4444444444 %
Std accuracy:  15.1885271842 %

# 4-Dense-A
Model1 val accuracy: 73.2098381071 %
Model2 val accuracy: 67.9327521793 %
Model3 val accuracy: 70.9059775841 %
Model4 val accuracy: 71.8711083437 %
Model5 val accuracy: 68.9601494396 %

mean= 70.5759651308 %
std= 1.91421138181 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[3327, 0, 57, 54, 882, 879, 14, 13]
[0, 3074, 20, 20, 0, 0, 93, 94]
[2, 86, 2496, 273, 509, 79, 187, 89]
[1, 88, 274, 2496, 59, 466, 87, 188]
[136, 0, 731, 20, 2008, 444, 2, 22]
[144, 0, 20, 735, 447, 2037, 21, 1]
[0, 383, 140, 1, 0, 0, 3608, 545]
[0, 439, 2, 141, 0, 0, 563, 3623]
Accuracy per class:  [92 75 66 66 51 52 78 79]
Mean accuracy:  69.875 %
Std accuracy:  13.0904306652 %

# 4-Dense-A with additional 'unknown' class
Model1 val accuracy: 53.807728558 %
Model2 val accuracy: 42.9783223374 %
Model3 val accuracy: 37.3986804901 %
Model4 val accuracy: 50.1413760603 %
Model5 val accuracy: 45.7021677663 %

mean= 46.0056550424 %
std= 5.68002461716 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[3158, 0, 6, 17, 595, 641, 3, 2, 70]
[58, 1966, 100, 222, 294, 253, 367, 423, 7984]
[0, 2, 1337, 279, 578, 71, 217, 4, 106]
```

```
[1, 5, 290, 1323, 60, 575, 4, 109, 87]
[130, 0, 746, 0, 1409, 442, 6, 30, 102]
[92, 0, 0, 717, 437, 1398, 29, 12, 96]
[17, 200, 851, 22, 114, 47, 2948, 343, 1878]
[21, 401, 12, 705, 70, 177, 324, 2639, 2379]
[133, 1496, 398, 455, 348, 301, 677, 1013, 8228]
Accuracy per class:  [87 48 35 35 36 35 64 57 39]
Mean accuracy:  48.4444444444 %
Std accuracy:  16.9581911451 %

# 2-ConvNN-A
Model1 val accuracy: 73.1787048568 %
Model2 val accuracy: 75.6693648817 %
Model3 val accuracy: 70.2677459527 %
Model4 val accuracy: 67.99501868 %
Model5 val accuracy: 76.9146948941 %

mean= 72.8051058531 %
std= 3.31081295039 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[3501, 0, 0, 0, 418, 396, 10, 0]
[0, 2479, 19, 18, 0, 0, 340, 125]
[0, 87, 2679, 295, 233, 17, 247, 97]
[0, 66, 306, 2600, 7, 293, 76, 372]
[68, 88, 685, 0, 2388, 652, 0, 45]
[41, 92, 0, 762, 859, 2546, 24, 7]
[0, 886, 49, 2, 0, 0, 3684, 421]
[0, 372, 2, 63, 0, 1, 194, 3508]
Accuracy per class:  [96 60 71 69 61 65 80 76]
Mean accuracy:  72.25 %
Std accuracy:  11.0651479882 %

# 2-ConvNN-A with additional 'unknown' class
Model1 val accuracy: 72.2808671065 %
Model2 val accuracy: 69.9245994345 %
Model3 val accuracy: 55.5984919887 %
Model4 val accuracy: 58.8595664467 %
Model5 val accuracy: 66.2582469369 %

mean= 64.5843543827 %
std= 6.38866748985 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[3375, 1, 6, 20, 302, 245, 0, 0, 39]
[0, 1836, 16, 15, 0, 0, 246, 89, 5144]
```

```
[0, 45, 2406, 250, 297, 31, 85, 29, 133]
[0, 15, 244, 2256, 74, 363, 12, 117, 186]
[35, 61, 648, 0, 2158, 553, 0, 30, 63]
[60, 54, 2, 892, 949, 2565, 28, 0, 86]
[0, 296, 54, 5, 0, 0, 3572, 345, 1813]
[0, 280, 4, 34, 0, 0, 311, 3567, 939]
[140, 1482, 360, 268, 125, 148, 321, 398, 12527]
Accuracy per class:  [93 45 64 60 55 65 78 77 59]
Mean accuracy:  66.2222222222 %
Std accuracy:  13.5218159347 %

# 6-ConvNN-A
Model1 val accuracy: 72.9140722291 %
Model2 val accuracy: 62.7801992528 %
Model3 val accuracy: 63.8075965131 %
Model4 val accuracy: 70.5479452055 %
Model5 val accuracy: 63.8387297634 %

mean= 66.7777085928 %
std= 4.13059041887 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[3251, 31, 56, 28, 911, 972, 10, 11]
[0, 3160, 32, 1, 2, 4, 453, 147]
[1, 158, 2336, 272, 61, 79, 82, 55]
[29, 37, 226, 2142, 447, 699, 20, 513]
[96, 144, 755, 8, 1905, 481, 2, 39]
[233, 27, 119, 1152, 506, 1593, 22, 40]
[0, 460, 195, 30, 1, 0, 3709, 417]
[0, 53, 21, 107, 72, 77, 277, 3353]
Accuracy per class:  [90 77 62 57 48 40 81 73]
Mean accuracy:  66.0 %
Std accuracy:  16.0779351908 %

# 6-ConvNN-A with additional 'unknown' class
Model1 val accuracy: 67.7851083883 %
Model2 val accuracy: 65.0424128181 %
Model3 val accuracy: 60.5560791706 %
Model4 val accuracy: 68.2375117813 %
Model5 val accuracy: 64.5805843544 %

mean= 65.2403393025 %
std= 2.75229493464 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[3008, 1, 109, 10, 675, 654, 0, 0, 51]
```

```
[0, 989, 14, 4, 0, 49, 352, 179, 1321]
[0, 166, 2080, 197, 105, 10, 67, 137, 87]
[3, 39, 93, 1807, 340, 698, 4, 246, 152]
[57, 102, 792, 28, 2038, 473, 0, 31, 84]
[267, 27, 192, 1140, 482, 1608, 14, 17, 85]
[0, 237, 73, 4, 0, 0, 2748, 173, 741]
[0, 4, 12, 95, 46, 80, 116, 2709, 786]
[275, 2505, 375, 455, 219, 333, 1274, 1083, 17623]
Accuracy per class:  [83 24 55 48 52 41 60 59 84]
Mean accuracy:  56.2222222222 %
Std accuracy:  17.8498538916 %

27-ConvNN-A
Model1 val accuracy: 49.5797011208 %
Model2 val accuracy: 43.3219178082 %
Model3 val accuracy: 47.3381070984 %
Model4 val accuracy: 48.6924034869 %
Model5 val accuracy: 56.6780821918 %

mean= 49.1220423412 %
std= 4.34307837544 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[3461, 184, 159, 328, 834, 1014, 0, 1]
[12, 1570, 120, 90, 25, 123, 482, 519]
[7, 188, 889, 164, 96, 69, 1281, 165]
[0, 22, 443, 1122, 7, 193, 180, 755]
[120, 29, 1894, 194, 1811, 645, 0, 38]
[10, 23, 160, 1630, 1132, 1833, 38, 3]
[0, 1593, 9, 3, 0, 0, 2095, 97]
[0, 461, 66, 209, 0, 28, 499, 2997]
Accuracy per class:  [95 38 23 30 46 46 45 65]
Mean accuracy:  48.5 %
Std accuracy:  21.100947846 %

27-ConvNN-A with additional 'unknown' class
Model1 val accuracy: 51.9792648445 %
Model2 val accuracy: 40.0377002828 %
Model3 val accuracy: 47.9547596607 %
Model4 val accuracy: 51.6399622997 %
Model5 val accuracy: 45.6644674835 %

mean= 47.4552309142 %
std= 4.39215170669 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
```

```
[2966, 137, 118, 301, 1004, 1123, 17, 16, 256]
[25, 1185, 68, 42, 14, 16, 305, 388, 2832]
[0, 7, 892, 79, 70, 2, 632, 82, 153]
[0, 2, 368, 1014, 32, 112, 193, 443, 139]
[53, 7, 1136, 181, 1305, 269, 8, 39, 138]
[24, 8, 121, 914, 913, 1705, 34, 10, 120]
[0, 28, 193, 59, 76, 1, 1582, 28, 556]
[26, 542, 97, 497, 66, 189, 326, 1929, 4139]
[516, 2154, 747, 653, 425, 488, 1478, 1640, 12597]
Accuracy per class:  [82 29 23 27 33 43 34 42 60]
Mean accuracy:  41.4444444444 %
Std accuracy:  17.7081971672 %


## Using Euclidean Distance Matrix resized to 128x128
# MobileNet-A
Model1 val accuracy: 74.9533001245 %
Model2 val accuracy: 73.7391033624 %
Model3 val accuracy: 70.6724782067 %
Model4 val accuracy: 71.8555417186 %
Model5 val accuracy: 66.1581569116 %

mean= 71.4757160648 %
std= 3.04244661032 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[3401, 12, 62, 40, 249, 156, 0, 0]
[0, 3207, 12, 7, 8, 35, 24, 34]
[6, 221, 2465, 141, 352, 38, 30, 72]
[11, 64, 168, 2128, 159, 792, 36, 178]
[107, 190, 458, 54, 2276, 374, 15, 192]
[81, 23, 106, 520, 490, 2011, 41, 10]
[4, 22, 405, 222, 196, 180, 3800, 419]
[0, 331, 64, 628, 175, 319, 629, 3670]
Accuracy per class:  [94 78 65 56 58 51 83 80]
Mean accuracy:  70.625 %
Std accuracy:  14.2823098622 %

# MobileNet-A with additional 'unknown' class
Model1 val accuracy: 56.7577756833 %
Model2 val accuracy: 57.521206409 %
Model3 val accuracy: 61.8944392083 %
Model4 val accuracy: 59.0197926484 %
Model5 val accuracy: 55.7964184731 %

mean= 58.1979264844 %
std= 2.12785786311 %
```

```
Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[2792, 1, 1, 0, 279, 80, 0, 0, 0]
[0, 420, 0, 0, 0, 0, 0, 0, 5]
[0, 14, 1068, 0, 101, 0, 4, 0, 0]
[0, 5, 0, 936, 0, 281, 0, 0, 1]
[33, 35, 120, 0, 1278, 16, 0, 16, 0]
[18, 51, 25, 103, 65, 853, 1, 2, 1]
[0, 0, 11, 0, 1, 1, 1485, 26, 4]
[0, 1, 0, 58, 0, 2, 25, 1131, 8]
[767, 3543, 2515, 2643, 2181, 2672, 3060, 3400, 20911]
Accuracy per class:  [77 10 28 25 32 21 32 24 99]
Mean accuracy:  38.6666666667 %
Std accuracy:  27.5842144874 %
```

C.2. Reconocedor con Segmentación

C.2.1. Resultados de validación cruzada

```
## Using images without background
# MobileNet
Model1 val accuracy: 90.00 %
Model2 val accuracy: 82.10 %
Model3 val accuracy: 88.97 %
Model4 val accuracy: 88.84 %
Model5 val accuracy: 89.34 %

mean= 87.85 %
std= 2.9031 %
- confusion
mean= 88.2557825751 %
std= 2.22869912334 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[171, 1, 2, 0, 1, 0, 0, 0]
[1, 124, 0, 0, 0, 0, 0, 0]
[0, 0, 121, 0, 30, 0, 5, 0]
[0, 2, 0, 151, 0, 1, 0, 2]
[2, 18, 47, 0, 181, 0, 32, 0]
[6, 0, 0, 0, 0, 171, 4, 0]
[0, 4, 6, 0, 0, 0, 110, 0]
[0, 3, 0, 0, 1, 0, 0, 196]
Accuracy per class:  [ 95  81  68 100  84  99  72  98]
Mean accuracy:  87.125 %
Std accuracy:  11.8789467126 %
```

```
# MobileNet-A
Model1 val accuracy: 91.25 %
Model2 val accuracy: 77.78 %
Model3 val accuracy: 95.04 %
Model4 val accuracy: 89.01 %
Model5 val accuracy: 90.99 %

mean= 88.81 %
std= 5.8518 %
- confusion
mean= 89.1317157001 %
std= 5.42918741481 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[340, 1, 0, 0, 0, 1, 0, 4]
[1, 269, 0, 0, 0, 0, 12, 2]
[0, 3, 304, 26, 15, 3, 11, 6]
[0, 0, 0, 250, 2, 20, 0, 9]
[8, 8, 20, 2, 364, 1, 30, 33]
[10, 3, 0, 25, 1, 358, 3, 7]
[0, 11, 0, 0, 2, 2, 291, 0]
[1, 9, 3, 24, 1, 0, 2, 288]
Accuracy per class:  [94 88 92 76 94 92 83 82]
Mean accuracy:  87.625 %
Std accuracy:  6.20357759684 %

# MobileNet-U with additional 'unknown' class
Model1 val accuracy: 84.03 %
Model2 val accuracy: 72.83 %
Model3 val accuracy: 88.34 %
Model4 val accuracy: 68.75 %
Model5 val accuracy: 82.07 %

mean= 79.20 %
std= 7.2785 %
- confusion
mean= 79.8303212981 %
std= 7.31866149661 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[143, 1, 0, 0, 0, 0, 0, 0, 3]
[0, 31, 0, 0, 0, 0, 0, 0, 14]
[0, 0, 59, 0, 0, 0, 10, 0, 10]
[0, 0, 0, 123, 0, 0, 0, 0, 13]
[0, 0, 13, 0, 109, 0, 1, 0, 7]
[0, 0, 0, 0, 0, 145, 2, 0, 2]
```

```
[0, 5, 6, 0, 0, 0, 34, 0, 9]
[0, 0, 0, 0, 0, 0, 1, 147, 23]
[37, 115, 98, 28, 104, 27, 103, 51, 1954]
Accuracy per class:  [79 20 33 81 51 84 22 74 96]
Mean accuracy:  60.0 %
Std accuracy:  27.3333333333 %

# MobileNet with additional 'unknown' class
Model1 val accuracy: 86.28 %
Model2 val accuracy: 71.56 %
Model3 val accuracy: 80.17 %
Model4 val accuracy: 79.22 %
Model5 val accuracy: 84.87 %

mean= 80.42 %
std= 5.1797 %
- confusion
mean= 80.6310024608 %
std= 4.52743803379 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[167, 1, 0, 0, 0, 0, 0, 0, 14]
[0, 33, 0, 0, 0, 0, 1, 0, 25]
[0, 1, 112, 0, 15, 0, 32, 0, 91]
[1, 0, 0, 150, 0, 2, 0, 0, 19]
[0, 2, 2, 0, 106, 0, 0, 0, 17]
[1, 0, 0, 0, 0, 169, 0, 0, 10]
[0, 0, 0, 0, 8, 0, 36, 0, 21]
[0, 0, 0, 0, 0, 0, 2, 177, 39]
[11, 115, 62, 1, 84, 1, 80, 21, 1799]
Accuracy per class:  [92 21 63 99 49 98 23 89 88]
Mean accuracy:  69.1111111111 %
Std accuracy:  29.6027192611 %

# MobileNet-UA with additional 'unknown'
Model1 val accuracy: 82.88 %
Model2 val accuracy: 53.91 %
Model3 val accuracy: 86.54 %
Model4 val accuracy: 66.64 %
Model5 val accuracy: 84.01 %

mean= 74.80 %
std= 12.5771 %
- confusion
mean= 75.1017162739 %
std= 12.7082552196 %

Confusion Matrix
```

```
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[298, 0, 10, 0, 5, 2, 11, 13, 52]
[0, 50, 0, 0, 0, 0, 3, 1, 273]
[0, 0, 173, 1, 0, 0, 12, 0, 32]
[0, 0, 4, 195, 0, 7, 0, 3, 35]
[3, 0, 15, 0, 245, 0, 0, 13, 22]
[3, 0, 0, 6, 1, 267, 0, 1, 14]
[1, 2, 0, 1, 1, 0, 174, 1, 58]
[0, 0, 0, 0, 1, 1, 0, 178, 41]
[55, 252, 125, 124, 132, 108, 149, 139, 3543]
Accuracy per class:  [82 16 52 59 63 69 49 51 87]
Mean accuracy:  58.6666666667 %
Std accuracy:  19.6920739837 %

# MobileNet-A with additional 'unknown' class
Model1 val accuracy: 86.82 %
Model2 val accuracy: 52.27 %
Model3 val accuracy: 87.98 %
Model4 val accuracy: 75.08 %
Model5 val accuracy: 78.08 %

mean= 76.04 %
std= 12.8753 %
- confusion
mean= 75.7891386245 %
std= 12.8576599415 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[312, 7, 1, 0, 2, 2, 0, 0, 33]
[0, 58, 0, 0, 0, 0, 0, 0, 318]
[0, 0, 239, 3, 8, 0, 17, 4, 63]
[0, 0, 3, 223, 1, 19, 4, 7, 80]
[0, 0, 3, 2, 272, 1, 2, 1, 19]
[7, 1, 1, 3, 25, 339, 7, 14, 81]
[0, 0, 0, 0, 1, 0, 185, 0, 53]
[0, 9, 1, 0, 4, 1, 1, 209, 104]
[41, 229, 79, 96, 72, 23, 133, 114, 3319]
Accuracy per class:  [86 19 73 68 70 88 53 59 81]
Mean accuracy:  66.3333333333 %
Std accuracy:  20.0111080264 %
```

C.2.2. Resultados de test

```
## Using images without background
# MobileNet
Model1 val accuracy: 75.9339975093 %
Model2 val accuracy: 84.3088418431 %
```

```
Model3 val accuracy: 77.5529265255 %
Model4 val accuracy: 75.0622665006 %
Model5 val accuracy: 80.2926525529 %

mean= 78.6301369863 %
std= 3.35174274426 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[1597, 266, 10, 0, 28, 51, 101, 0]
[4, 1184, 3, 2, 2, 7, 84, 10]
[86, 161, 1113, 96, 282, 6, 317, 12]
[0, 3, 192, 1825, 0, 2, 16, 3]
[40, 201, 209, 0, 1817, 1, 199, 1]
[21, 21, 35, 90, 6, 1656, 47, 0]
[55, 196, 108, 0, 9, 2, 1038, 1]
[2, 3, 50, 7, 36, 0, 348, 2398]
Accuracy per class:  [88 58 64 90 83 96 48 98]
Mean accuracy:  78.125 %
Std accuracy:  17.6382928596 %

# MobileNet-A with data augmentation
Model1 val accuracy: 89.4146948941 %
Model2 val accuracy: 89.8194271482 %
Model3 val accuracy: 81.5691158157 %
Model4 val accuracy: 83.8885429639 %
Model5 val accuracy: 89.6170610212 %

mean= 86.8617683686 %
std= 3.45569261526 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright']
[3234, 328, 13, 36, 163, 66, 1, 5]
[1, 2480, 0, 0, 15, 0, 115, 174]
[180, 466, 3479, 141, 407, 7, 36, 49]
[57, 84, 171, 3487, 58, 350, 4, 63]
[12, 18, 13, 0, 3223, 0, 4, 0]
[62, 134, 5, 58, 8, 3445, 9, 105]
[27, 481, 33, 0, 8, 2, 4400, 27]
[37, 79, 26, 18, 23, 35, 6, 4152]
Accuracy per class:  [89 60 93 93 82 88 96 90]
Mean accuracy:  86.375 %
Std accuracy:  10.7114133054 %

# MobileNet-U with additional 'unknown' class
Model1 val accuracy: 63.4872761546 %
Model2 val accuracy: 64.5994344958 %
```

```
Model3 val accuracy: 69.575871819 %
Model4 val accuracy: 71.1969839774 %
Model5 val accuracy: 65.4288407163 %

mean= 66.8576814326 %
std= 2.99061874235 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[1375, 108, 3, 0, 41, 56, 8, 0, 194]
[3, 155, 0, 0, 13, 3, 3, 0, 25]
[20, 0, 573, 12, 32, 25, 91, 2, 28]
[0, 0, 211, 1080, 29, 1, 5, 0, 1]
[2, 0, 1, 0, 959, 0, 127, 0, 1]
[0, 0, 9, 0, 104, 1407, 47, 0, 0]
[25, 3, 13, 0, 13, 13, 406, 1, 51]
[0, 0, 0, 0, 187, 0, 309, 1659, 45]
[380, 1769, 910, 928, 802, 220, 1154, 763, 10120]
Accuracy per class:  [76  7 33 53 43 81 18 68 96]
Mean accuracy:  52.7777777778 %
Std accuracy:  28.3383437836 %

# MobileNet with additional 'unknown' class
Model1 val accuracy: 74.9293119698 %
Model2 val accuracy: 76.7954759661 %
Model3 val accuracy: 56.6069745523 %
Model4 val accuracy: 67.2007540057 %
Model5 val accuracy: 66.9745523091 %

mean= 68.5014137606 %
std= 7.14996572057 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[1499, 42, 19, 3, 118, 63, 2, 4, 290]
[2, 237, 0, 0, 2, 1, 1, 1, 44]
[65, 41, 830, 96, 202, 30, 195, 8, 924]
[6, 0, 94, 1309, 0, 0, 51, 0, 5]
[29, 3, 0, 0, 1243, 0, 2, 0, 2]
[0, 0, 0, 6, 11, 1528, 25, 0, 0]
[0, 11, 59, 0, 0, 0, 739, 8, 124]
[0, 0, 5, 0, 74, 0, 376, 1791, 82]
[204, 1701, 713, 606, 530, 103, 759, 613, 8994]
Accuracy per class:  [83 11 48 64 57 88 34 73 85]
Mean accuracy:  60.3333333333 %
Std accuracy:  24.3857882109 %

# MobileNet-UA with additional 'unknown' class
```

```
Model1 val accuracy: 73.9302544769 %
Model2 val accuracy: 76.7106503299 %
Model3 val accuracy: 70.3958529689 %
Model4 val accuracy: 75.4288407163 %
Model5 val accuracy: 74.2035815269 %

mean= 74.1338360038 %
std= 2.11356742603 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[2526, 98, 0, 0, 189, 38, 36, 3, 178]
[0, 133, 0, 0, 0, 0, 0, 3, 3]
[22, 1, 2251, 33, 16, 2, 54, 28, 245]
[30, 36, 175, 2597, 53, 271, 0, 15, 68]
[11, 0, 34, 0, 2981, 0, 0, 2, 13]
[78, 0, 0, 20, 10, 3126, 86, 1, 15]
[15, 11, 7, 0, 0, 29, 2597, 1, 96]
[122, 58, 0, 52, 7, 52, 69, 2949, 144]
[806, 3733, 1273, 1038, 649, 387, 1733, 1573, 20168]
Accuracy per class:  [69  3 60 69 76 80 56 64 96]
Mean accuracy:  63.6666666667 %
Std accuracy:  24.2074368738 %

# MobileNet-A with additional 'unknown' class
Model1 val accuracy: 82.0452403393 %
Model2 val accuracy: 81.9415645617 %
Model3 val accuracy: 77.342130066 %
Model4 val accuracy: 82.0452403393 %
Model5 val accuracy: 78.1903864279 %

mean= 80.3129123468 %
std= 2.09690687542 %

Confusion Matrix
['up', 'down', 'left', 'right',
    'upleft', 'upright', 'downleft', 'downright', 'unknown']
[2937, 312, 9, 7, 124, 64, 0, 1, 693]
[23, 463, 0, 0, 0, 0, 32, 28, 210]
[5, 2, 3049, 7, 99, 2, 17, 0, 89]
[11, 3, 66, 3291, 12, 219, 0, 47, 163]
[22, 4, 3, 0, 3375, 11, 2, 0, 29]
[48, 3, 0, 17, 5, 3499, 0, 0, 42]
[40, 39, 17, 4, 6, 1, 3471, 14, 466]
[8, 8, 0, 6, 9, 0, 12, 3509, 226]
[516, 3236, 596, 408, 275, 109, 1041, 976, 19012]
Accuracy per class:  [81 11 81 87 86 89 75 76 90]
Mean accuracy:  75.1111111111 %
Std accuracy:  23.2256775739 %
```

Apéndice D

Manual de usuario del prototipo

Este manual detalla los pasos a seguir para ejecutar el prototipo de reconocedor visual de gestos en el repositorio: `https://github.com/LeonBP/Drone_Rviz_Visualization_Control_DeepLearning`

Antes de lanzar el programa, este prototipo hace uso de ciertas herramientas y programas sin los que la ejecución no puede realizarse:

- *Robot Operating System* (ROS)[1]: Es necesario porque es empleado para la comunicación entre los distintos procesos y con el simulador del dron.
- *rotors_simulator*[2]: Paquete dentro de ROS para simular un dron volador. Permite visualizar el efecto de los comandos dados al dron.
- *Python 2.7*[3]: Leguaje empleado para programar el sistema.
- *Caffe2*[4]: *Framework* de *Python* para creación y entrenamiento de redes neuronales. Se utiliza para ejecutar el detector de personas.
- *Detectron*[5]: Implementación oficial del modelo que se ha utilizado en el detector de personas.
- *Tensorflow*[6]: Otro *framework* de *Python* para creación y entrenamiento de redes neuronales.Necesario para la ejecución del modelo de clasificación de gestos empleado.

Durante los pasos necesarios para lanzar el programa, es necesario ejecutar varios terminales distintos, por lo que se han incluído imágenes para poder observar lo que debe aparecer. Hay que tener en cuenta que es posible que los mensajes no sean exactamente iguales.

[1] `http://www.ros.org/`
[2] `http://wiki.ros.org/rotors_simulator`
[3] `https://www.python.org/downloads/`
[4] `https://caffe2.ai/`
[5] `https://github.com/facebookresearch/Detectron`
[6] `https://www.tensorflow.org/?hl=es`

En primer lugar, hay que iniciar ROS. Para ello se abre **un terminal** y se ejecuta el comando:

`roscore`

```
mrrobot@mrrobot:~$ roscore
... logging to /home/mrrobot/.ros/log/11e15dc4-ee4e-11e8-9e59-70
8bcdabfb3e/roslaunch-mrrobot-32559.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://mrrobot:44427/
ros_comm version 1.12.14


SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.14

NODES

auto-starting new master
process[master]: started with pid [32569]
ROS_MASTER_URI=http://mrrobot:11311/

setting /run_id to 11e15dc4-ee4e-11e8-9e59-708bcdabfb3e
process[rosout-1]: started with pid [32582]
started core service [/rosout]
```

Figura D.1: Apariencia del terminal tras ejecutar el comando *roscore*

El resultado se ven en la figura D.1. Esto inicializa ROS y se queda abierto.

A continuación, se lanza el sistema de captura de imágenes en un **nuevo terminal**. En el caso de la simulación en este trabajo, se utilizó la cámara Intel® RealSense™Camera R200. Se descargó el paquete de utilidades de ROS para esta cámara[7] y su ejecución es con el comando:

`roslaunch realsense_camera r200_nodelet_default.launch`

```
Intel RealSense F200_camera ; 2.60.0.0
Intel RealSense LR200_camera ; 2.0.71.18
Intel RealSense R200_camera ; 1.0.72.06
Intel RealSense SR300_camera ; 3.10.10.0
Intel RealSense ZR300_adapter ; 1.29.0.0
Intel RealSense ZR300_camera ; 2.0.71.28
Intel RealSense ZR300_motion_module ; 1.25.0.0
[ INFO] [1542888137.542950041]: /camera/driver - Detected the following camera:
                                - Serial No: 2411013349, USB Port ID: 2-3, Name:
 Intel RealSense R200, Camera FW: 1.0.72.06
[ INFO] [1542888137.543042255]: /camera/driver - Connecting to camera with Seria
l No: 2411013349, USB Port ID: 2-3
[ INFO] [1542888138.108047714]: /camera/driver - Setting static camera options
[ INFO] [1542888138.133555895]: /camera/driver - Enabling Depth in manual mode
[ INFO] [1542888138.134448455]: /camera/driver - Enabling Color in manual mode
[ INFO] [1542888138.135922502]: /camera/driver - Starting camera
[ INFO] [1542888138.160927365]: /camera/driver - Publishing camera transforms (/
tf_static)
[ INFO] [1542888138.161425523]: /camera/driver - Setting dynamic camera options
(r200_dc_preset=5)
[ INFO] [1542888138.909608948]: /camera/driver - Initializing Depth Control Pres
et to 5
[ INFO] [1542888140.523718602]: /camera/driver - Setting dynamic camera options
```

Figura D.2: Apariencia del terminal tras iniciar la cámara.

El resultado se ve en la figura D.2. Sin embargo, cualquier sistema de publicación de imágenes en ROS es válido si se coloca el nombre del canal donde se publican en los ficheros `Config.py` y `Visual_Sim.rviz`.

[7] `http://wiki.ros.org/RealSense`

Ahora se abre una **tercera terminal** que va a lanzar el simulador de drones. Lanzar el simulador requiere que se carguen las herrramientas del paquete *catkin* en ROS, lo que se hace con el siguiente comando, suponiendo que *catkin* se encuentra en el directorio base:

```
source  /catkin_ws/devel/setup.bash
```

Y ya se puede lanzar el simulador con el comando:

```
roslaunch rotors_gazebo mav_hovering_example.launch mav_name:=firefly
world_name:=basic
```

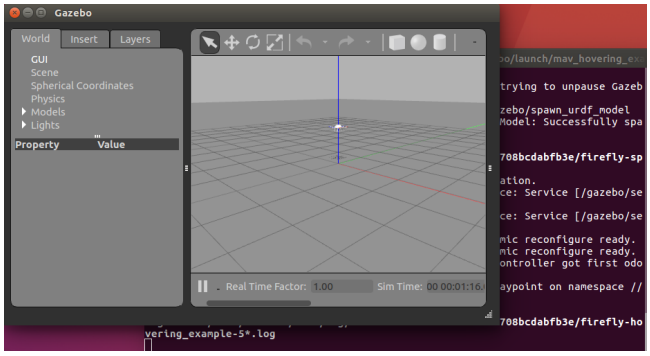


Figura D.3: Apariencia del simulador de drones utilizado.

El resultado se ven en la figura D.3. De esta ejecución se abre una nueva ventana en la que se ve un dron simulado arrancar.

A continuación se pasa a la ejecución del programa propiamente dicho. En una **nueva terminal**, se carga el paquete *catkin* otra vez:

```
source  /catkin_ws/devel/setup.bash
```

Y se ejecuta el primer *script* de *Python* del programa:

```
python Viz_Dron.py
```

El resultado se ven en la figura D.4. Esto abre otra ventana con la interfaz gráfica del programa.

En la **quinta terminal** se carga *catkin* de nuevo:

```
source  /catkin_ws/devel/setup.bash
```

Y se ejecuta el segundo *script* del programa:

```
python Moves.py
```

El resultado se ven en la figura D.5. Esto añade unos puntos azules en la interfaz gráfica que indican los posibles comandos aceptados.

En último lugar, en una **sexta terminal**, es necesario cargar *catkin*:

```
source  /catkin_ws/devel/setup.bash
```

Pero además hay que asegurarse de *Caffe2*, el *Detectron* y *Tensorflow* están incluídos en el `PATH` del sistema. Siendo esto así, no hay más que lanzar el tercer y último *script* del programa:

```
python Mov_Ev.py
```

El resultado se ven en la figura D.6. Con esto, se cargarán los modelos y se comenzará la ejecución.

El modo de empleo del simulador es:

1. Una persona debe aparecer en la imagen y señalar en una dirección.

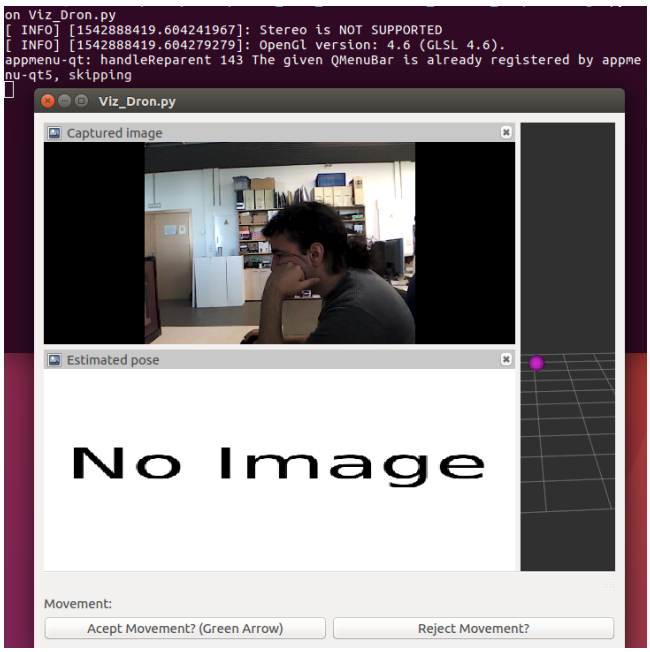


Figura D.4: Apariencia de la interfaz gráfica del programa.

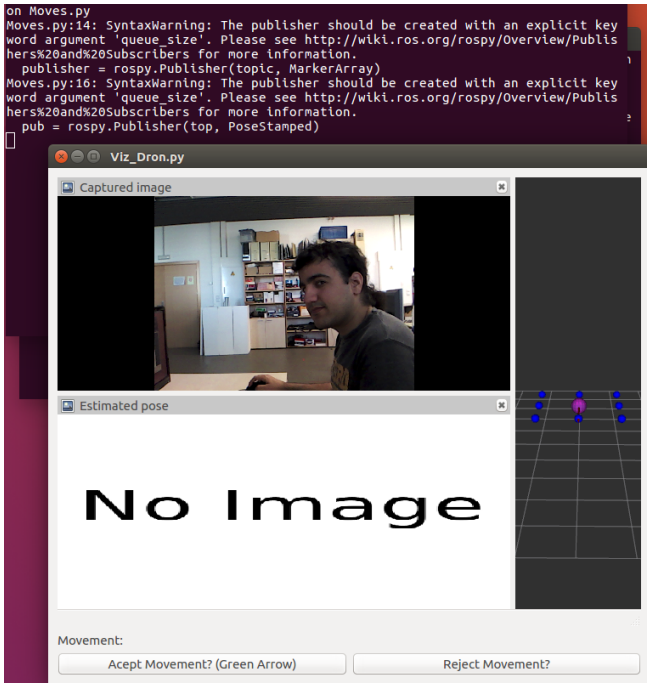


Figura D.5: Apariencia del programa tras lanzar el script *Moves.py*.

2. Esperar a que el detector la reconozca. Cuando esto ocurra parecerá en la

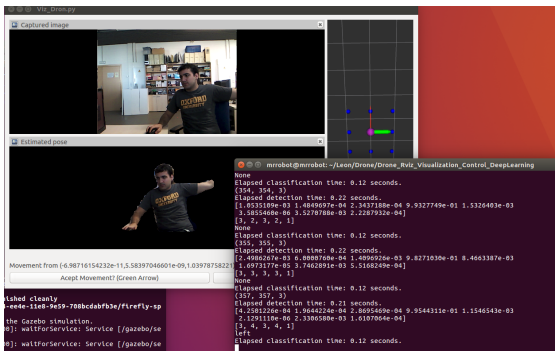


Figura D.6: Apariencia del programa completo.

interfaz gráfica una imagen recortada de la persona en la imagen. Además, aparecerá una flecha verde indicando la dirección en la que se ha reconocido que el dron debe avanzar según la indicación de la persona en la imagen.

3. Si el gesto ha sido reconocido correctamente o si se quiere que el dron avance en la dirección de la flecha verde, hacer *click* sobre el botón que dice *Accept Movement? (Green Arrow)* y observar cómo se desplaza el dron en la simulación. Si no, pulsar el botón *Reject Movement?* para descartar el movimiento.

4. Repetir hasta que se esté satisfecho con la simulación.

Bibliografía

[1] Zhengyou Zhang. Microsoft kinect sensor and its effect. *IEEE multimedia*, 19(2):4–10, 2012.

[2] Emanuele Principi, Stefano Squartini, Roberto Bonfigli, Giacomo Ferroni, and Francesco Piazza. An integrated system for voice command recognition and emergency detection based on audio signals. *Expert Systems with Applications*, 42(13):5668–5683, 2015.

[3] Ryan Aylward and Joseph A Paradiso. A compact, high-speed, wearable sensor network for biomotion capture and interactive media. In *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, pages 380–389. IEEE, 2007.

[4] Pavlo Molchanov, Shalini Gupta, Kihwan Kim, and Jan Kautz. Hand gesture recognition with 3d convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 1–7, 2015.

[5] Giulio Marin, Fabio Dominio, and Pietro Zanuttigh. Hand gesture recognition with leap motion and kinect devices. In *Image Processing (ICIP), 2014 IEEE International Conference on*, pages 1565–1569. IEEE, 2014.

[6] Frederik Schaffalitzky. Human interaction with unmanned aerial vehicles, March 20 2018. US Patent 9,921,579.

[7] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.

[8] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[10] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[11] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005.*

IEEE Computer Society Conference on, volume 1, pages 886–893. IEEE, 2005.

[12] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. *arXiv preprint arXiv:1611.08050*, 2016.

[13] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[14] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017.

[15] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[17] Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. Detectron. `https://github.com/facebookresearch/detectron`, 2018.

[18] Antonio W Vieira, Thomas Lewiner, William Robson Schwartz, and Mario Campos. Distance matrices as invariant features for classifying mocap data. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 2934–2937. IEEE, 2012.

[19] Antonin Bernardin, Ludovic Hoyet, Antonio Mucherino, Douglas Gonçalves, and Franck Multon. Normalized euclidean distance matrices for human motion retargeting. In *Proceedings of the Tenth International Conference on Motion in Games*, page 15. ACM, 2017.