Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Máster Universitario en Inteligencia Artificial

Trabajo Fin de Máster

Aplicación de Sensores Inerciales en Interfaces Robóticos

Autor: Gabriel Alejandro Peña Delfín
Tutor: Javier De Lope Asiaín

Madrid, Julio 2022

Este Trabajo Fin de Máster se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Máster
Máster Universitario en Inteligencia Artificial

Título: Aplicación de Sensores Inerciales en Interfaces Robóticos

Julio 2022

Autor: Gabriel Alejandro Peña Delfín
Tutor: Javier De Lope Asiaín
Inteligencia Artificial
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

Los robots industriales son ampliamente utilizados para realizar tareas repetitivas o en escenarios en donde se que requiere de una precisión o fuerza superior a la de un humano, sin embargo, existen ciertas limitaciones que impiden una colaboración más cercana entre humano-robot. Por ejemplo, existe el riesgo de que un robot durante su periodo de operación hiera a un trabajador que se encuentre cerca de su área de trabajo, debido a este riesgo, normalmente los robots y humanos se encuentran separados por una celda. Este entorno de trabajo no resulta muy favorable si se quiere optimizar la producción mediante una colaboración humano-robot.

Una de las formas de fomentar esta colaboración es mediante el control del robot mediante gestos o movimientos con la mano, con este método el usuario le puede indicar al robot cómo y cuándo debe moverse, así como las tareas que tiene que realizar. En este trabajo se utilizan sensores inerciales para el control de un robot colaborativo, para ello se desarrolla un modelo de reconocimiento de gestos en tiempo real que le permite al usuario controlar a un robot.

Dado que se desea contribuir a conseguir un entorno en donde un humano y un robot trabajen juntos de forma colaborativa, es importante obtener un modelo de reconocimiento con la suficiente precisión como para reducir o eliminar los riesgos que causan su separación. En este trabajo, se hace una revisión del estado del arte en el contexto del reconocimiento de gestos o movimientos en la robótica, para después aplicar los métodos más potentes en el desarrollo de un sistema con un modelo que permita al usuario controlar a un robot de forma precisa.

El sistema desarrollado se pone a prueba en un experimento en un entorno simulado, en donde se analiza el comportamiento del mismo al realizar acciones de recoger y dejar objetos con un robot colaborativo.

Abstract

Industrial robots are widely used for repetitive tasks or scenarios that require strength or precision superior to that of humans, however, there are certain limitations that hinder a closer kind of human-robot collaboration. For instance, there is a risk that a robot during its working period might injure a worker who is close to its working area, due to this risk, robots and humans are usually separated by a cell. This workspace is not very favorable if we want to optimize production by human-robot collaboration.

One of the ways to encourage this collaboration is with movement or gesture-controlled robots, with this method the user is able to indicate to the robot how and when it should move, and the tasks that it has to complete. In this work inertial sensors are used to control a collaborative robot, a real time gesture recognition model is developed which allows the user to control a collaborative robot.

Since contributing to achieve a workspace were humans and robots work together in a collaborative fashion is desired, it is important to obtain a recognition model with enough precision in order to reduce or eliminate the risks that cause their separation. In this work, the state-of-the-art in the context of gesture or movement recognition in robotics is revised, in order to apply the most potent methods in the development of a system with a model that allows the user to precisely control a robot.

The developed system is put to the test in an experiment in a simulated environment, where the system's behaviour is analyzed while performing pick and place tasks with a collaborative robot.

Tabla de contenidos

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Motivación

La Robótica proporciona una gran ayuda en tareas que normalmente serían difíciles de cumplir por parte del personal habitual, este tipo de situaciones se ven con más frecuencia en el entorno industrial, en donde es habitual encontrar tareas repetitivas y difíciles de ejecutar. Es por eso que es ampliamente investigado el desarrollo de sistemas robóticos que sean capaces de solventar estos problemas, asistiendo al trabajador y mejorando tanto su calidad de vida como su productividad. Los robots colaborativos o Cobots permiten al operador ejecutar tareas que debido a su complejidad, resultan muy complicado de automatizar, y además, les permiten llevar a cabo tareas peligrosas de una forma segura, como el transporte de sustancias tóxicas o la carga y descarga de objetos pesados. Todos estos trabajos se ejecutarán de una forma más eficiente gracias a la ayuda de un robot programado para tales fines.

Tradicionalmente, los robots industriales operan en una celda que lo separa del trabajador, esto se hace para proteger al personal en caso de que el robot falle. Sin embargo, recientemente se ha estado experimentando con entornos en donde el trabajador y el robot operan sin estar separados por esa celda (Figura 1.1). La posibilidad de que exista una colaboración entre humano y robot está ganando más popularidad con el objetivo de combinar la fuerza, resistencia y precisión del robot con la intuición, flexibilidad y resolución de problemas de lo humanos.

Una de las formas de conseguir una colaboración entre humanos y robots es a través de unos sistemas de reconocimento de los gestos que realizan humanos. El robot se encuentra a la espera de recibir alguna señal por parte del operador, y en cuanto reciba dicha señal ejecutará la acción correspondiente. Esto es aplicable en líneas de montaje en donde el robot se encargaría de transportar las herramientas y partes necesarias para la tarea según los gestos que perciba del operador.

Dentro del marco de reconocimiento de gestos, los sistemas más habituales son los que se componen de cámaras, sin embargo, también se pueden encontrar sistemas de reconocimiento de voz o también con sensores inerciales o IMU (Inertial Measuerement Unit).

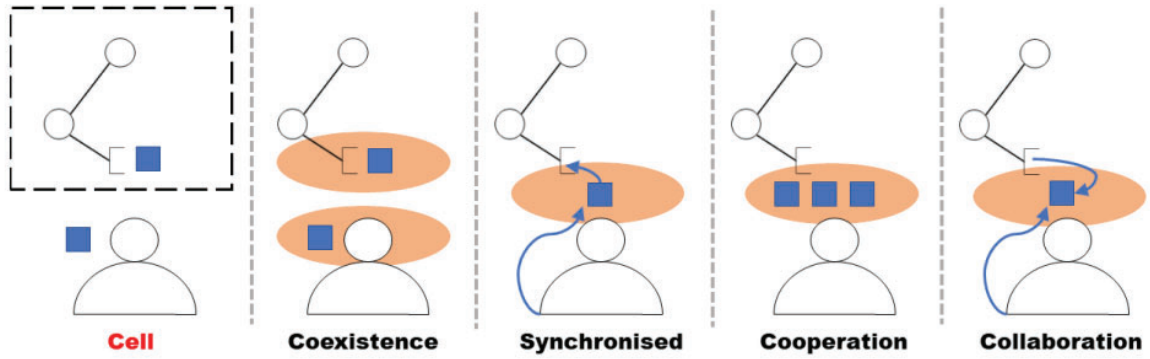

Figura 1.1: Tipos de uso de un robot [Matheson *et. al* (2019)].

1.2. Objetivos

En relación al uso que se les da a los Cobots actualmente, y al tipo de tareas que se realizan, los objetivos de este Trabajo de Fin de Máster son los siguientes:

- Desarrollar un sistema capaz de recononcer de forma continua los gestos del usuario para realizar determinadas tareas con un robot colaborativo.
- Arrojar luz sobre la viabilidad en el uso de sensores inerciales para la realización de tareas con un robot colaborativo.
- Estudiar los métodos más novedosos de reconocimiento continuo de gestos y sus aplicaciones en la robótica.
- Conocer y aplicar las herramientas y técnicas para la obtención de datos a partir de sensores inerciales.
- Estudiar los modelos neuronales más potentes para el tratamiento de datos en serie cronológica.

1.3. Estructura del documento

El documento contiene la siguiente estructura: el Capítulo 1 1 introduce el contexto del proyecto, explicando la motivación y objetivos del mismo; en el Capítulo 2 se expone el Estado del Arte respecto a los métodos de reconocimiento de gestos aplicados a la robótica; a continuación, en el Capítulo 3 se realiza una explicación *top-down* del sistema desarrollado para comentar en el Capítulo 4 los resultados obtenidos; finalmente, en el Capítulo 5 se concluye el trabajo comentando los aspectos más relevantes y los trabajos futuros.

Capítulo 2

Estado del Arte

El lenguage corporal es vital para la transmisión de información entre los seres humanos, y por ende, juega un papel importante en la interacción entre humanos y robots. Para que exista una comunicación fluida entre ambas partes debe existir una serie de gestos definidos que tengan un significado conocido por ambos. Además, el robot debe ser capaz de ”entender” estos gestos para luego ejecutar la acción correspondiente.
En la literatura existen diversos métodos de reconocimiento de gestos, la gran mayoría están basados en el uso de Sensores Inerciales o tecnologías de Visión por Computador para la captación de datos, sin embargo, también se pueden encontrar otras estrategias menos exploradas como el uso de la Electromiografía, ultrasonidos o infrarojos. A continuación se va a revisar los métodos más novedosos basados en la clasificación mencionada anteriormente.

2.1. Enfoques basados en sensores inerciales

Las Unidades de Medición Inercial (IMU) presentan una alternativa menos costosa en cuanto a hardware frente a los enfoques basados en Visión, además, con el incremento en popularidad de las tecnologías IoT, los sistemas *Wearable* están ganando una gran popularidad en el reconocimiento de gestos y en las aplicaciones con la robótica. Trabajos como el de [Bai *et. al* (2012)] demuestran que es posible reconocer los movimientos de una persona según los datos relativos a la aceleración de sus movimientos. En este caso los datos se toman usando el chip MMA7260Q que se colocó en la cadera del usuario. En este artículo se propone el método de los *Movelets* para clasificar los movimientos, que consiste en tomar un número determinado de muestras por un tiempo determinado de cada movimiento, contruyendo así un "diccionario"de movimientos, y por lo tanto al clasificar un movimiento que no está etiquetado simplemente se le asocia a la misma etiqueta que los movimientos más "parecidos", es decir se utiliza una función de distancia y la etiqueta se asocia al conjunto de movimientos del "diccionario"que sean más cercanos al que se quiere clasificar, de forma similar al algoritmo KNN (Figura 2.1).

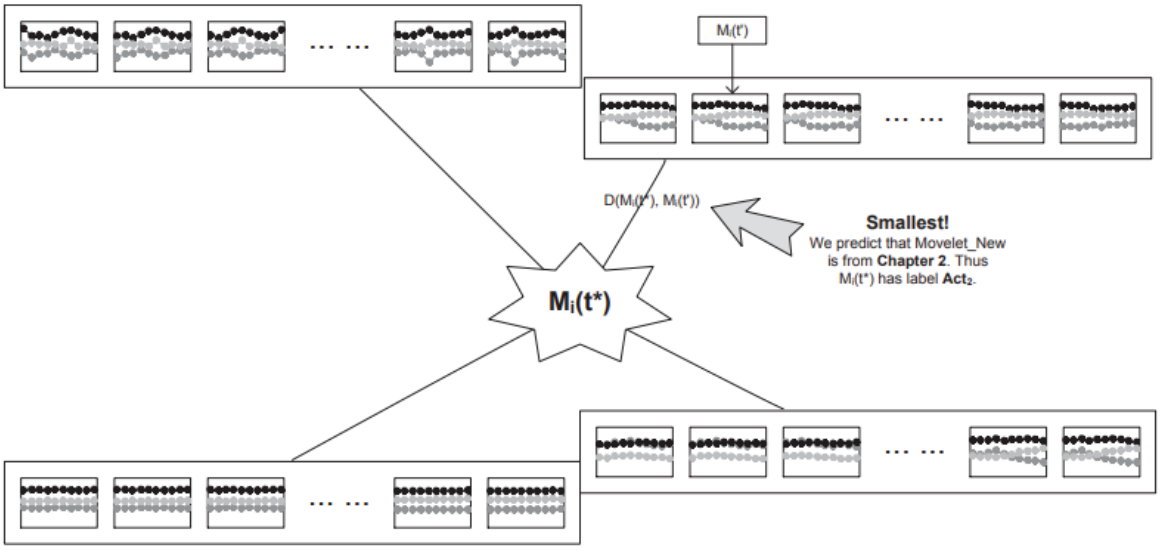

Figura 2.1: Método de clasificación de movimientos propuesto por [Bai *et. al* (2012)].

[Huang y Onnela (2019)] continuan con el método propuesto anteriormente. Pero en este artículo se tienen en cuenta los datos sobre la velocidad angular, a parte de los de aceleración. Se utiliza un Iphone 5S y un Iphone 7 que se colocan en los bolsillos del usuario porque vienen integrados con un giroscopio y un acelerómetro, además, se emplean sensores *ActiGraph GT9X Link* en las muñecas y el tobillo derecho que también contienen un acelerómetro y giroscopio y se adapta el método de los *Movelets* (propuesto por[Bai *et. al* (2012)]) a este problema. En este caso el tiempo de medida de cada Movelet es de una ventana de 1 segundo, y teniendo en cuenta que se asume que se estarán captando los datos con una frecuencia continua, esta ventana se "desliza"desde el primer dato hasta el último. El sistema es capaz de determinar cuándo una persona está de pie sin moverse, caminando por una superficie plana, subiendo escaleras, bajando escaleras, sentándose o poniéndose de pie.

[Fathi y Curran (2017)] presenta un sistema de reconocimiento de posturas inadecuadas durante el trabajo. La principal motivación de este artículo es la aparición constante de lesiones de la espalda causadas por el entorno laboral. Los autores utilizan los sensores Shimmer que están colocados en 3 zonas de la columna para recoger los datos de aceleración y velocidad angular del sujeto (Figura 2.2), y posteriormente se aplica el algoritmo de SAX (Symbolic Aggregate approXimation) para clasificar las posturas. SAX permite reducir la dimensión de los datos de serie temporal convirtiéndolos en una serie de caracteres en donde cada caracter representa la media de valores de un segmento de la muestra de datos, posteriormente, se calcula la distancia entre las cadenas para determinar la postura a la que corresponde, de forma similar al algoritmo KNN, además, se emplea el método de la ventana "deslizante"para el reconocimiento continuo. Con este sistema se consigue prevenir lesiones como la Espondilitis Anquilosante, que está fuertemente relacionada con los malos hábitos en el entorno de trabajo, detectando las posturas inadecuadas del sujeto (espalda encorvada al estar sentado) y proporcionándole el feedback necesario.

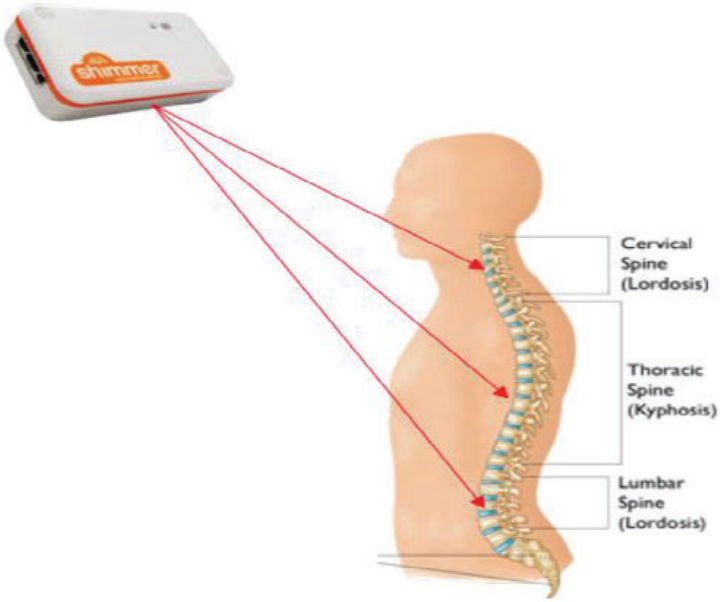

Figura 2.2: Ubicaciones de los 3 sensores Shimmer ([Fathi y Curran (2017)]).

En el grupo de reconocimiento de gestos a través de *Wearables* se encuentran trabajos como el de [Wang *et. al* (2018)] en donde se propone un sistema interactivo robot-humano para entornos industriales de ensamblaje. Dicho sistema está compuesto de un IMU, que se utiliza para recoger los datos relativos a los movimientos que realiza el usuario con brazo, y de un sensor de Electromiografía (EMG), que se encargan de recoger los datos relativos a la activación múscular en su antebrazo. Durante la adquisición de datos se aplica el filtrado de Kalman para reducir el ruido que presenta el IMU por defecto. Este sistema se considera *Wearable* porque se trata de un reloj que se puede colocar fácilmente en la muñeca del usuario. Con la ayuda de estos sensores y Modelos Ocultos de Markov (HMM), se consigue crear un clasificador capaz de reconocer gestos de "pedir" o "dar" objetos (Figura 2.3), que son esenciales para tareas de ensamblaje, de tal forma que el sistema es capaz de reconocer las intenciones del usuario.

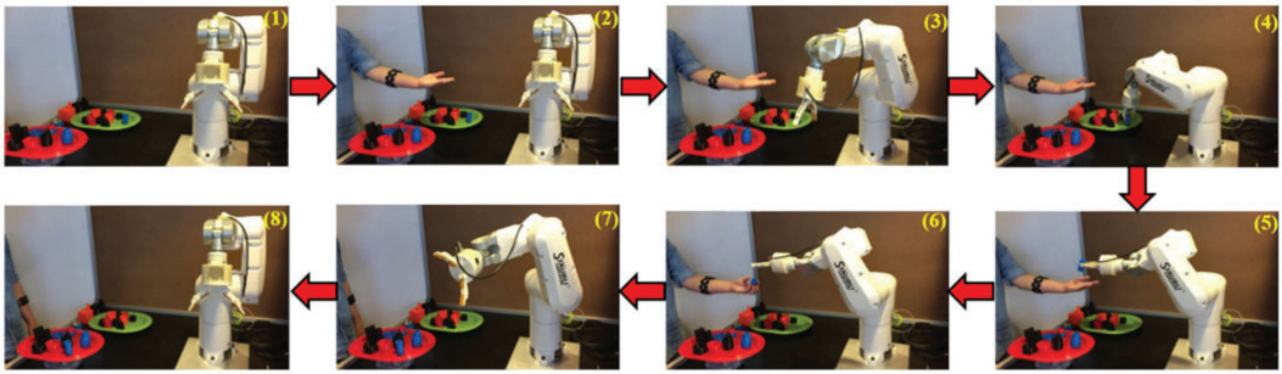

Figura 2.3: Caso de uso en donde el usuario le "pide" un objeto al robot ([Wang *et. al* (2018)]).

[Jiang *et. al* (2017)] desarrolla una pulsera capaz de reconocer gestos en tiempo real a partir de una fusión de sensores. El dispositivo consta de un IMU y de una Electromiografía (Figura 2.4) y se aplica el método de la ventana "deslizanteçomentado anteriormente para poder reconocer los gestos en tiempo real. Como se están utilizando 2 tipo de sensores de forma simultánea, es posible encontrar gran cantidad de ruido y de datos redundantes, por lo tanto, los autores aplican un proceso de extracción de Características que facilite la clasifiación de gestos. Se eligieron características como la Desviación Media porque proporciona información acerca de la fuerza de las señales y la amplitud, y la Longitud de Onda para obtener información acerca de la duración del movimiento. Con la ayuda de un algoritmo de clasificación basado en Análisis Discriminante Lineal (LDA) se consiguen recononcer hasta 12 gestos en donde se incluyen tanto gestos relativamente simples como levantar un dedo o cerrar el puño, como movimientos más complejos como un chasquido de dedos.

Figura 2.4: Prototipo de pulsera con el IMU y los sensores de EMG ([Jiang *et. al* (2017)]).

[Wen *et. al* (2016)] presentan un sistema llamado *Serendipity*, que se compone de un algoritmo de clasificación que recibe los datos a partir del smartwatch Samsung Galaxy Gear, que contiene un acelerómetro y un giroscopio. Durante la obtención de datos se realizaron medidas de 10 segundos por gesto y una frecuencia de muestreo de 50hz. Mediante un tamaño de ventana de 1 segundo se extrajeron 7 features de cada gesto en donde se incluyen la media, desviación estándar, los máximos, mínimos y 3 cuartiles. Con un algoritmo basado en el Support Vector Machine (SVM) se consiguen distinguir hasta 5 gestos con la mano (pellizcar, hacer click con el dedo, frotarse los dedos, cerrar el puño, saludar) que pueden ser de utilidad para interactuar con otros dispositivos, al ser fácilmente desplegable en el Samsung Galaxy Gear. El sistema también cuenta con método para evitar falsos positivos, se aplicó un algoritmo basado en la Deformación dinámica del tiempo (DTW) y el KNN que calcula la distancia de cada gesto que contenga una etiqueta, si dicha distancia se encuentra dentro de un umbral entonces el sistema lo clasifica, en caso contrario se considera como ruido.

Los dedos de las manos resultan de gran utilidad para la comunicación gestual debida a la cantidad de información que es posible transmitir a través de sus movimientos, así lo demuestran [Mummadi *et. al* 2018], quienes proponen un sistema de detección de gestos con la mano, basado en un guante equipado de un acelerómetro, giroscopio y magnetómetro en la yema de cada dedo. Dado que los autores están enfocados únicamente en el movimiento de los dedos, es importante que el sistema identifique la orientación de cada dedo para que el proceso de reconocimiento funcio-

ne de manera satisfactoria, debido a esto, se aplica un proceso de fusión de sensores para reducir la desviación que puede presentarse durante el cálculo de los ángulos de rotación causado por una falta de referencia. Se aplica el Filtro Complementario con los 3 sensores mencionados para obtener los ángulos de giro en los 3 ejes. El modelo de clasificación está basado en el Random Forest classifier, con el que se logran clasificar hasta 22 gestos pertenecientes a la Lengua de señas francesa (Figura 2.5).

Figura 2.5: Ejemplo de reconocimiento de gestos con el sistema de [Mummadi *et. al* 2018].

2.2. Enfoques basados en Visión por Computador

Los enfoques basados en Visión aplican procesamiento de imágenes y reconocimiento de objetos, dentro de este grupo se encuentran trabajos como el de [Coupeté *et. al* (2015)] en donde se utiliza el Kinect de Microsoft para lograr reconocer los gestos realizados por el operador en su trabajo y conseguir que el robot se anticipe a sus acciones. El sistema está pensado para ser aplicado en líneas de ensamblaje, debido a esto la cámara se encuentra encima del usuario en el escenario que se observa en la Figura 2.6 y así no se estorba el trabajo del robot y el usuario además de tener una visión clara de sus movimientos. Antes de reconocer los gestos la primera tarea es la de llevar un seguimiento de las manos del usuario, el método que utilizan los autores es el de calcular la distancia geodésica entre cada punto del torso del usuario hasta la cabeza mediante el algoritmo de Dijkstra. Para ello se divide la imagen de profundidad en una cuadrícula y a cada punto se le asocia un peso equivalente a su profundidad. Por lo tanto los puntos que corresponden a las manos se obtienen con los 2 caminos de mayor distancia respecto de la cabeza. La distancia entre las manos a la cabeza también es utilizado como característica principal para el clasificador de gestos, dado que estas distancias cambian contínuamente durante el trabajo del usuario. El algoritmo de clasificación se obtiene combinando el algoritmo K-Means con los Modelos ocultos de Markov (HMM), primero se discretizan las características dividéndolas en *Clusters* con el K-Means, para luego entrenar el modelo HMM.

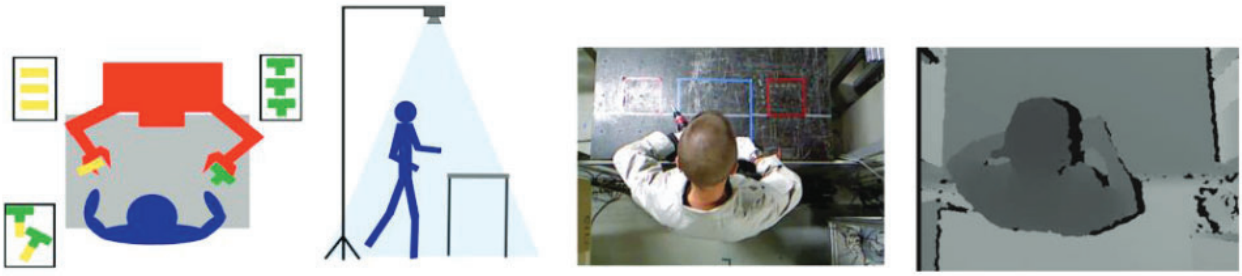

Figura 2.6: De izquierda a derecha: Escenario del caso de uso entre el operador (azul) y robot (rojo); posición de la cámara; imagen RGB obtenida; mapa de profundidad ([Coupeté *et. al* (2015)]).

En [Erdoğan *et. al* (2016)] se diseña un sistema de reconocimiento de gestos de manos utilizando el Leap Motion Controller. Se trata de un sensor capaz de obtener los datos relativos a la posición de las manos y dedos de las personas. La información que recoge el Leap Motion Controller viene en forma de columnas en donde se van muestreando los datos en el tiempo, cada columna representa la posición de uno de los dedos en uno de los ejes, por lo tanto el conjunto de datos contiene 30 columnas en total, 10 dedos y sus posiciones en los 3 ejes. Los autores diseñaron un escenario donde probar el sistema que contiene 3 cajas, una al lado de la otra, y con el algoritmo de clasificación que se obtiene a partir de una Red Neuronal Artificial se consigue determinar si una persona quiere coger una caja de los 3 disponibles. (Figura 2.7).

Figura 2.7: El sujeto elige uno de los 3 objetos y el algoritmo reconoce sus intenciones ([Erdoğan *et. al* (2016)]).

En [Mazhar *et. al* (2019)] diseña un framework dedicado a la colaboración humano-robot, en donde se utiliza el Microsoft Kinect y la Red Neuronal Convolucional Inception V3 para detectar gestos con las manos tomados de la Lengua de signos americana. Al igual que en la publicación comentada anteriormente, antes de reconocer los gestos de los individuos primero es necesario obtener un seguimiento de las manos

en las imágenes obtenidas, para ello los autores utilizan la librería OpenPose, con la que se obtienen las coordenadas de las articulaciones del usuario, a continuación, se estima la localización de las manos mediante el cálculo de la distancia entre el codo y la muñeca, esta distancia se extiene una cantidad de un tercio en la dirección de la mano para llegar al centro de la misma. Para asegurar la robustez del sistema, se procede a entrenar el modelo con imágenes de gestos con distintos fondos (Figura 2.8). El rendimiento del framework se valida con el manipulador robótico BAZAR y con el que además se propone una aplicación industrial que consiste en utilizar el reconocedor de gestos para guardar distintas posiciones del robot y posteriormente ejecutarlas, detenerr el robot durante la ejecución de un movimento o reanudar su actividad.

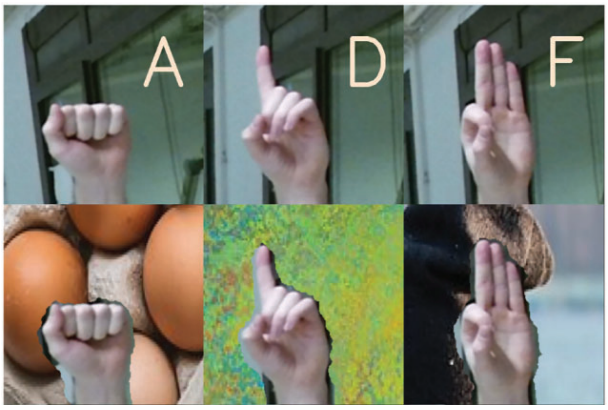

Figura 2.8: Ejemplos de imágenesde gestos originales y con el fondo cambiado ([Mazhar *et. al* (2019)]).

Los métodos que emplean Visión por Computador normalmente se basan en el reconocimiento de los gestos que realiza el usuario frente a una cámara o sensor de profundidad, sin embargo, [Chen *et. al* (2018)] propone un método poco explorado que se basa en reconocer los movimientos del usuario a partir de las imágenes de video que recoge un cámara colocada en su muñeca (Figura 2.9). El algoritmo del sistema se compone de una primera fase de extracción de características que están contenidos en las imágenes capturadas por la *WristCam*. Tanto la posición de la mano en la imagen como la velocidad de la trayectoria del movimiento son relevantes para la primera fase, la posición de la mano se obtiene ”separando” el fondo de la imagen de la mano que se encuentra en frente mediante el algoritmo de Lazy Snapping. La velocidad se obtiene con el algoritmo Speeded Up Robust Features (SURF). La segunda fase consiste en segmentar la imágenes de tal forma que el gesto esté representado por un conjunto de imágenes extraídas del video, y dicho conjunto debe ser disjunto respecto al de los gestos sucesivos, para ello los autores definieron un gesto con la mano que sirve de ”aviso” para el sistema de que el usuario va a realizar un gesto. Finalmente, se clasifica en tiempo real el gesto de cada segmento utilizando el algoritmo Dynamic Time Warping (DTW) y se propone una aplicación que consiste en que un robot dibuje las formas que el usuario va describiendo con la cámara en su muñeca.

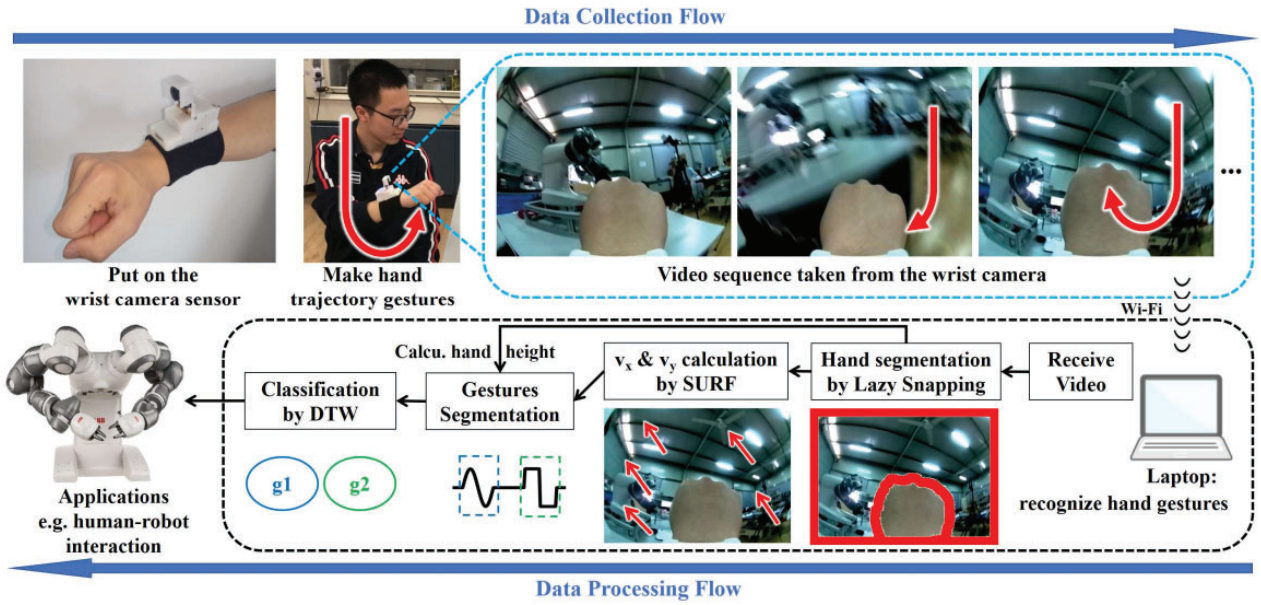

Figura 2.9: Flujo de trabajo para el reconocimiento de gestos de [Chen *et. al* (2018)].

2.3. Otros enfoques

En la literatura existen otros enfoques que están también dedicados al reconocimiento de gestos pero prescindiendo de sensores o cámaras como las mencionadas anteriormente. Un método ampliamente investigado es el del uso de sensores que se colocan en la piel del sujeto y miden el nivel de activación muscular del cuerpo humano en forma de actividad eléctrica. Un ejemplo de este tipo de práctica se puede encontrar en [Dong *et. al* (2017)], quienes proponen un sistema interactivo robot-máquina basado en sensores piezoeléctricos *Wearable* fabricados por los autores que se colocan en las muñecas del individuo. Dichos sensores se encargan de medir el nivel de activación muscular de los antebrazos y mediante la aplicación del algoritmo de Análisis Discriminante Lineal (LDA) se clasifican hasta 4 gestos distintos (Figura 2.10) con los que se consigue controlar un robot mediante instrucciones de "Avanzar hacia delante", "Girar a la derecha", "Girar a la izquierda", "Parar". La extracción de características se realiza mediante el método de la ventana deslizante comentada en publicaciones anteriores, con un porcentaje de solapamiento del 50 % se extraen valores como la media cuadrática, la longitud de la onda, el cruce por cero de cada onda y los cambios de signo de las pendientes.

Los sensores de Electromiografía superficiales también son objeto de estudio dentro de la reconocimiento de gestos, estos dispositivos recogen la actividad neuromuscular y detectan la actividad eléctrica del tronco nervioso en la superficie de la piel. Dentro de este grupo se pueden encontrar trabajos como el de [Shi *et. al* (2018)], en donde aplicando el algoritmo KNN se consiguen clasificar 4 gestos del usuario (cerrar el puño, levantar el índice, levantar el pulgar, levantar todos los dedos menos el pulgar) según las señales capturadas con la EMG del antebrazo para controlar una mano biónica. La extracción de características se realiza mediante un umbral obtenido a partir del cálculo de la media de la señal de referencia (cuando los músculos están relajados) más la desviación estándar de la señal multiplicada por 3, este valor se uti-

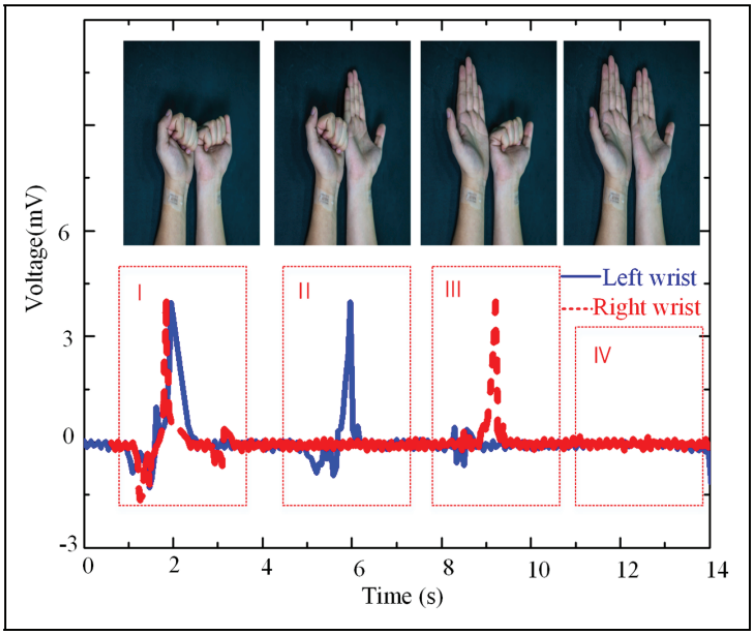

Figura 2.10: Señales recibidas de cada gesto ([Dong *et. al* (2017)]).

liza para determinar cuándo un gesto se debería tener en cuenta para la clasificación y cuándo se trata de un falso positivo. Una vez se identifica un gesto real, se debe cumplir la condición de que el umbral tiene que superarse durante al menos 4 segundos para que se tenga en cuenta y para asegurarse de que se capture el movimiento entero. Las características a extraer son una de las más populares en reconocimiento de gestos en donde se incluyen el valor absoluto medio, cruce por ceo, longitud de la onda y cambio de signo en las pendientes.

En la literatura es frecuente encontrar trabajos en donde únicamente se utiliza un tipo de sensor para el reconocimiento de movimientos. Sin embargo, [Jiang *et. al* (2020)] demuestran que es posible mejorar la precisión de los clasificadores de gestos mediante el uso de más de un tipo de sensor, formando u sistema híbrido. En este caso se emplean en conjunción un sensor de EMG y uno de FMG (force myography) para medir el nivel de contracción o relajación muscular de la zona del antebrazo, por lo tanto se enfoca en los cambios de volumen de la zona en donde se encuentra el músculo. Las características extraídas de las señales de EMG fueron la desviación media, cruce por ceros, cambios de signo en pendiente y la longitud de onda, mientras que de las señal de FMG se extrajo únicamente la desviación media. Para una clasificación continua se utilizó la ventana deslizante con un porcentaje de solapamiento del 25 %. Se aplica el algoritmo de Análisis Discriminante Lineal (LDA) para la clasificación de gestos que corresponden a los digitos 0-9 de la Lengua de signos americana. Los experimentos llevados a cabo por los autores demostraron que la utilización de EMG y FMG en conjunción supone un aumento de la precisión del clasificador en lugar de usar los sensores de forma aislada, puesto que la cantidad de información recogida de ambos sensores carece de redundancia.

A pesar de que la Electromiografía está asentada en el área de reconocimiento de gestos como un método estándar, sigue presentando varios inconvenientes con res-

pecto a la precisión de las medidas obtenidas sobre movimientos más complejos con los dedos de las manos. [Yang *et. al* (2018)] presentan una alternativa a estos inconvenientes, en el mencionado trabajo se propone un sistema de reconocimiento de gestos en tiempo real basado en sensores de ultrasonido en modo A, que son capaces de penetrar en la superficie de la piel con mayor profundidad que con el EMG, y captar la información recibia de una forma más precisa. El dispositivo con los sensores se coloca en el antebrazo del sujeto (Figura 2.11) y el sistema es capaz de reconocer 11 gestos con los dedos mediante un algoritmo de clasificación basado en el Análisis Discriminante Lineal (LDA). Dado que las lecturas iniciales obtenidas con el ultrasonido contienen gran cantidad de ruido, antes de extraer las características esenciales primero se aplicó un Filtro Gaussiano. Como las ondas de ultrasonidos recibidas son susceptibles de ser atenuadas por la profundidad de la superficie, los autores emplearon una Compensación de ganancia-tiempo (TGC) para evitar la pérdida de información. Cada onda leída fue dividida en segmentos, y en cada segmento se aplicó una Regresión lineal que finalmente fueron combinados para obtener la característica de dicha onda.

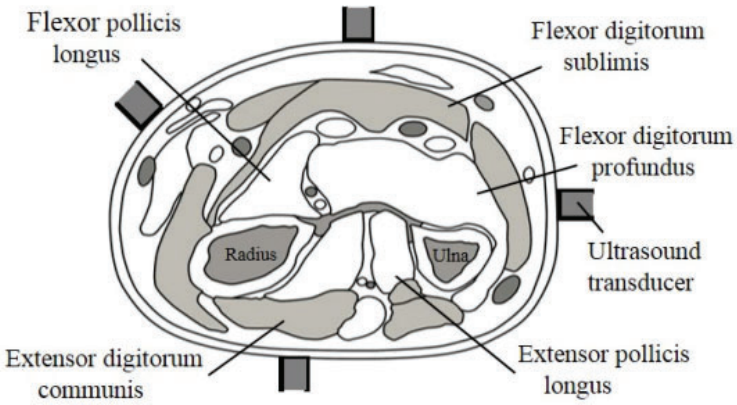

Figura 2.11: Posición de los sensores en del antebrazo ([Yang *et. al* (2018)]).

En [Ortega-Avila *et. al* (2015)] se presenta un método novedoso basado en las lecturas obtenidas a partir de la actividad muscular mediante sensores ópticos. Se trata de otro sistema *Wearable* que se coloca en el antebrazo del sujeto, el dispositivo está dotado de un sensor de infrarojos Near Infrared (NIR), que es capaz de detectar el movimiento de los tendones y músculos involucrados en los movimientos de los dedos de la mano. Para la clasificación de gestos se emplea el algoritmo K-NN y el Support Vector Machine (SVM).

[Kato y Takemura (2016)] exponen un método de reconocimiento poco explorado en la literatura basado en la detección de sonidos conducidos por el hueso. Se utiliza un dispositivo *Wearable* similar a una pulsera que contiene un micrófono de contacto y un actuador de vibración en la zona del radio y cúbito. El actuador emite una onda sobre el radio, y la onda reflejada se mide con el micrófono que está sobre el cúbito. La precisión de este método reside en la gran cantidad de información que se puede obtener a partir del movimiento de la zona de los huesos carpianos causados por los gestos de la mano. Para la clasificación de gestos se utiliza el SVM y los features a

extraer se obtienen calculando la Densidad Espectral de la vibración captada por el micrófono. Con este sistema se consiguen clasificar 7 gestos con una precisión satisfactoria.

En la literatura sobre reconocimiento de movimientos es frecuente encontrar trabajos en donde se utilizan los métodos clásicos de Aprendizaje Automático para llevar a cabo la tarea, como los Modelos Ocultos de Markov, Support Vector Machine, KNN o métodos basados en el teorema de Bayes como el Análisis discriminante lineal. Sin embargo, [Wang *et. al* (2020)] proponen un sistema de reconocimiento continuo basado en un potente modelo de red neuronal conocido como Long-short Term Memory (LSTM), que es un tipo de red neuronal recurrente capaz de modelar secuencias temporales, ya sean textos, voces, videos, o datos muestreados por sensores como los comentados en este capítulo. Los autores utilizan la base de datos NinaPro, que contiene datos obtenidos con EMG acerca de 6 movimientos con las manos, así como los ángulos de las articulaciones de las manos tras haber ejecutado alguno de los movimientos definidos con el guante *CyberGlove II*. Mediante el método de la ventana deslizante y un porcentaje de solapamiento del 50 % se consigue crear un sistema que es capaz de estimar de forma continua los ángulos de las articulaciones de las manos a partir de los datos de EMG que recibe de entrada.

Capítulo 3

Desarrollo

3.1. Descripción del sistema y sus componentes

A continuación se describe el sistema de reconocimiento desarrollado y especifican los algoritmos y métodos para el tratamiento de los datos así como los materiales utilizados para el proyecto.

Como ya se ha explicado en la Introducción, uno de los objetivos de este proyecto es el de desarrollar un sistema capaz de reconocer de forma continua y en tiempo real los gestos que realiza un individuo, para luego poder controlar un robot colaborativo de manera precisa. Dado este contexto, el diseño del sistema a alto nivel consiste de los siguientes componentes:

- Un modelo capaz de reconocer los gestos del usuario usando los datos provenientes de un sensor inercial o IMU.
- Un sensor o grupo de sensores inerciales capaz de capturar las características que contienen los gestos del usuario.
- Un algoritmo capaz de aplicar el modelo anterior y sensor para reconocer los gestos del usuario de manera continua y en tiempo real.
- Un algoritmo que permita utilizar las predicciones hechas por el modelo para controlar los movimientos del robot colaborativo.

Teniendo en cuenta los elementos anteriores, los primeros pasos a dar para el desarrollo del sistema consisten en elegir el tipo de modelo a utilizar para la clasificación de los gestos, los gestos que se quieren clasificar y el sensor con el cual obtener los datos sobre los gestos.

3.1.1. Tipo de modelo de clasificación utilizado

La fase inicial del proyecto consiste en elegir el tipo de modelo de clasificación de gestos, así como los gestos que se pretenden clasificar. En el capítulo anterior se comentaron brevemente varias publicaciones de interés en el mundo del reconocimiento de gestos, se observó que en su gran mayoría se empleaban métodos clásicos de Aprendizaje Automático como el SVM o el HMM. Para este trabajo se opta por basar el modelo de clasificación en una red neuronal del tipo Long Short-Term Memory

(LSTM), puesto que presentan una potente capacidad para tratar con datos de serie temporal como los que provienen de los sensores inerciales tratados en este proyecto.

Una red LSTM es un tipo de red neuronal recurrente (RNN) cuya característica principal es la de ser capaces de modelar datos secuenciales y guardar en memoria el estado acerca de la información que ha visto durante un mayor tiempo que una red neuronal recurrente tradicional, de manera similar a la forma que tenemos los humanos de leer o escuchar una frase, se procesa palabra por palabra mientras se recuerda la información del pasado para obtener una representación clara del significado de la frase. Este proceso difícilmente se puede conseguir con una red neuronal densa debido a que no tiene memoria, para procesar una secuencia de datos se tendría que transformar toda la secuencia en un solo vector para poder presentárselo a la red.

En el contexto de este trabajo, el modelo LSTM deberá procesar los datos muestreados de los gestos a reconocer mediante un bucle interno propio de las redes LSTM y RNN, este bucle consiste en leer los datos de cada paso en el tiempo, guardando un estado acerca de la información que ha leído y actualizándolo cuando corresponda. La cantidad de datos a procesar depende de la frecuencia de muestreo de los sensores, a una mayor frecuencia, más cantidad de datos a leer y guardar.

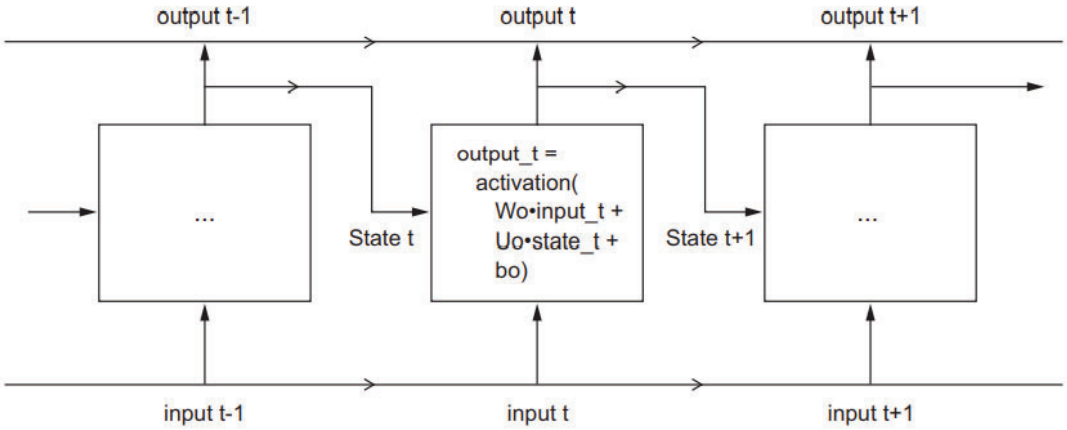

Figura 3.1: Estructura de una RNN simple ([Chollet (2021)])

En la Figura 3.1 se ilustra la estructura de una RNN simple, se puede observar que el funcionamiento se basa en nodos o células que leen un dato de entrada y el estado en un momento en el tiempo y calculan una salida mediante una función de activación con los pesos W, U y b. Por lo tanto, la información del pasado es capaz de transmitirse hacia el presente y futuro a través de la variable de estado, sin embargo, si la secuencia de datos leída es de una gran longitud, es posible que la información del pasado se pierda durante el tiempo, perjudicando el aprendizaje de la red.

Las redes LSTM resuelven el problema descrito anteriormente añadiendo un flujo de datos llamado carry, representado en la Figura 3.2 mediante la letra c. El carry y el peso V forma parte de la función de activación de cada célula y se va actualizando con el paso del tiempo. De esta forma la información del pasado se va conservando en el estado gracias a esta adición.

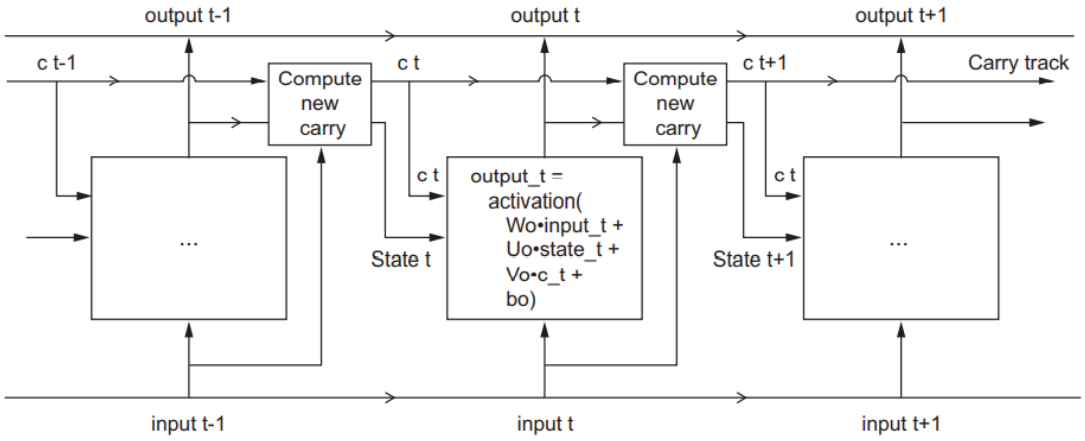

Figura 3.2: Estructura de una red LSTM ([Chollet (2021)])

3.1.2. Elementos a clasificar con el modelo

El sistema de reconocimiento de gestos está orientado a la aplicación con los Cobots, concretamente, el control de un Cobot para desempeñar tareas industriales típicas como el ensamblaje. Por lo tanto, los gestos a clasificar se han seleccionado en base a su simpleza y su facilidad para ser aplicados por usuarios con poca formación o experiencia en el ámbito industrial.

Gesto	Descripción	Etiqueta
Posición neutral o *idle*	Mantener el sensor sin moverse apenas y en posición horizontal	0
Señalar hacia arriba	Mover el sensor para ”apuntar” hacia arriba	1
Señalar hacia abajo	Mover el sensor para ”apuntar” hacia abajo	2
Señalar a la derecha	Mover el sensor para ”apuntar” hacia la derecha del usuario	3
Señalar a la izquierda	Mover el sensor para ”apuntar” hacia la izquierda del usuario	4
Girar en sentido horario	Girar el sensor similar a cuando se ”abre” una puerta	5
Girar en sentido antihorario	Girar el sensor en el sentido opuesto al caso anterior	6

Cuadro 3.1: Gestos elegidos para su clasificación. Las descripciones de los gestos en donde se nombra ”derecha” , ”izquierda” , ”horario” y ”antihorario” se han escrito en base a la perspectiva del usuario que tiene el sensor en la mano y en frente de su cuerpo.

En la Tabla 3.1 se exponen los gestos elegidos para su clasificación y posterior uso en una aplicación dirigida a los Cobots. Dado que se trata de un problema de Clasificación Supervisada, además se incluye la etiqueta que le corresponde a cada gesto. En las Figuras 3.3 y 3.4 se enseña un maniquí sujetando con su mano derecha una esfera roja que representa el sensor, las flechas representan el movimiento que debe

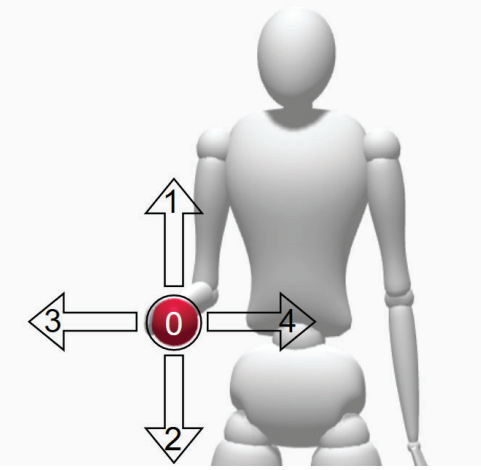

Figura 3.3: Representación gráfica de los gestos 0, 1, 2, 3, 4

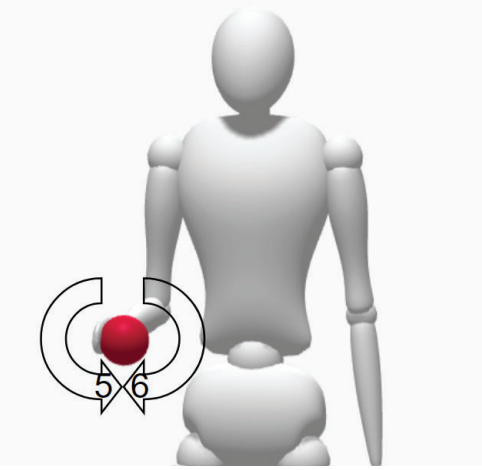

Figura 3.4: Representación gráfica de los gestos 5, 6

seguir su mano para realizar los gestos de la Tabla 3.1.

Para el entrenamiento del modelo LSTM se utiliza Keras, porque ya cuenta con una implementación de red LSTM a la que se le puede ajustar los parámetros y entrenar de forma sencilla.

3.1.3. Sensor inercial utilizado

Una vez elegido el modelo de clasificación y los gestos que se quieren clasificar, el siguiente paso es elegir el método para generar los datos con los que poder entrenar dicho modelo. En el capítulo anterior se comentaron brevemente varias publicaciones en donde se utilizaron sensores sofisticados con una gran precisión, para este proyecto se eligió utilizar un sensor Arduino debido a su facilidad de programación, gran disponibilidad en el mercado y su asequibilidad económica.

Arduino es una plataforma de placas electrónicas de código abierto. Las placas de Arduino son capaces de leer los datos de diversos sensores y mediante una serie de instrucciones al microcontrolador, es posible convertir esos datos en una salida a un monitor para poder ser visualizados. Para poder utilizar el Arduino con un módulo externo, como un sensor, se debe realizar una conexión a través de un cableado y una *protoboard*.

Para este proyecto se optó por utilizar la placa Arduino UNO (Figura 3.5) y el IMU MPU6050 (Figura 3.6), que cuenta con un acelerómetro y giroscopio, ambios triaxiales. El acelerómetro contiene un MEMS (MicroElectroMechanical Systems) que permite medir la aceleración de forma similar a un sistema masa-resorte, concretamente, se trata de un acelerómetro piezoeléctrico cuyo funcionamiento se basa en las descargas eléctricas generadas a partir de la fuerza que se le aplica al material piezoeléctrico. El giroscopio utiliza otro MEMS para medir la velocidad angular a través del efecto Coriolis las especificaciones del módulo se pueden encontrar en la Tabla

3.2. La conexión resultante entre los 2 dispositivos se ilustra en las figuras 3.7 y 3.8.

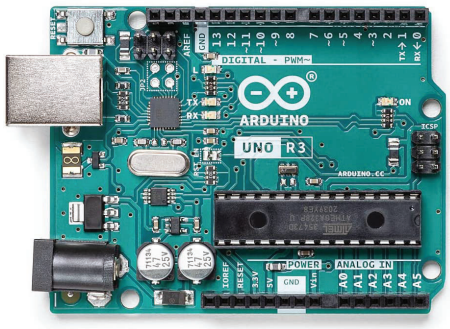

Figura 3.5: Placa Arduino UNO

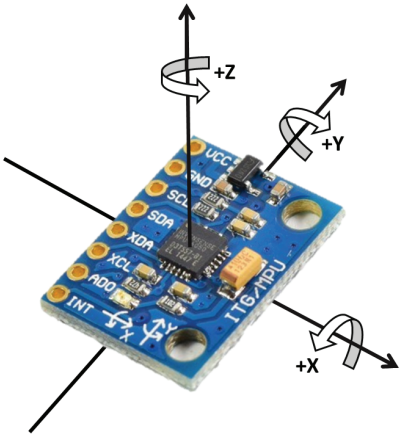

Figura 3.6: Módulo MPU6050 y su orientación en los 3 ejes junto con la polaridad de rotación

MPU6050	Acelerómetro	Giroscopio
Frecuencia de muestreo	1 kHz	8 kHz
Sensitividad	16384 LSB/g	131 LSB/º/seg
Rango máximo	±2 g	±250 º/seg

Cuadro 3.2: Especificaciones del móduo MPU6050.

3.2. Desarrollo del sistema

3.2.1. Programación del sensor

Tras haber identificado a alto nivel los componentes del sistema de reconocimiento de gestos, es necesario programar el componente que se encarga de capturar los datos de aceleración y velocidad angular. Esta fase del proyecto es crucial debido a que la precisión del modelo de reconocimiento depende de la calidad de los datos que se le presenten, los datos deben contener las suficientes características acerca de los

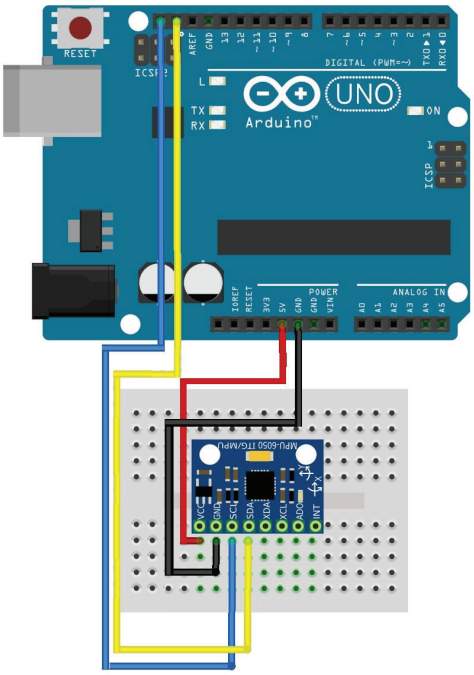

Figura 3.7: Esquema de la conexión entre el Arduino UNO y el MPU6050

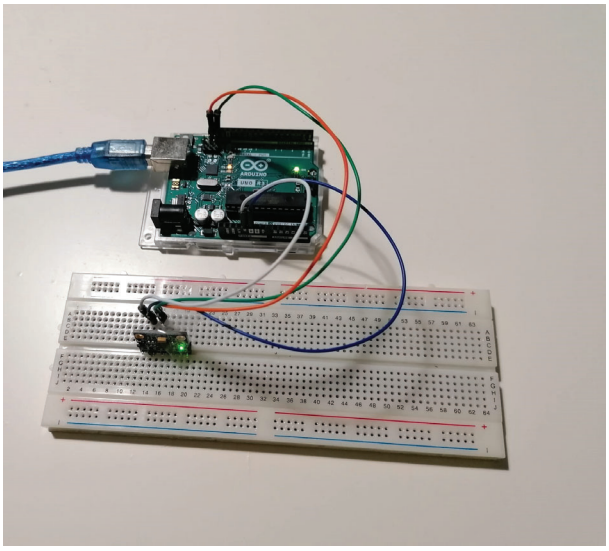

Figura 3.8: Conexión real

gestos para favorecer la precisión del modelo.

Arduino cuenta con un su propio entorno de desarrollo, con el que medante un lenguaje de programación basado en c y c++ es posible programar el dispositivo de una manera sencilla. En el contexto de este trabajo, se programa la placa para que capte los datos recibidos del módulo MPU6050 y los muestre por consola en forma de columnas separadas por comas, para poder guardar los resultados en ficheros de formato csv.
En la Figura 3.9 se ilustra un ejemplo de cómo son los datos obtenidos por el sensor según el programa inicial. En orden de columnas de izquierda a derecha se observan los valores de aceleración en los ejes X, Y, Z, y los valores de velocidad angular en los ejes X, Y, Z.

Tal y como se explica en la Tabla 3.2, los datos son presentados como LSB (Least Significant Bit). Si se quiere convertir las unidades a metros por segundo al cuadrado en el caso del acelerómetro, se debe multiplicar el resultado por 9,81 y dividir entre 16384. En el caso del giroscopio se debe dividir entre 131 para convertir los valores a grados por segundo.

En la Figura 3.10 se observan los datos obtenidos por los sensores tras convertir los valores en metros por segundo al cuadrado para las columnas de aceleración, y grados por segundo para las columnas de velocidad angular. En orden de izquierda a derecha se encuantran los valores de aceleración en los ejex X, Y, Z, y la velocidad angular en los ejes X, Y, Z.

Tal y como se indica en la Tabla 3.1, el modelo debe ser capaz de reconocer ges-

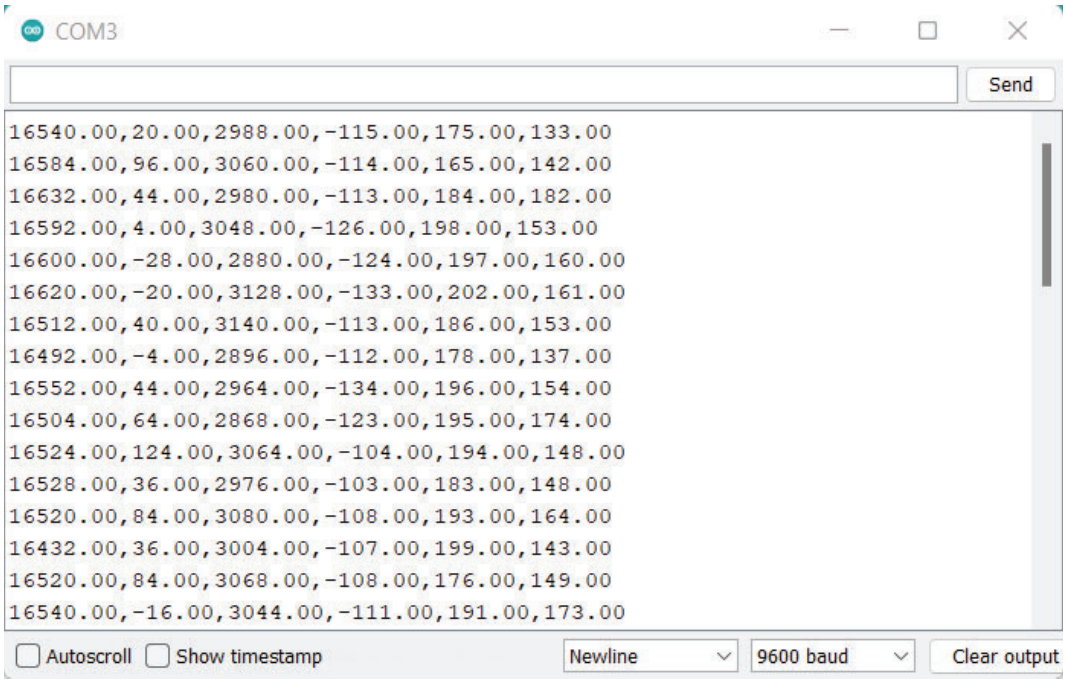

Figura 3.9: Ejemplo de datos recibidos por el MPU6050

tos en donde se incluyen giros pronunciados del sensor (Gestos 5 y 6). A pesar de que estos 2 movimientos presentan características lo suficientemente distinguibles como para que sea posible reconocerlos (se presentan cambios en los valores de la velocidad angular en el eje Y), se decidió aplicar el filtro de Kalman para obtener los datos relativos al ángulo en el que se encuentra posicionado el sensor con respecto a los ejes X e Y, con el objetivo de aumentar la cantidad de información asociada a cada movimiento y favorecer la precisión del clasificador.

El filtro de Kalman es un algoritmo ampliamente usado en procesamiento de señales. Su principal utilidad es la de poder estimar el estado de un sistema a partir de distintas fuentes de datos que contienen ruido. En el contexto de este trabajo, el algoritmo se aplicó para obtener de forma precisa el ángulo en el que se encuentra el sensor respecto de los ejes X e Y, teniendo en cuenta que la posición neutral es la que tiene el sensor cuando se encuntra el chip mirando hacia arriba, como se observa en la Figura 3.6. El ángulo en el que se encuentra el sensor respecto del eje Z no se puede obtener de forma precisa mediante el uso de un acelerómetro y giroscopio, debido a que el módulo no tiene un Norte de referencia sobre el que basarse, por lo tanto el error por defecto que presentan las medidas de los sensores generarían desplazamientos en los ángulos obtenidos, resultando en medidas incorrectas con el paso del tiempo. Para solventar ese problema se recomienda el uso de un magnetómetro, dicha solución fue descartada para este proyecto debido a que los ángulos en X e Y son suficientes para poder clasificar el juego de movimientos elegido.

En la Figura 3.11 se observa la diferencia entre obtener los ángulos en los ejes X e Y usando solamente el giroscopio, y usando el Filtro de Kalman que tiene en cuenta tanto el acelerómetro como el giroscopio. Las líneas de color verde y amarillo son los ángulos obtenidos con Kalman en X e Y, respectivamente, mientras que las de color

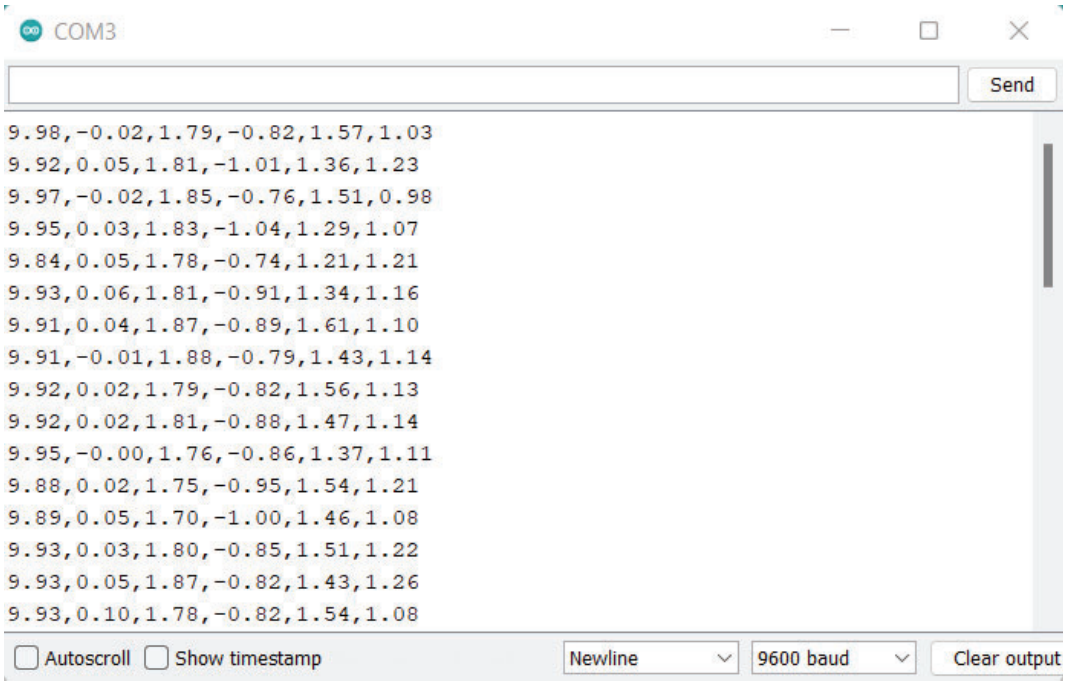

Figura 3.10: Ejemplo de datos recibidos por el MPU6050 tras aplicar las fórmulas de conversión

azul y rojo son los obtenidos únicamente con el giroscopio para los ejes X e Y. La posición del sensor es la misma que la que se observa en la Figura 3.8 durante todo el tiempo de medición, sin embargo, se puede observar cómo con el paso del tiempo se van produciendo errores acumulativos en los valores obtenidos únicamente con el giroscopio (líneas azul y rojo), lo que resulta en los desplazamientos que se observan en la figura. El Filtro de Kalman está integrando el acelerómetro en el cálculo de los ángulos, debido a esto, los valores obtenidos son más precisos que los anteriores, se tiene una referencia sobre la que basarse en los cálculos.

Esta diferencia entre valores es aún más notoria cuando el sensor se encuentra en movimiento, al contrario que en el caso anterior, en donde se encontraba en una posición fija durante todo el tiempo de medición. En la Figura 3.12 se observa un ejemplo gráfico de visualización de los ángulos X e Y obtenidos tanto con la manera tradicional (lineas rojo y azul) como con el Filtro de Kalman (lineas amarillo y verde), al principio del proceso de medición ambos métodos coinciden con los valores obtenidos, pero según avanza el tiempo empiezan a observarse los desplazamientos comentados anteriormente por parte de los ángulos obtenidos con el giroscopio. Teniendo en cuenta que el sistema debe ser capaz de reconocer de forma continua los gestos del usuario, el sistema estará obteniendo de forma continua los datos de los sensores, con lo cual, fue necesario aplicar el Filtro de Kalman para solventar esta problemática.

En la Figura 3.13 se observa que se añadieron 2 columnas adicionales respecto de los casos anteriores (Figuras 3.9 y 3.10), en este caso se trata de los ángulos en los que se encuentra el sensor respecto de los ejes X e Y, y son las últimas 2 columnas

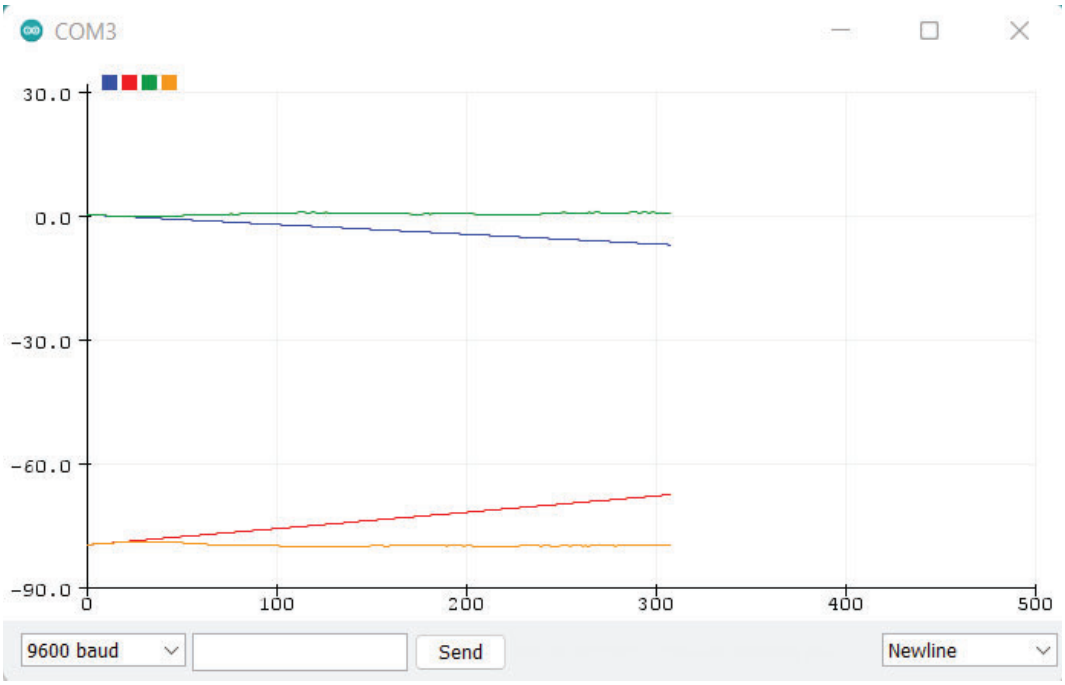

Figura 3.11: Diferencia entre obtener los ángulos en X e Y mediante Kalman y usando solamente el giroscopio

comenzando por la izquierda.

3.2.2. Muestreo de los movimientos

Una vez se ha programado el sensor, la siguiente fase consiste en obtener una cantidad suficiente de ejemplos de cada movimiento para posteriormente, entrenar el modelo LSTM con ellos. Como se explicó en la Tabla 3.2, el acelerómetro y giroscopio presentan sus propias frecuencias de muestreo por defecto. Es importante elegir correctamente la frecuencia a la que se quiere muestrear los datos de los sensores, puesto que a una mayor frecuencia, mayor nivel de detalle se obtendrá de cada movimiento, sin embargo, esto aumentará la cantidad de datos a guardar y tratar, y la fase de entrenamiento del modelo será más extensa. En el caso de que se decida establecer una frecuencia más baja, se acelerará la fase de entrenamiento y los datos no ocuparán tanto espacio como en el caso anterior, pero esto puee causar un peor el rendimiento por parte del modelo debido a la poca cantidad de información disponible de cada ejemplo.

Finalmente, la frecuencia de muestreo elegida de 20Hz, además, se establece que cada gesto se va a medir durante una ventana de 2,5 segundos para asegurar que se está capturando todo el movimiento del usuario. Debido a esta decisión, los datos que se obtienen de cada gesto se encuentran en forma de matrices de dimensión 50 * 8. 50 filas de datos que corresponden a la medición durante 2,5 segundos a una frecuencia de 20hz (2,5 * 20=50), y 8 columnas que corresponden a los datos que se obtienen de los sensores, aceleración en los 3 ejes, velocidad angular en los 3 ejes, ángulo respecto a los ejes X e Y obtenido con el filtro de Kalman.

Se procede a muestrear 40 ejemplos de cada gesto, generando un total de 280 ejem-

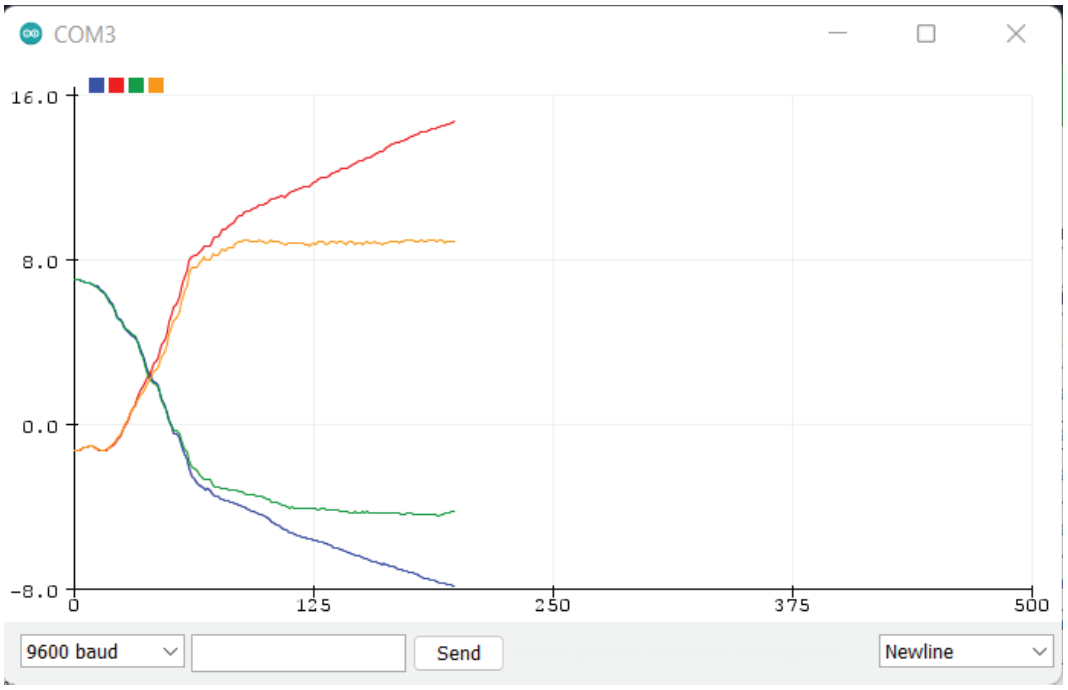

Figura 3.12: Diferencia entre obtener los ángulos usando solo el giroscopio y con el Filtro de Kalman

plos obtenidos manualmente con su respectiva etiqueta que fueron guardados en 2 ficheros por separado. Con el objetivo de aumentar la robustez del modelo, se ejecuta cada gesto a velocidades distintas y con trayectorias ligeramente distintas a las descritas en las Figuras 3.3 y 3.4, resultando en una gran variedad de ejemplos con características únicas. El código del entorno de Arduino se incluye en el Apéndice, su ejecución produce salidas como la comentada en la Figura 3.13. Todo el código escrito para ejecutar las tareas de muestreo con Arduino en esta sección está disponible en el Apéndice A.

3.2.3. Generación de datos adicionales

Como se explicó anteriormente, se generan 280 ejemplos en total de forma manual. Este número resulta bajo si se quiere entrenar un modelo neuronal, sobre todo si se trata de un modelo LSTM. Para solventar este problema se aplica un método considerado estado del arte de *Data Augmentation* para datos de serie temporal ([Iwana y Uchida (2021)]) cuando se trata de clasificación por medio de sensores y redes LSTM.

El método aplicado es el conocido Ruido Gaussiano. El concepto es el mismo que el que se aplica en el tratamiento de imágenes, se trata de generar más ejemplos aplicando una "distorsion" a las matrices asociadas a cada movimiento añadiéndoles un ruido que está definido como una función que sigue una distribución normal cuyos parámetros son 0 y la desviación estándar del conjunto de datos (Función 3.2). Por lo tanto, a la muestra x se le aplica el ruido definido por la Ecuación 3.1 durante toda su duración T.

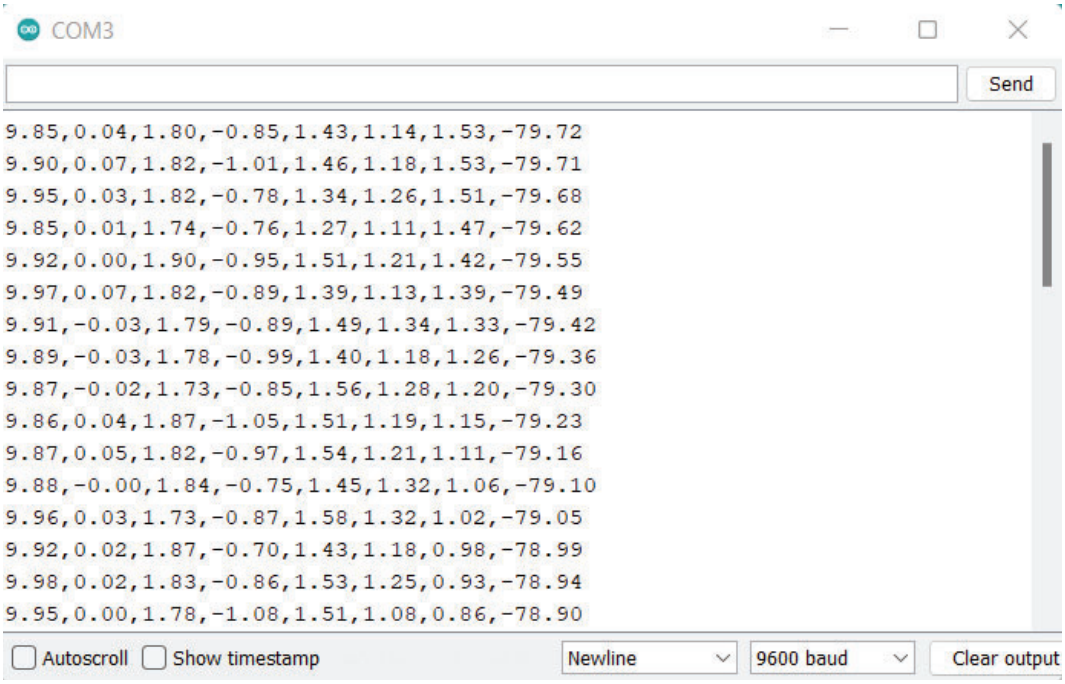

Figura 3.13: Ejemplo de datos recibidos por el MPU6050 tras aplicar las fórmulas de conversión y tras la obtención de los angulos en X e Y con el Filtro de Kalman

$$x' = x_1 + \epsilon_1, \cdots, x_T + \epsilon_T \tag{3.1}$$

Donde ϵ se define como la Función 3.2

$$\epsilon \sim N(0, \sigma^2) \tag{3.2}$$

Con este método se consiguen generar 200000 ejemplos adicionales para el entrenamiento y validación del modelo de clasificación. Tanto los ejemplos nuevos como las etiquetas fueron guardados en sus respectivos ficheros csv. El código con la implementación del Ruido Gaussiano se encuentra disponible en el Apéndice B.

3.2.4. Preprocesado de datos

Para poder entrenar el modelo primero es necesario estructurar el conjunto de datos de forma que la red los pueda procesar. Con Keras se crea un tensor de dimensión 200000 * 50 * 8, lo que equivale a 200000 matrices de 50 * 8, igual que el número de ejemplos que se obtuvieron con el Data Augmentation y que se guardaron en un fichero csv. Después se escribe un script que se encarga de pasar los datos del fichero csv anterior al tensor. Acceder de forma individual a cada movimiento en el fichero csv es sencillo porque todos tienen la misma dimensión (50 * 8), basta con multiplicar la posición de su etiqueta en el vector por 50 y restarle 49. Si se quiere acceder en el csv al primer movimiento, se calcula 0 * 50 + 1 = 1. Con lo cual, el primero movimiento corresponde a las filas 1 hasta 50 del fichero csv. Si se quiere acceder al movimiento 20 se calcula 19 * 50 + 1 = 951. Con lo cual el movimiento nº 20 corresponde a las filas 951 hasta 1000 del fichero csv, y así sucesivamente. Por último, se normalizan

los datos del tensor restándole a cada elemento su media y dividiéndolo entre la varianza (Ecuación 3.3), este método es conocido como Estandarización. Este último paso se realiza para mejorar el aprendizaje de la red, al tratar con datos que contienen valores con rangos distintos entre sí (3 valores de aceleración en m/s$\hat{2}$, 3 valores de velocidad angular en º/seg, 2 valores de ángulos en grados). Al estar utilizando datos normalizados en la fase de entrenamiento, es necesario normalizar los datos durante las pruebas y en los experimentos posteriores.

$$x' = \frac{x - \mu}{\sigma^2} \tag{3.3}$$

Donde μ se define como la Ecuación 3.4 y σ^2 se define como la Ecuación 3.5.

$$\mu = \frac{1}{m} * \sum_{p=1}^{m} X_p \tag{3.4}$$

$$\sigma^2 = \frac{1}{m} * \sum_{p=1}^{m} X_p^2 - \mu^2 \tag{3.5}$$

Una vez generado el tensor de 200000 matrices de 50 * 8, se divide en los siguientes 2 tensores: el primero corresponde a los datos de entrenamiento de la red que se forman de 160000 ejemplos, y el segundo corresponde a los datos de validación que se forman de los 40000 ejemplos restantes. Los 280 ejemplos originales y generados manualmente con el sensor se reservan para la prueba final tras el entrenamiento y validación. El conjunto de datos utilizado está descrito en la Figura 3.14.
Todo el código escrito para ejecutar las tareas descritas en está subsección está disponible en el Apéndice C.1 y C.2.1.

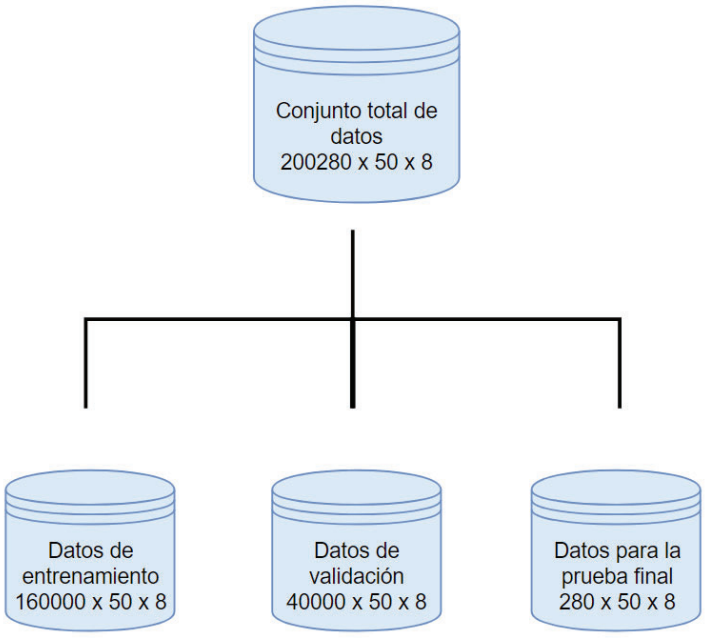

Figura 3.14: Conjunto de datos usado para el entrenamiento y pruebas del modelo

3.2.5. Arquitectura del modelo

Tras generar los datos de los gestos, se procede a entrenar el modelo con diversas arquitecturas LSTM, el modelo que presentó un mayor rendimiento se describe a continuación, mientras que en el capítulo de Resultados se detalla más acerca de su precisión durante el entrenamiento y durante la validación.

El modelo neuronal de clasificación de gestos consiste de una primera capa LSTM de 64 células que es la que recibe como entrada los datos obtenidos de los sensores (matrices de tamaño 50 * 8) y el vector con las etiquetas correspondientes; una segunda capa LSTM de 32 células; una tercera capa Densa de 32 neuronas y una última capa Densa con el mismo número de neuronas que las clases que se tienen en cuenta, es decir 7. Todas las capas utilizan la función de activación Relu salvo por la última que es una Softmax porque se trata de un problema de clasificación. El optimizador que se usó fue el Adam y con una tasa de apredizaje de 1e-4, con un Decay de 1e-5 para evitar el *overfitting*. La función de pérdida utilizada es la de *sparse_categorical_crossentropy*, debido a que se trata de un problema de clasificación y porque las etiquetas no están codificadas con el método *One-Hot*, sino que se tratan de números enteros (ver Tabla 3.1 y Figura 3.15). El modelo descrito con las estructuras que recibe como entrada y las salidas se ilustran en la Figura 3.15. Todo el código escrito para definir la arquitectura del modelo está disponible en el Apéndice C.2.2.

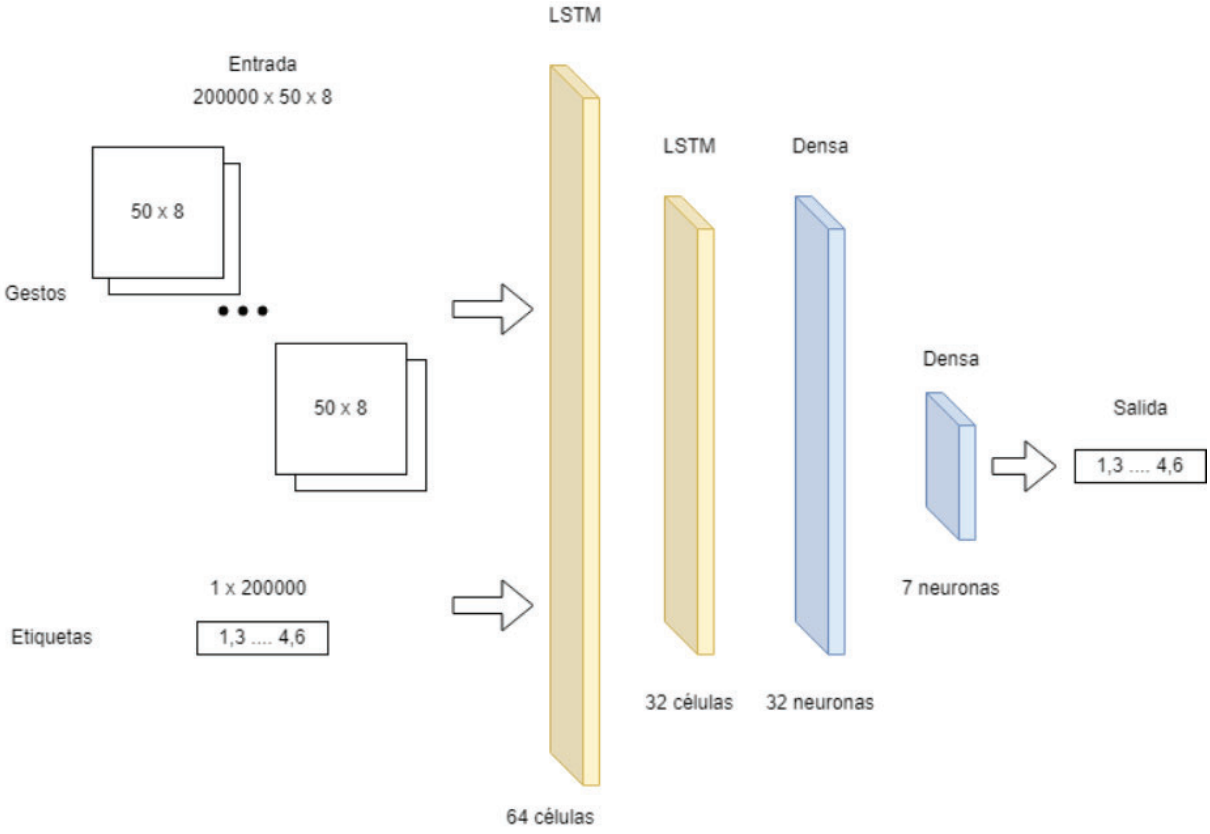

Figura 3.15: Arquitectura del modelo de reconocimiento de gestos

3.2.6. Reconocimiento continuo de gestos

Como se ha explicado anteriormente, se ha elegido el modelo neuronal Long Short-Term Memory (LSTM) debido a su gran utilidad para modelar datos de serie temporal comparado a un red neuronal recurrente tradicional. Esto se debe a que tienen mayor capacidad para retener en memoria datos observados durante un gran número de pasos. Dado que se está tratando con matrices que representan patrones de movimiento, es fundamental que el modelo pueda guardar durante el tiempo que le toma analizar una matriz entera el estado del movimiento que trata de clasificar antes asignarle la etiqueta correspondiente.

Como este sistema está orientado a la aplicación con Cobots, se debe poder aplicar el modelo de clasificación de una forma contínua y en tiempo real, para poder controlar los movimientos del robot mediante los gestos que realiza el usuario con el sensor. Este problema se resuelve de una forma similar a la vista en varias publicaciones comentadas en el capítulo anterior. Se utiliza una ventana deslizante de tamaño 50 * 8, igual a la dimensión de las matrices asociadas a los gestos a reconocer, dicha ventana va automáticamente capturando los datos que se captan de los sensores y se va desplazando en tandas de 10, por lo tanto, existe un porcentaje de solapamiento del 80 % debido a que en un instante T se va a tener una ventana cuyas 40 primeras filas de su matriz serán coincidentes con las 40 últimas filas del instante anterior T - 1, y las 10 filas restantes son nuevas. Este alto porcentaje de solapamiento se elige para poder asegurar un reconocimiento contínuo de gestos por parte del modelo, si se hubiera elegido un bajo porcentaje se producirían retrasos por parte del sistema durante el reconocimiento, ya que habría que esperar un mayor tiempo para que la ventana se llene de datos y presentárselos al modelo, teniendo en cuenta que la ventana es de tamaño 50 * 8 (2,5 segundos midiendo con los sensores a una frecuencia de 20 hZ), un porcentaje de solapamiento tan bajo como 20 % producirían tiempos de espera de hasta 2 segundos.

En las Figura 3.16 se puede observar a nivel gráfico el proceso que sigue la ventana. Se empieza capturando los primeros datos hasta que la ventana se encuentre llena y enviar esos datos al modelo para clasificarlos, para posteriormente, ”deslizar” la ventana de tal forma que se encuentre solapada en un 80 % con la iteración anterior, el 20 % de datos restantes son nuevos para la ventana en ese momento. Tras deslizar la ventana se vuelve a enviar los datos obtenidos al modelo para que los clasifique, y así sucesivamente.

Para evitar la aparición de falsos positivos durante el proceso de reconocimiento continuo, se introducen las siguientes condiciones a cumplir: en primer lugar, se establece que una predicción hecha por el módelo sólo se tendrá en cuenta si previamente el sensor se encontraba en posición neutral, es decir, la predicción será válida si en el instante anterior la ventana contenía un gesto cuya clase es 0 (ver Tabla 3.1), de esta forma se asegura que el gesto a ser usado para controlar al robot sea específicamente el que el usuario le indique mediante este método. En segundo lugar, se establece que el gesto a tener en cuenta será el que se haya presentado con más frecuencia en las 3 iteraciones siguientes a la primera en donde se detectó el gesto 0. Esta segunda condición sirve para garantizar que el gesto que se reconoce sea exactamente el que el usuario realiza, puesto que, teniendo en cuenta que se está trabajando con ventanas que contienen 80 % de la información de la iteración anterior y 20 % de información

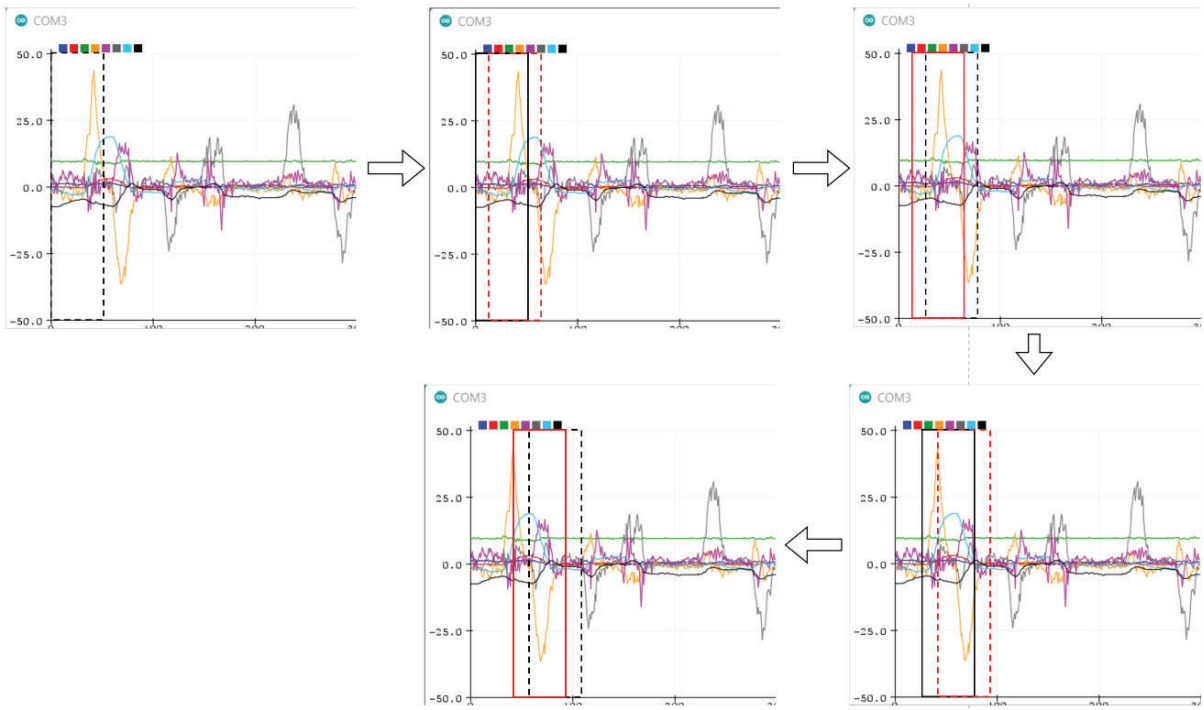

Figura 3.16: Funcionamiento de la ventana deslizante con un 80 % de solapamiento

nueva, es posible que existan errores de clasificación cuando se le presente al modelo una matriz que contiene únicamente 20 % de información que pertenece al gesto que se quiere clasificar.

Para implementar las condiciones expuestas anteriormente se utiliza una estructura FIFO (*First In First Out*) de tamaño 4, es decir una Cola de tamaño 4. En dicha Cola se guardan las predicciones hechas por el modelo y la ventana deslizante, y cuando la cola alcanza la capacidad máxima de predicciones, el primer elemento en salir es el que entró el primero. Se especifica que si sólamente existe una etiqueta 0 en la cola y esa etiqueta se trata del primer elemento de la estructura, entonces se observarán el resto de las etiquetas para encontrar el elemento con mayor frecuencia y tenerlo en cuenta como predicción. En la Figura 3.17 se enseñan varios ejemplos de predicciones teniendo en cuenta las condiciones especificadas anteriormente.

Por último, para poder permitir al usuario volver a la posición neutral tras el reconocimiento de un gesto, sin causar que el modelo reconozca el movimiento cuando no estaba intencionado, se especifica que tras haber utilizado la cola para reconocer un gesto distinto de 0, el sistema se pause por 2 segundos, aproximadamente.

Todo el código escrito para ejecutar las tareas descritas en está subsección está disponible en el Apéndice D.

3.2.7. Control de robot colaborativo mediante gestos

En este proyecto se utiliza el entorno CoppeliaSim para la simulación de robots. CoppeliaSim es un simulador que cuenta con su propio entorno de desarrollo, en donde mediante la creación de "escenas" es posible simular robots y su interacción con

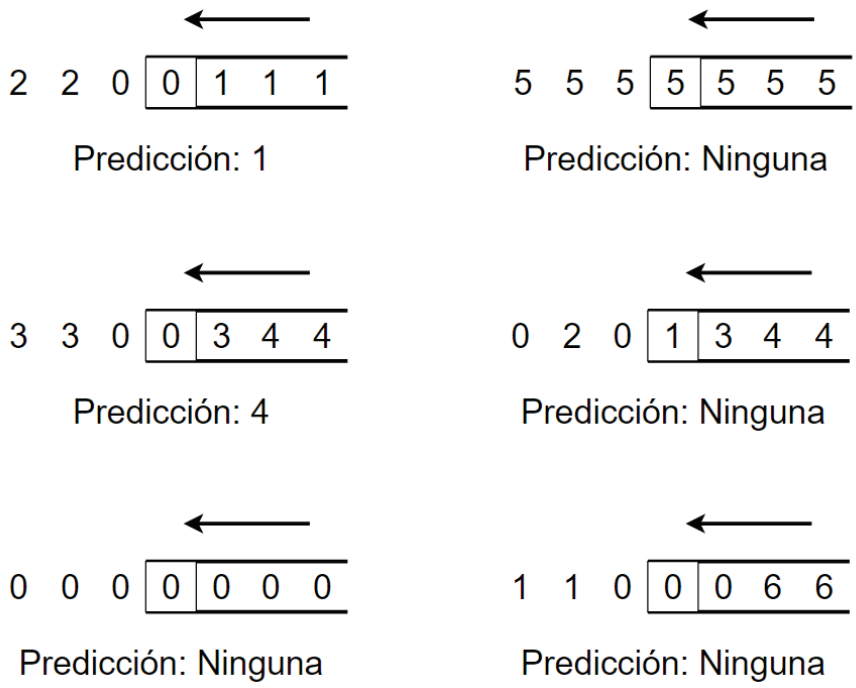

Figura 3.17: Predicciones hechas por el sistema formado por la ventana deslizante y la Cola de etiquetas según las condiciones establecidas

otros objetos de la escena a través de scripts embebidos que pueden estar escritos en C/C++, Python, Java, Lua, Matlab u Octave. El robot elegido es el UR3e de Universal Robots, se trata de un robot industrial colaborativo con aspecto de brazo que está formado por 6 articulaciones, todas con una capacidad de rotación de 360 grados, lo que lo hace útil en tareas de ensamblaje. Dicho modelo tiene una implementación en el CoppeliaSim, por lo que fue posible añadir el robot a la escena sin necesidad de instalar librerías externas.

Por defecto, el UR3e no cuenta con una herramienta para coger objetos como una pinza o un succionador, por lo tanto, es necesario añadir a la escena una pinza compatible con el modelo. En este proyecto se usa la pinza Robotiq 2F-85 debido a su fácil manejo en el entorno de CoppeliaSim y aplicabilidad en tareas típicas de ensamblaje o recoger y colocar objetos.

Finalmente, tras obtener un modelo de clasificación continua de gestos y haber elegido el robot con el que trabajar, es necesario realizar un *mapping* o una asociación entre la etiqueta de cada gesto a un movimiento de un robot, de tal manera que cada acción del robot se pueda ejecutar con uno de los gestos aprendidos por el modelo, en la Tabla 3.3 se ve reflejado esta asociación. Como se puede observar, se diseña el sistema de reconocimiento y las acciones del robot de tal manera que el robot se oriente hacia donde el usuario "apunte" con el sensor para poder recoger y dejar objetos. Se diseña de esta forma para conseguir una mayor facilidad de uso para el usuario y que sea capaz de controlar el robot de una manera intuitiva.

Los movimientos del brazo se definen mediante una serie de configuraciones sobre las que se va navegando según el gesto que se reconoce. En cada configuración, me-

diante un vector de tamaño 6 se define la posición (en grados respecto a la posición inicial) que debe adoptar cada articulación del robot para llegar al objetivo deseado, este proceso se conoce en Robótica como Cinemática Directa y es fácilmente aplicable en el simulador mediante la API interna que posee. En la Tabla 3.4 se listan las distintas configuraciones que se han definido para el UR3e. El flujo de trabajo y de información transmitida se puede visualizar en la Figura 3.19, en el capítulo de Resultados se describe cómo se utilizó el sistema en un experimento. El código con la implementación del sistema se encuentra en el Apendice D.

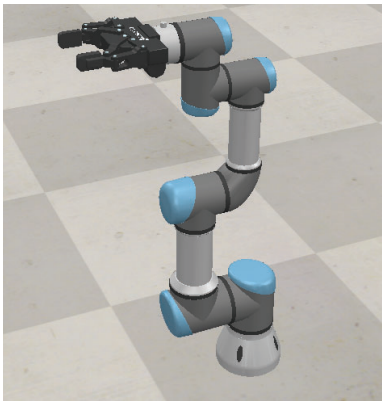

Figura 3.18: Visualización del robot UR3 con pinza (o *gripper*) desde el simulador CoppeliaSim

Etiqueta del gesto	Acción del robot
0	No realiza ninguna acción, dicha etiqueta sirve para evitar la aparición de falsos positivos, como se explicó en la Subsección 3.2.6
1	Subir el brazo en caso de que se encuentre abajo
2	Bajar el brazo en caso de que se encuentre arriba
3	Orientar el brazo una posición a la derecha
4	Orientar el brazo una posición a la izquierda
5	Cerrar la pinza en caso de que se encuentre abierta
6	Abrir la pinza en caso de que se encuentre cerrada

Cuadro 3.3: Controles del robot UR3e mediante los gestos aprendidos por el modelo

Posición del brazo	Configuración de las articulaciones	Identificador
Brazo arriba y orientado al centro	[90, -90, 0, -90, -90, 0]	qf
Brazo abajo y orientado al centro	[90, 0, 0, -90, -90, 0]	qfb
Brazo arriba y orientado a la izquierda del usuario	[50, -90, 0, -90, -90, 0]	qd
Brazo abajo y orientado a la izquierda del usuario	[50, 0, 0, -90, -90, 0]	qdb
Brazo arriba y orientado a la derecha del usuario	[130, -90, 0, -90, -90, 0]	qi
Brazo abajo y orientado a la derecha usuario	[130, 0, 0, -90, -90, 0]	qib

Cuadro 3.4: Configuraciones del brazo teniendo en cuenta que el usuario se encuentra en frente del robot

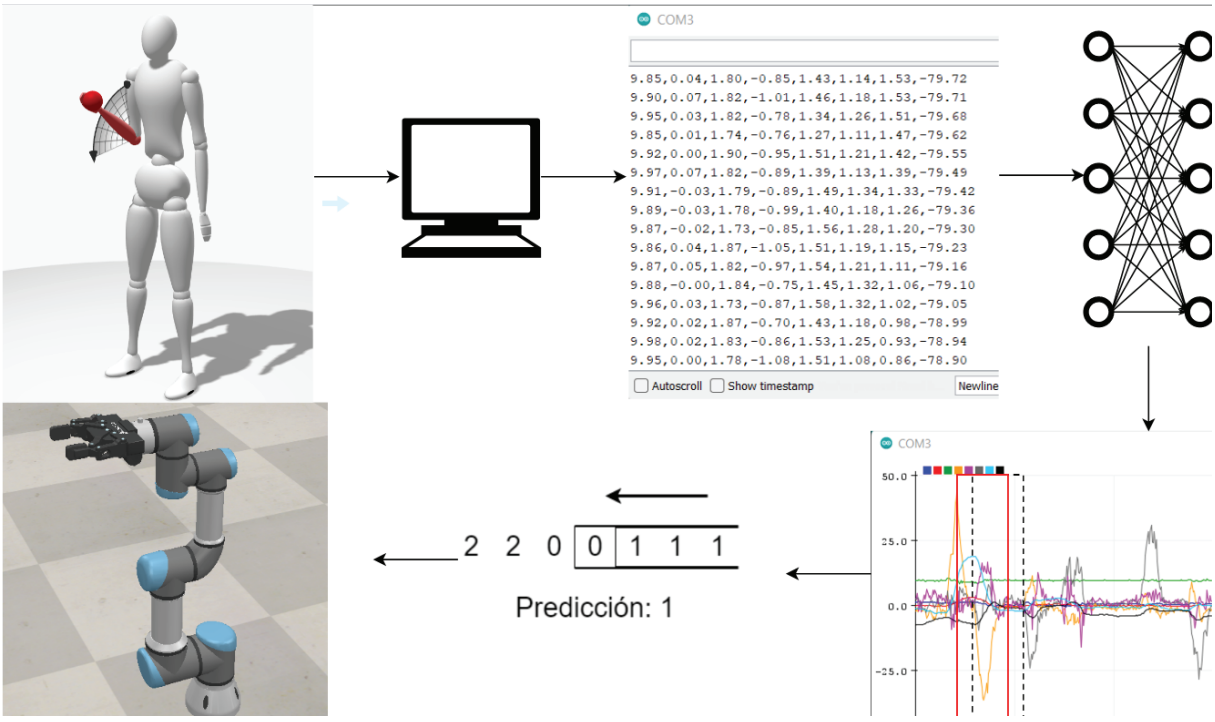

Figura 3.19: Flujo de trabajo del experimento

Capítulo 4

Resultados

En este capítulo se procederá a analizar los resultados obtenidos tanto en el entrenamiento del modelo como en el experimento.

4.1. Modelo de reconocimiento

4.1.1. Entrenamiento del modelo y pruebas

Tras la generación del modelo y el preprocesado de datos se procede a entrenar el modelo con los 160000 ejemplos anteriores y como datos de validación se usan los 40000 ejemplos restantes. Se especifica que se realizara el entrenamiento únicamente por 4 epochs y tras numerosas pruebas se observa que el modelo es capaz de llegar a un Accuracy de 100 % en esas 4 epochs y con un *Loss* aceptable tanto en la fase de entrenamiento como en la de validación (Figuras 4.5 y 4.6). Después del entrenamiento del modelo se procede a hacer una prueba final con los 280 ejemplos que se generaron al principio de forma manual con el sensor. Con el método *evaluate* de Keras se evalúa el modelo con este dataset nuevo y se observa que el Accuracy también es del 100 % y con una pérdida o *Loss* aceptable (Figura 4.7).

Como se explicó en el capítulo anterior, el entrenamiento del modelo se lleva a cabo con los datos normalizados por el método de Estandarización, esto se hace debido a que por el contrario, el proceso de entrenamiento se volvería más ineficiente al estar tratando con datos que se mueven en rangos distintos y con unidades de medición distintos. En las Figuras 4.1, 4.2, 4.5 y 4.6 se observa la diferencia de rendimiento que provoca la normalización de datos, en donde *loss* y *val loss* se refiere al valor devuelto por la función de pérdida con los datos de entrenamiento y validación, respectivamente. *Accuracy* y *val accuracy* se refieren a la precisión del modelo con los datos de entrenamiento y validación, respectivamente.

Las redes RNN y LSTM presentan una gran vulnerabilidad ante el problema del *Exploding gradient*, esto se debe a que este tipo de redes se utilizan en situaciones donde se requiere memoria a corto plazo y los pesos aprendidos pueden explotar fácilmente durante el entrenamiento, puesto que es fácil cometer errores durante el entrenamiento y acumular gradientes altos. Cuando se acumulan errores que producen gradientes altos se producen grandes cambios en los pesos aprendidos, resultando en un aprendizaje pobre y valores de Loss excesivamente altos. Para evitar este problema

no se debe elegir una tasa de aprendizaje o *learning rate* excesivamente alta, para contribuir a que los cambios que se producen en los pesos se mantengan pequeños y controlados. En las Figuras 4.3 y 4.4 se observa un ejemplo de esta problemática, para este entrenamiento se utiliza una tasa de aprendizaje de 5e-3 , y a pesar de utilizar datos normalizados el rendimiento no se le acerca al del modelo final con una tasa de aprendizaje de 1e-4.

Cabe notar otra particularidad de las redes RNN y LSTM, y es que al estar tratando con datos de serie temporal, evidentemente las epochs son bastante más duraderas que las de una red neuronal habitual, cada ejemplo se define por los datos de una matriz de tamaño 50*8 en lugar de un vector con los valores de cada atributo. Por eso el número de epochs de entrenamiento no puede ser tan alto como las que se le especificarían a una red habitual. Esto es un ejemplo de la importancia que tiene el presentarle a la red un conjunto de datos con la suficiente calidad para favorecer su capacidad de generalización y ahorrar en tiempo de entrenamiento. Además, en la gráficas incluso se puede observar cómo va evolucionando el Accuracy y Loss de la red durante la ejecución de una epoch, por eso se observa en el eje de las abcisas como se introducen decimales en las epochs. Por ejemplo en la segunda epoch (en la gráfica es la epoch 1.0) se observa un 1,5 , refiriéndose a que en ese punto del entrenamiento, la ejecución de la epoch no se había completado en su totalidad, de ahí la aparición de decimales.

El código de entrenamiento y validación de la red está disponible en el Apéndice C.2.2 , C.2.3 y C.3.

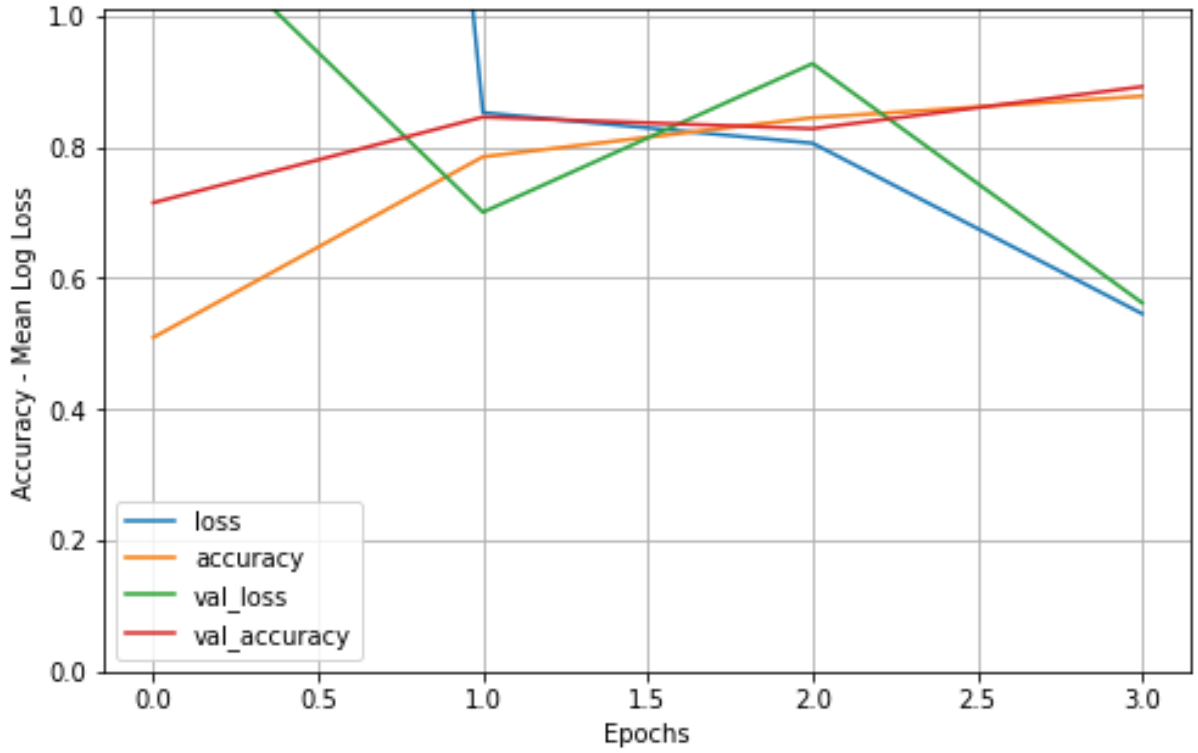

Figura 4.1: Representación gráfica del rendimiento del modelo durante las epochs con datos no normalizados

```
Epoch 1/4
5000/5000 [==============================] - 94s 19ms/step - loss: 5.9623 - accuracy: 0.5091 - val_loss: 1.1870 - val_accuracy: 0.7146
Epoch 2/4
5000/5000 [==============================] - 94s 19ms/step - loss: 0.8519 - accuracy: 0.7846 - val_loss: 0.7000 - val_accuracy: 0.8452
Epoch 3/4
5000/5000 [==============================] - 91s 18ms/step - loss: 0.8054 - accuracy: 0.8441 - val_loss: 0.9266 - val_accuracy: 0.8274
Epoch 4/4
5000/5000 [==============================] - 91s 18ms/step - loss: 0.5453 - accuracy: 0.8770 - val_loss: 0.5619 - val_accuracy: 0.8915
```

Figura 4.2: Representación numérica del rendimiento del modelo durante las epochs con datos no normalizados

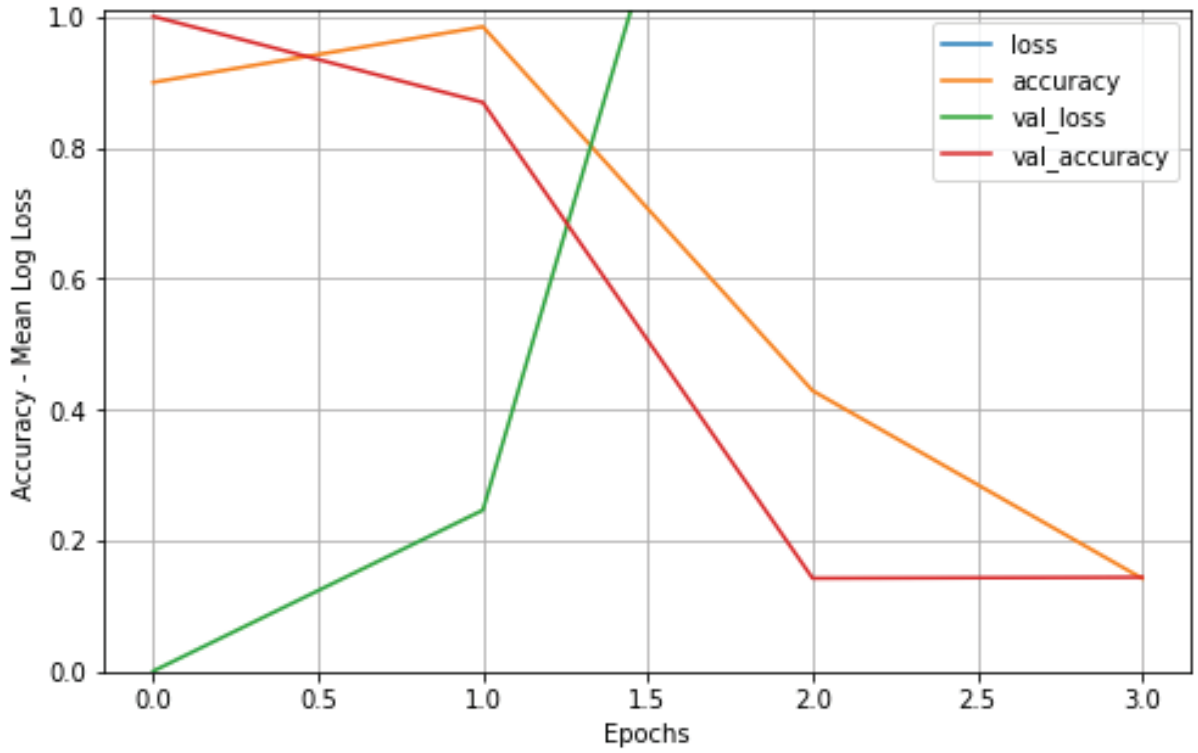

Figura 4.3: Gráfica del resultado del entrenamiento con una tasa de aprendizaje más alta

4.2. Experimento en entorno simulado

4.2.1. Resultados del experimento realizado

Como se ha descrito en el capítulo anterior, la escena del experimento consiste en una aplicación de recoger y dejar objetos en donde el usuario deberá manejar el robot para colocar los objetos en su lugar correspondiente. La escena donde se realizó el experimento es la siguiente: se trata de una situación en donde el robot se encuentra en frente de una cinta transportadora, dicha cinta se encarga de mover objetos por el entorno y se detiene cuando uno de los objetos que lleva se encuentra en frente del robot, este comportamiento se logró mediante el uso de un sensor de proximidad. Esto permite al usuario del sensor realizar los gestos necesarios para lograr coger el objeto controlando los movimientos del robot, y dejar el objeto en uno de los cuencos que se encuentra a su lado. Dependiendo del color del objeto se debe colocar el objeto

```
Epoch 1/4
5000/5000 [==============================] - 87s 17ms/step - loss: 7.1280 - accuracy: 0.8995 - val_loss: 6.9168e-04 - val_accuracy: 1.0000
Epoch 2/4
5000/5000 [==============================] - 86s 17ms/step - loss: 2.8738 - accuracy: 0.9843 - val_loss: 0.2464 - val_accuracy: 0.8688
Epoch 3/4
5000/5000 [==============================] - 87s 17ms/step - loss: 9630.6621 - accuracy: 0.4288 - val_loss: 1.9465 - val_accuracy: 0.1425
Epoch 4/4
5000/5000 [==============================] - 86s 17ms/step - loss: 1.9464 - accuracy: 0.1421 - val_loss: 1.9464 - val_accuracy: 0.1440
```

Figura 4.4: Representación numérica del resultado del entrenamiento con una tasa de aprendizaje más alta

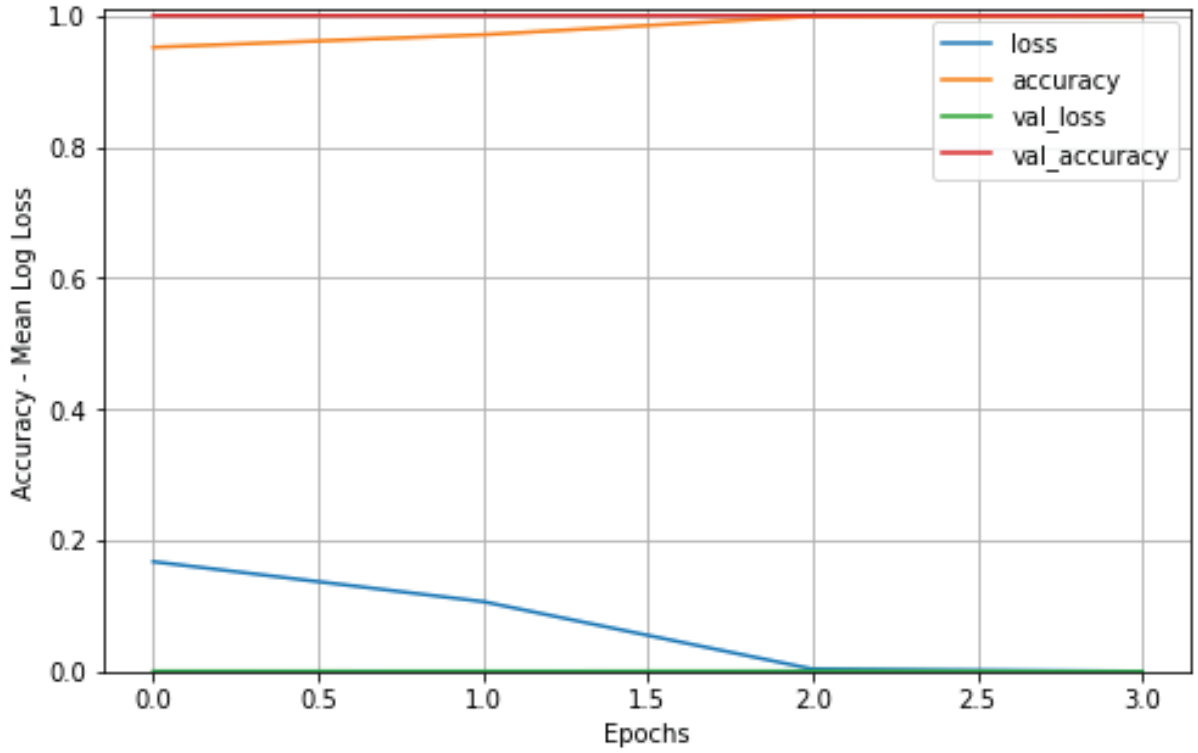

Figura 4.5: Representación gráfica del rendimiento del modelo durante las epochs con datos normalizados

en el cuenco correcto, los objetos azules deben ir en el cuenco que se encuentra en la zona azul, y los objetos rojos deben ir al cuanco de la zona roja. La escena descrita se puede observar en la Figura 4.8.

Teniendo en cuenta las distintas configuraciones del robot y maneras de navegar por ellas mediante los gestos aprendidos por el modelo, la forma de completar la tarea descrita anteriormente se puede describir mediante los siguientes conjuntos de instrucciones: El robot comienza estando orientado hacia la cinta con los objetos y con la pinza cerrada, por lo tanto la primera acción a realizar es abrir la pinza mediante el gesto 6 4.9. El primer objeto a recoger es un objeto azul, por el tanto se debe transportar a la zona azul mediante el conjunto de instrucciones de la Tabla 4.1, en las Figuras 4.10, 4.11, 4.12, 4.13, 4.14, 4.15, 4.16, 4.17 se puede observar la secuencia descrita. Es interesante notar como en dichas imágenes se observa el funcionamiento del sistema de reconocimiento de gestos en la salida por consola. Como se explicó en el capítulo anterior, la ventana deslizante reconoce los datos que

```
Epoch 1/4
5000/5000 [==============================] - 109s 21ms/step - loss: 0.1674 - accuracy: 0.9516 - val_loss: 5.4386e-04 - val_accuracy: 0.9999
Epoch 2/4
5000/5000 [==============================] - 103s 21ms/step - loss: 0.1065 - accuracy: 0.9707 - val_loss: 4.0239e-04 - val_accuracy: 1.0000
Epoch 3/4
5000/5000 [==============================] - 103s 21ms/step - loss: 0.0037 - accuracy: 0.9990 - val_loss: 5.9235e-04 - val_accuracy: 1.0000
Epoch 4/4
5000/5000 [==============================] - 103s 21ms/step - loss: 2.5012e-05 - accuracy: 1.0000 - val_loss: 1.0160e-06 - val_accuracy: 1.0000
```

Figura 4.6: Representación numérica del rendimiento del modelo durante las epochs con datos normalizados

```
model.evaluate(x_Ftest, y_Ftest)    #Evaluar el modelo
✓ 0.3s

9/9 [==============================] - 0s 7ms/step - loss: 2.9256e-06 - accuracy: 1.0000
```

Figura 4.7: Resultado de la evaluación del modelo

va recibiendo con el tiempo, mientras que la estructura *FIFO* o cola se encarga de establecer unas condiciones que permiten al usuario controlar al robot evitando falsos positivos. Por ejemplo en la Figura 4.9 se observa como los últimos elementos que contiene la cola son las etiquetas [0,3,6,6], por lo tanto el gesto que se tiene en cuenta para el robot es el 6, debido a que el sensor se encontraba en la posición neutral (el primer elemento de la cola es el gesto 0) y la etiqueta que más se repite es la número 6, por lo tanto el mensaje que se muestra por consola es la de "Pred: 6". En el caso descrito se demuestra que el algoritmo cumple su función de evitar falsos positivos, ya que a pesar de que la cola contiene una etiqueta de un gesto que no realizó el usuario (etiqueta 3), el robot responde correctamente. La razón por la que apareció una etiqueta errónea durante ese reconocimento es debido al funcionamiento de la ventana deslizante, como se explicó en el capítulo anterior, existe un 80 % de solapamiento entre ventanas, por lo tanto, después de reconocer el gesto 0 y pasar a reconocer el siguiente, la ventana contiene 80 % de la información de la iteración anterior mientras que el 20 % restante pertenece a la información nueva que ha llegado en ese momento. Evidentemente, un 20 % es insuficiente para reconocer el gesto con una fiabilidad aceptable, es por eso que apareció una etiqueta errónea en la Figura 4.9, sin embargo, al continuar reconociendo en el tiempo se observa como las 2 siguientes etiquetas son las correctas porque la ventana sí obtuvo la información suficiente en esas 2 iteraciones.

El siguiente objeto es uno rojo por lo tanto se debe aplicar de forma análoga al caso anterior el conjunto de intrucciones de la Tabla 4.2, por último, se vuelve a aplicar las instrucciones 4.1, al volver a tratar con un objeto azul.

Secuencia de intrucciones	Gesto a realizar	Identificador de la configuración deseada
Bajar el brazo a la altura del primer cubo azul	2	qfb
Cerrar la pinza para coger el objeto	5	qfb
Subir el brazo	1	qf
Orientar el brazo hacia la zona azul	3	qi
Bajar el brazo a la altura del cuenco	2	qib
Abrir la pinza para dejar el cubo azul	6	qib
Subir el brazo	1	qi
Orientar el brazo hacia el centro	4	qf

Cuadro 4.1: Instrucciones a seguir para colocar el primer cubo en el cuenco de la zona azul

Secuencia de intrucciones	Gesto a realizar	Identificador de la configuración deseada
Bajar el brazo a la altura del cubo rojo	2	qfb
Cerrar la pinza para coger el objeto	5	qfb
Subir el brazo	1	qf
Orientar el brazo hacia la zona roja	4	qd
Bajar el brazo a la altura del cuenco	2	qdb
Abrir la pinza para dejar el cubo rojo	6	qdb
Subir el brazo	1	qd
Orientar el brazo hacia el centro	3	qf

Cuadro 4.2: Instrucciones a seguir para colocar el cubo rojo en el cuenco de la zona roja

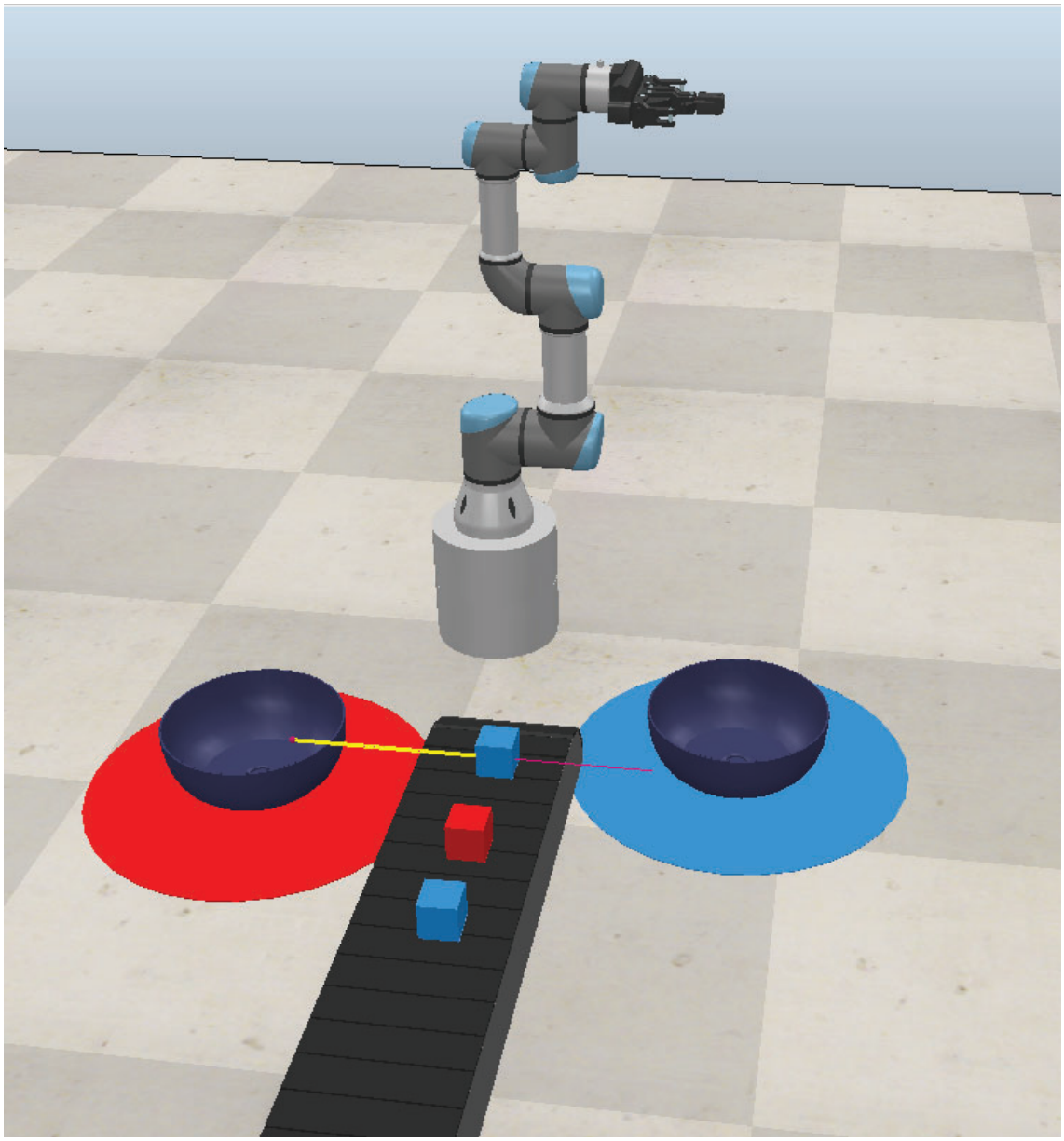

Figura 4.8: Escena del experimento

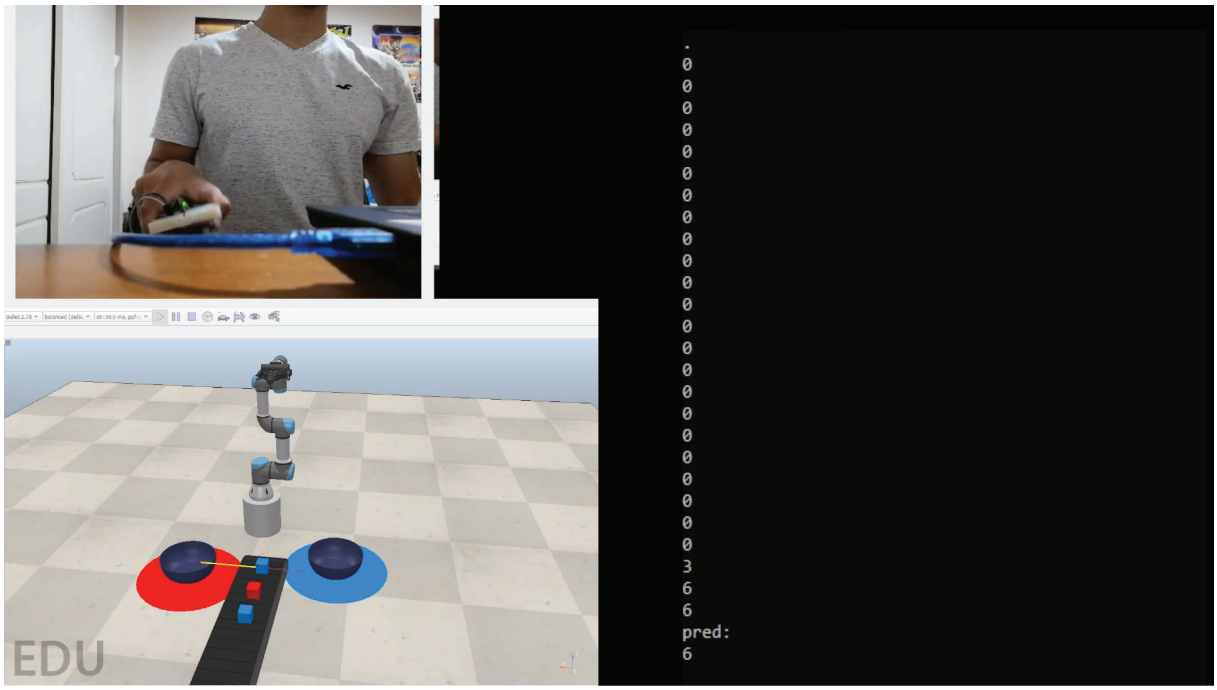

Figura 4.9: El usuario realiza el gesto 6 para abrir la pinza. En la salida por consola se observa como el sistema reconoce de forma continua los gestos hasta que se cumplen la condiciones para tener en cuenta el gesto y poder controlar el robot

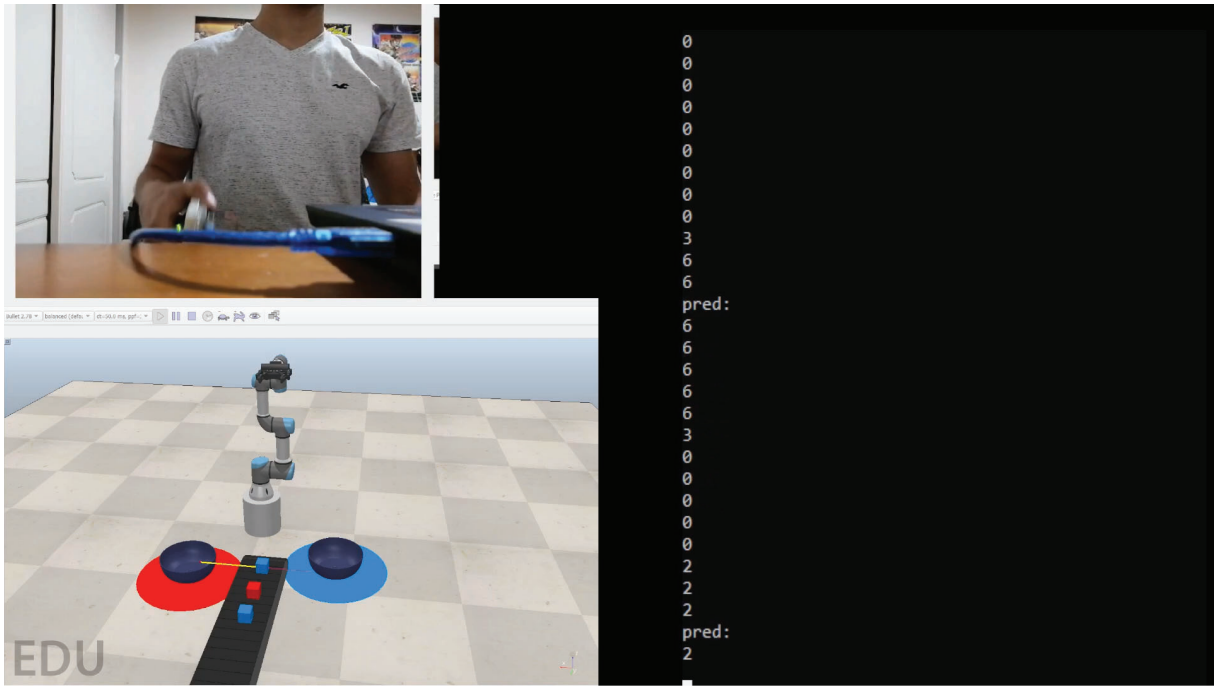

Figura 4.10: El usuario realiza el gesto 2 para bajar el brazo. En la salida por consola se observa como el sistema reconoce de forma continua los gestos hasta que se cumplen la condiciones para tener en cuenta el gesto y poder controlar el robot

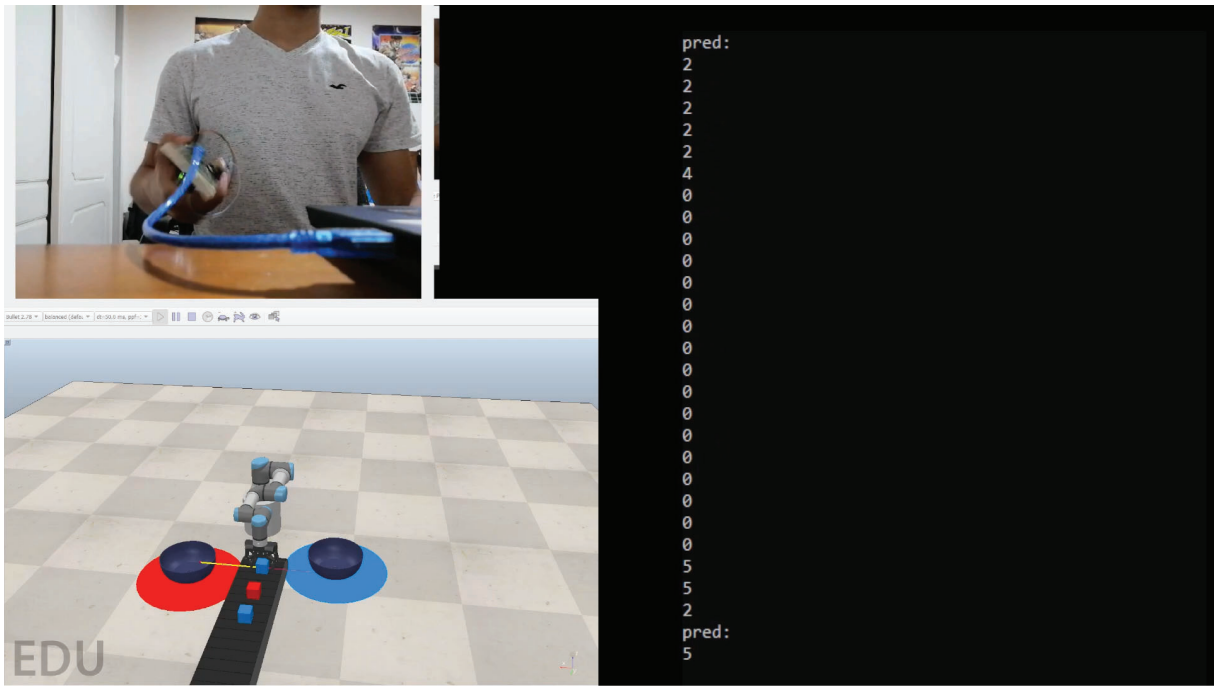

Figura 4.11: El usuario realiza el gesto 5 para cerrar la pinza y coger el objeto. En la salida por consola se observa como el sistema reconoce de forma continua los gestos hasta que se cumplen la condiciones para tener en cuenta el gesto y poder controlar el robot

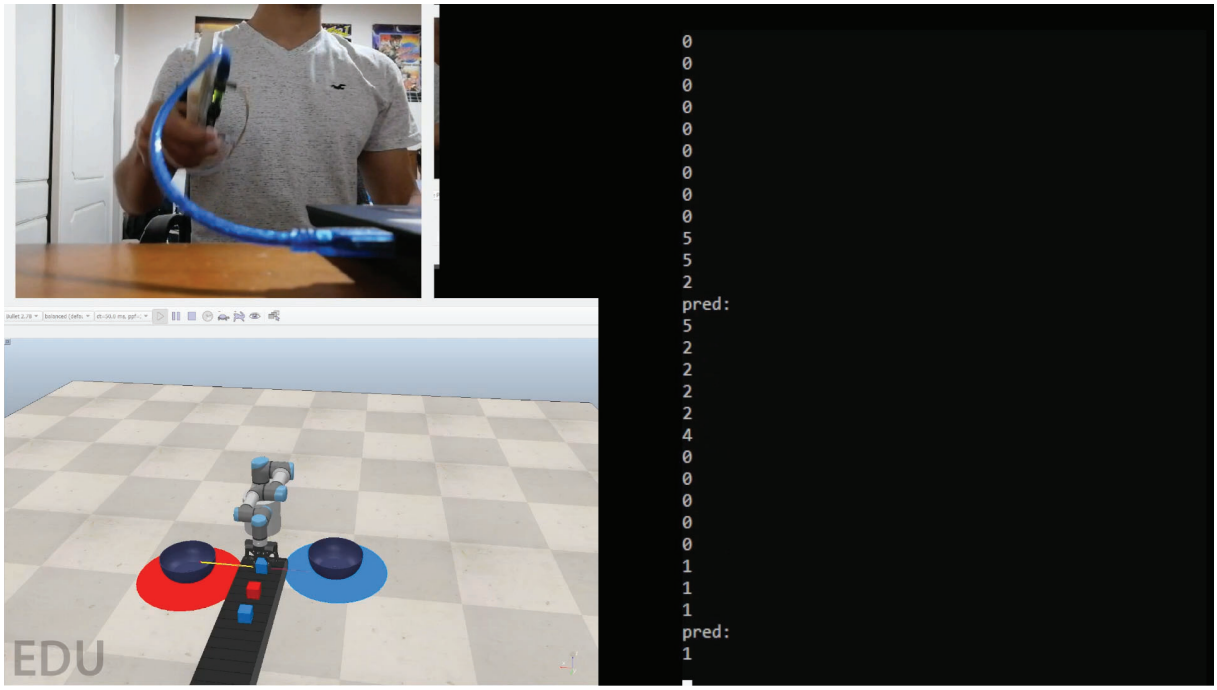

Figura 4.12: El usuario realiza el gesto 1 para subir el brazo. En la salida por consola se observa como el sistema reconoce de forma continua los gestos hasta que se cumplen la condiciones para tener en cuenta el gesto y poder controlar el robot

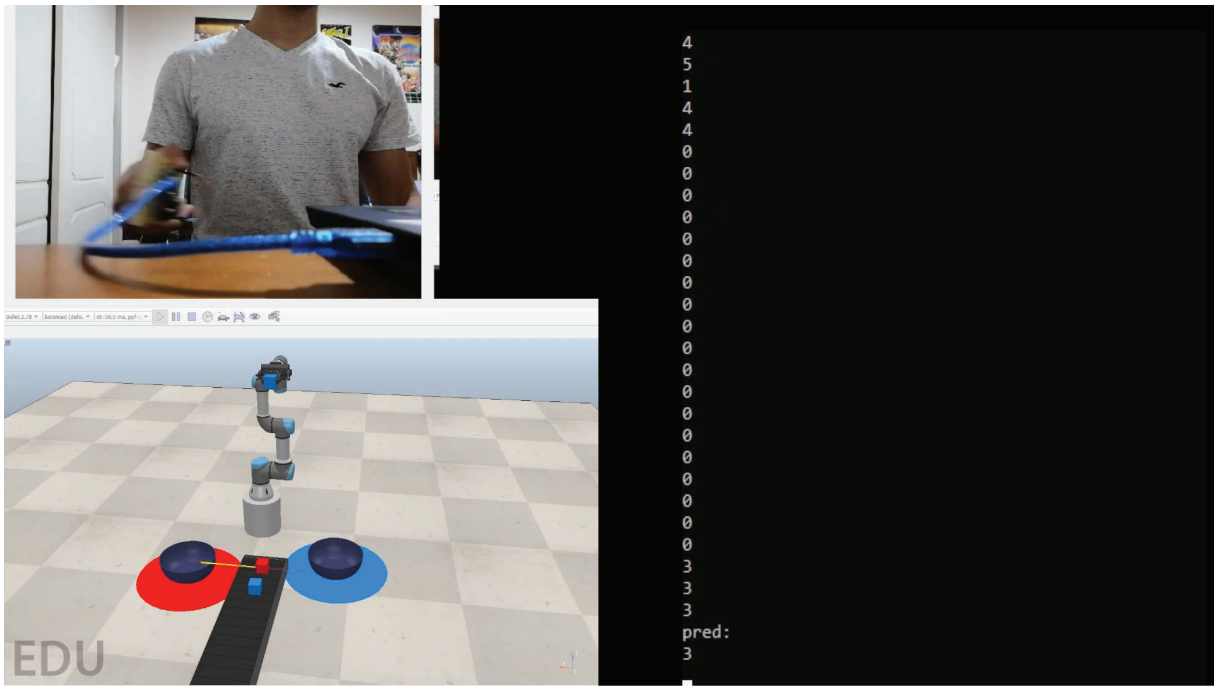

Figura 4.13: El usuario realiza el gesto 3 para orientar el brazo a la derecha. En la salida por consola se observa como el sistema reconoce de forma continua los gestos hasta que se cumplen la condiciones para tener en cuenta el gesto y poder controlar el robot

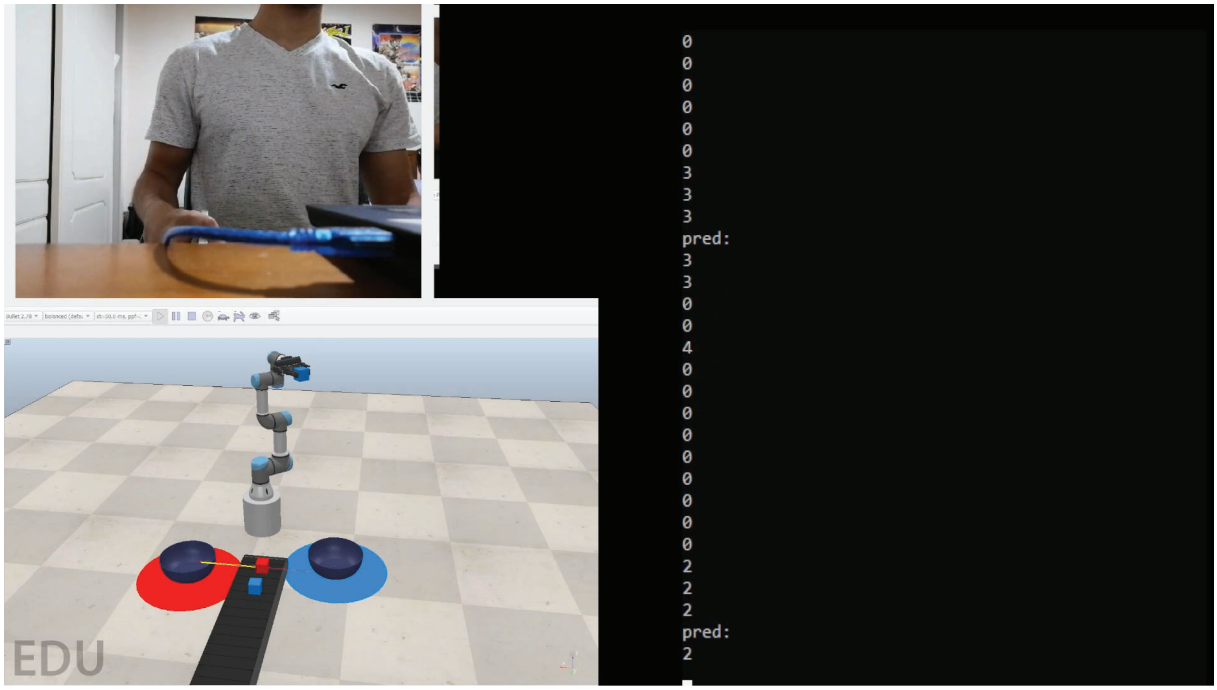

Figura 4.14: El usuario realiza el gesto 2 para bajar el brazo. En la salida por consola se observa como el sistema reconoce de forma continua los gestos hasta que se cumplen la condiciones para tener en cuenta el gesto y poder controlar el robot

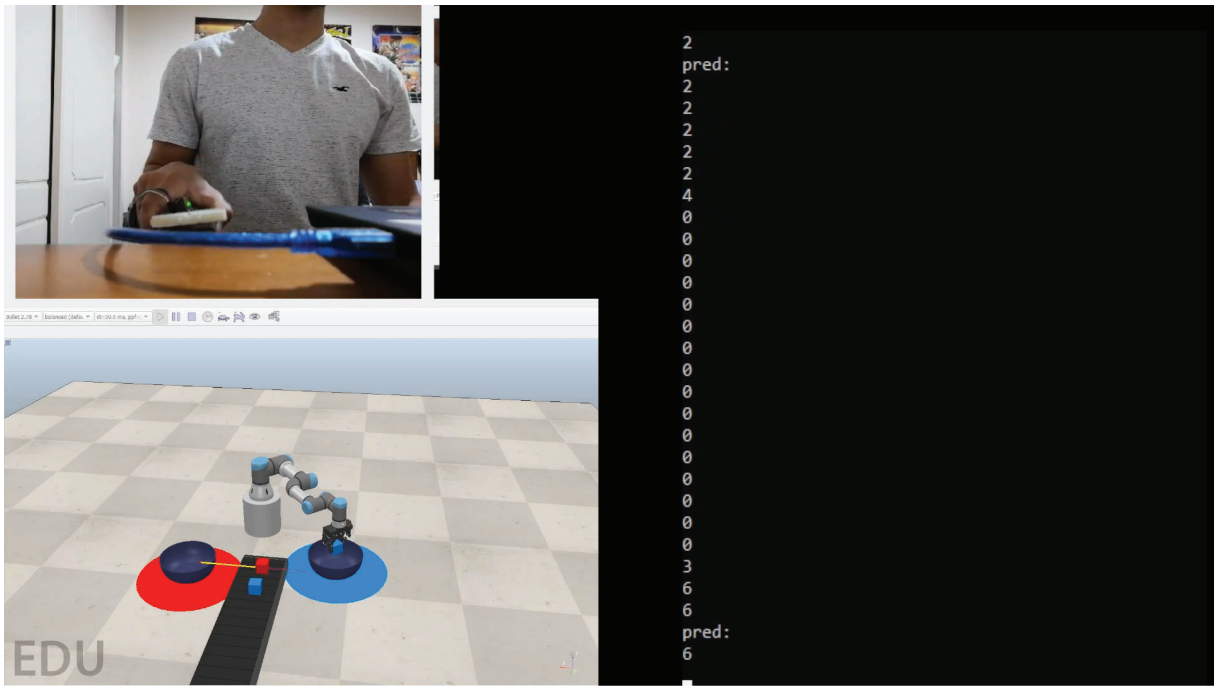

Figura 4.15: El usuario realiza el gesto 2 para abrir la pinza y soltar el objeto. En la salida por consola se observa como el sistema reconoce de forma continua los gestos hasta que se cumplen la condiciones para tener en cuenta el gesto y poder controlar el robot

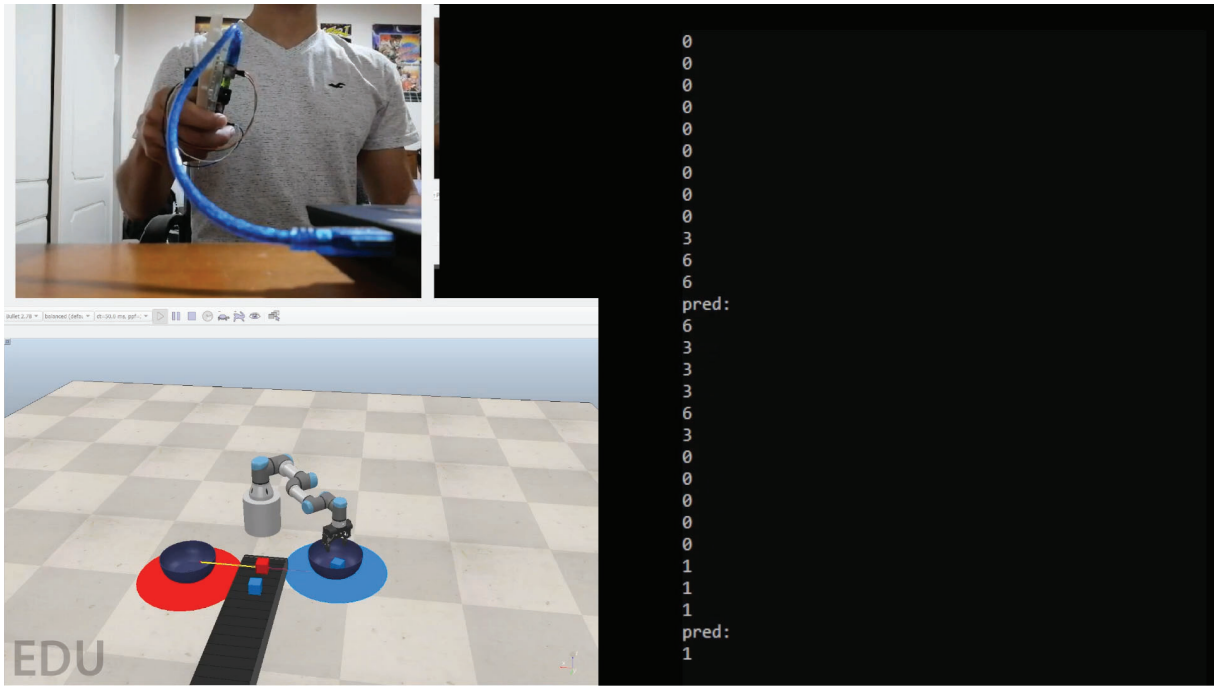

Figura 4.16: El usuario realiza el gesto 1 para subir el brazo. En la salida por consola se observa como el sistema reconoce de forma continua los gestos hasta que se cumplen la condiciones para tener en cuenta el gesto y poder controlar el robot

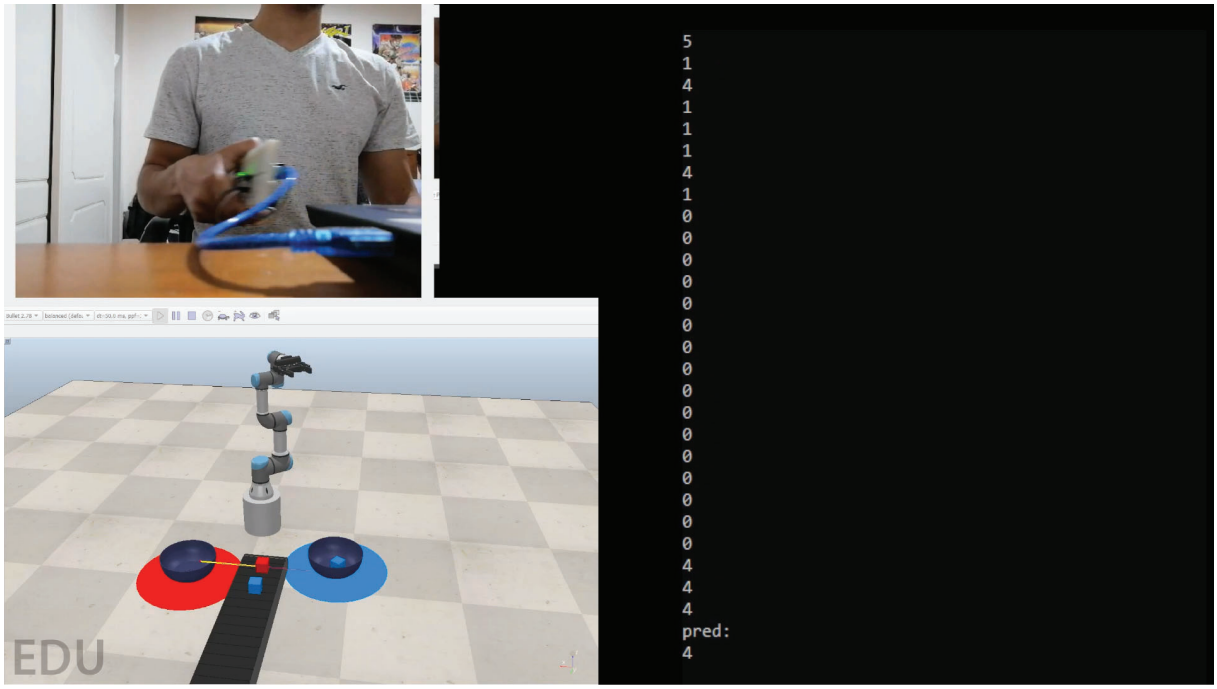

Figura 4.17: El usuario realiza el gesto 4 para orientar el brazo hacia la cinta transportadora. En la salida por consola se observa como el sistema reconoce de forma continua los gestos hasta que se cumplen la condiciones para tener en cuenta el gesto y poder controlar el robot

Capítulo 5

Conclusiones y trabajos futuros

Para finalizar este trabajo se procederá a analizar la efectividad de los métodos empleados durante el desarrollo del mismo.

5.1. Conclusiones

Con respecto a los materiales usados, Arduino presenta un entorno de desarrollo fácil de entender para la programación de sus placas con el que a través de simples conexiones entre módulos es posible recibir datos de diversos sensores. Este trabajo demuestra que es posible desarrollar modelos de reconocimiento de gestos mediante el uso de una plataforma de código abierto como Arduino en lugar de usar sensores más sofisticados como los vistos en el Capítulo 2 que por lo general son menos económicos y más difíciles de conseguir.

Con respecto al modelo neuronal elegido, se puede concluir que las redes LSTM presentan una potente capacidad para tratar con datos en serie cronológica como los vistos en este trabajo, puesto que en un tiempo de entrenamiento reducido se consiguió una precisión del 100 % en la clasificación de 7 movimientos, teniendo en cuenta que cada movimiento corresponde a una matriz con los datos recibidos por un acelerómetro y giroscopio durante un tiempo determinado.

Para el reconocimiento continuo se optó por usar una ventana deslizante con un 80 % de solapamiento, esta decisión se tomó para asegurar que el sistema reconociese en un tiempo razonable los gestos del usuario, puesto que un bajo porcentaje de solapamiento supondría altos tiempo de espera durante la recepción de todos los datos necesarios para proceder a su clasificación. Utilizar ventanas contiguas solamente agravaría este problema. En los experimentos se demostró cómo este alto porcentaje asegura que el reconocimiento se realiza casi en tiempo real y el usuario puede controlar al robot satisfactoriemante sin tener apenas retrasos en el envío de instrucciones al robot.

Para evitar la aparición de falsos positivos durante el reconocimiento continuo se utilizó una estructura *FIFO* con una serie de condiciones expuestas en el Capítulo 3. Durante los experimentos se demuestra cómo este método funciona durante el reconocimiento de gestos. Puesto que, el sistema solamente tiene en cuenta el gesto

reconocido cuando el sensor se encuentra previamente en posición neutral, por lo tanto si el usuario quiere bajar el brazo para relajarse o lo mueve de manera aleatoria, el robot permanecerá detenido. Además, incluso al no realizar un gesto con la precisión suficiente como para que sea 100 % reconocido con exactitud, los experimentos demuestran que las condiciones explicadas en el Capítulo 3 son suficientes para solventar este problema y el robot se controla correctamente.

El uso de cinemática directa para controlar los movimientos del robot a partir de sus articulaciones resultó de utilidad para la realización del experimento. Puesto que permite una fácil programación del robot mediante una serie de configuraciones, de tal forma que al controlar el robot mediante gestos, se está navegando por dichas configuraciones de forma precisa.

Por último, este trabajo demuestra que el uso de sensores inerciales para controlar un Cobot es fácilmente aplicable en un entorno industrial, concretamente en tareas en donde se necesite recoger y dejar objetos. Según avance la tecnología y se consigan modelos más potentes para reconocer gestos, la colaboración ente humanos y robots tendrá un futuro prometedor si se siguen desarrollando sistemas como el expuesto en este trabajo.

5.2. Mejoras y Trabajos futuros

En primer lugar, es necesario comentar que a pesar del alto porcentaje de Precisión que presenta el modelo neuronal y de los experimentos realizados, es necesario obtener algún tipo de métrica acerca de aspectos relativos a la usabilidad. Por eso, como mejora se puede realizar el experimento con un gran grupo de usuarios para que completen una encuesta sobre la facilidad al usar el sistema y sobre el éxito que tuvieron al tratar de completar la tarea del experimento. De esta forma se pueden obtener métricas que proporcionan más información acerca de la viabilidad del sistema en entornos industriales.

Durante la obtención de datos, se explicó que se utilizó el Ruido Gaussiano para obtener un mayor número de ejemplos de gestos para entrenar el modelo. Como propuesta de mejora, es posible aplicar otras técnicas de *Data Augmentation* pertenecientes al Estado del Arte y que son expuestas en la revisión de [Iwana y Uchida (2021)]. De esta forma, se conseguiría aumentar la robustez del modelo ante los gestos de los usuarios.

Es necesario comentar también que, a pesar de que este trabajo se enfoca en aplicaciones con los Cobots y en tareas de recoger y dejar objetos, sería interesante ver cómo se comportaría un sistema de reconocimiento de gestos en otros casos de uso. La conducción de drones y vehículos es un área en donde existe potencial, debido a que tradicionalmente se realiza con un mando a distancia, el objetivo sería eliminar ese dispositivo para utilizar los gestos como modo de control. En los videojuegos también existe potencial, debido a que en el pasado ya se han desarrollado juegos en donde es necesario utilizar acelerómetros para poder interacturar con ellos. Mediante un sistema de reconocimiento de gestos es posible aumentar la sensación realista de estos juegos y mejorar la experiencia del usuario en sistemas de realidad virtual.

Finalmente, como propuesta de trabajo futuro, sería interesante observar cómo se comportaría un sistema durante el recononcimiento de gestos más complejos que los vistos en este trabajo, se podría incorporar también el uso de sensores adicionales como un magnetómetro, que fue descartado para esta proyecto. Y así explorar el potencial que tienen los sistemes de este tipo y, sobre todo, su utilidad en distintos casos de uso.

Bibliografía

[Bai *et. al* (2012)] Bai, J., Goldsmith, J., Caffo, B., Glass, T. A., y Crainiceanu, C. M. (2012). Movelets: A dictionary of movement. Electronic journal of statistics, 6, 559.

[Chen *et. al* (2018)] Chen, F., Lv, H., Pang, Z., Zhang, J., Hou, Y., Gu, Y., ... y Yang, G. (2018). WristCam: A wearable sensor for hand trajectory gesture recognition and intelligent human–robot interaction. IEEE Sensors Journal, 19(19), 8441-8451.

[Chollet (2021)] Chollet, F. (2021). Deep learning with Python. Simon and Schuster.

[Coupeté *et. al* (2015)] Coupeté, E., Moutarde, F., y Manitsaris, S. (2015). Gesture recognition using a depth camera for human robot collaboration on assembly line. Procedia Manufacturing, 3, 518-525.

[Dong *et. al* (2017)] Dong, W., Xiao, L., Hu, W., Zhu, C., Huang, Y., y Yin, Z. (2017). Wearable human–machine interface based on PVDF piezoelectric sensor. Transactions of the Institute of Measurement and Control, 39(4), 398-403.

[Erdoğan *et. al* (2016)] Erdoğan, K., Durdu, A., y Yilmaz, N. (2016, April). Intention recognition using leap motion controller and Artificial Neural Networks. In 2016 International Conference on Control, Decision and Information Technologies (CoDIT) (pp. 689-693). IEEE.

[Fathi y Curran (2017)] Fathi, A., y Curran, K. (2017). Detection of spine curvature using wireless sensors. Journal of King Saud University-Science, 29(4), 553-560.

[Huang y Onnela (2019)] Huang, E., y Onnela, J. P. (2019). Activity Classification Using Smartphone Gyroscope and Accelerometer Data. arXiv preprint arXiv:1903.12616.

[Iwana y Uchida (2021)] Iwana, B. K., y Uchida, S. (2021). An empirical survey of data augmentation for time series classification with neural networks. Plos one, 16(7), e0254841.

[Jiang *et. al* (2017)] Jiang, S., Lv, B., Guo, W., Zhang, C., Wang, H., Sheng, X., y Shull, P. B. (2017). Feasibility of wrist-worn, real-time hand, and surface gesture recognition via sEMG and IMU sensing. IEEE Transactions on Industrial Informatics, 14(8), 3376-3385.

[Jiang *et. al* (2020)] Jiang, S., Gao, Q., Liu, H., y Shull, P. B. (2020). A novel, co-located EMG-FMG-sensing wearable armband for hand gesture recognition. Sensors and Actuators A: Physical, 301, 111738.

[Kato y Takemura (2016)] Kato, H., y Takemura, K. (2016, September). Hand pose estimation based on active bone-conducted sound sensing. In Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct (pp. 109-112).

[Matheson *et. al* (2019)] Matheson, E., Minto, R., Zampieri, E. G., Faccio, M., y Rosati, G. (2019). Human–robot collaboration in manufacturing applications: a review. Robotics, 8(4), 100.

[Mazhar *et. al* (2019)] Mazhar, O., Navarro, B., Ramdani, S., Passama, R., y Cherubini, A. (2019). A real-time human-robot interaction framework with robust background invariant hand gesture detection. Robotics and Computer-Integrated Manufacturing, 60, 34-48.

[Mummadi *et. al* 2018] Mummadi, C. K., Leo, F. P. P., Verma, K. D., Kasireddy, S., Scholl, P. M., Kempfle, J., y Laerhoven, K. V. (2018, June). Real-time and embedded detection of hand gestures with an IMU-based glove. In Informatics (Vol. 5, No. 2, p. 28). Multidisciplinary Digital Publishing Institute.

[Ortega-Avila *et. al* (2015)] Ortega-Avila, S., Rakova, B., Sadi, S., y Mistry, P. (2015, March). Non-invasive optical detection of hand gestures. In proceedings of the 6th augmented human international conference (pp. 179-180).

[Shi *et. al* (2018)] Shi, W. T., Lyu, Z. J., Tang, S. T., Chia, T. L., y Yang, C. Y. (2018). A bionic hand controlled by hand gesture recognition based on surface EMG signals: A preliminary study. Biocybernetics and Biomedical Engineering, 38(1), 126-135.

[Wang *et. al* (2018)] Wang, W., Li, R., Diekel, Z. M., Chen, Y., Zhang, Z., y Jia, Y. (2018). Controlling object hand-over in human–robot collaboration via natural wearable sensing. IEEE Transactions on Human-Machine Systems, 49(1), 59-71.

[Wang *et. al* (2020)] Wang, C., Guo, W., Zhang, H., Guo, L., Huang, C., y Lin, C. (2020). sEMG-based continuous estimation of grasp movements by long-short term memory network. Biomedical Signal Processing and Control, 59, 101774.

[Wen *et. al* (2016)] Wen, H., Ramos Rojas, J., y Dey, A. K. (2016, May). Serendipity: Finger gesture recognition using an off-the-shelf smartwatch. In Proceedings of the 2016 CHI conference on human factors in computing systems (pp. 3847-3851).

[Yang *et. al* (2018)] Yang, X., Sun, X., Zhou, D., Li, Y., y Liu, H. (2018). Towards wearable A-mode ultrasound sensing for real-time finger motion recognition. IEEE Transactions on Neural Systems and Rehabilitation Engineering, 26(6), 1199-1208.

Apéndice A

Código Arduino

```
#include <Wire.h>
#include <Kalman.h> // Source: https://github.com/TKJElectronics/KalmanFilter
#include <TimeLib.h>

Kalman kalmanX; // Crear instancias de Kalman
Kalman kalmanY;

/* Datos del IMU */
double accX, accY, accZ;
double gyroX, gyroY, gyroZ;

double kalAngleX, kalAngleY; // Angulos calculados usando el Filtro de Kalman

uint32_t timer;
uint8_t i2cData[14]; // Buffer para los datos provenientes del I2C

int cont = 0; // Contador para el numero de datos leidos del MPU6050

void setup() {
    Serial.begin(9600);
    Wire.begin();
    #if ARDUINO >= 157
        Wire.setClock(400000UL); // Establecer frecuencia del Arduino de 400kHz
    #else
        TWBR = ((F_CPU / 400000UL) - 16) / 2; // Establecer frecuencia del Arduino de 400kHz
    #endif

    i2cData[0] = 7; // Establecer la frecuencia de muestreo a 1 kHz
    while (i2cWrite(0x19, i2cData, 4, false)); // Escribir en los 4 registros a la vez

    while (i2cRead(0x75, i2cData, 1));
    if (i2cData[0] != 0x68) { // Leer el registro "WHO_AM_I"
        Serial.print(F("Error leyendo el sensor"));
        while (1);
    }

    delay(100); // Esperar a que el sensor se estabilice

    /* Establecer el angulo inicial en el que se encuentra el MPU6050 */
    while (i2cRead(0x3B, i2cData, 6)); // Leer los datos del MPU6050
    accX = (int16_t)((i2cData[0] << 8) | i2cData[1]);
    accY = (int16_t)((i2cData[2] << 8) | i2cData[3]);
    accZ = (int16_t)((i2cData[4] << 8) | i2cData[5]);

    //atan2 devuelve valores de -pi hasta pi (radianes) ver http://en.wikipedia.org/wiki/Atan2
    // Se convierte el valor a grados
    double roll  = atan2(accY, accZ) * RAD_TO_DEG; // Calcular el angulo usando solo el
                                                   //acelerometro
    double pitch = atan(-accX / sqrt(accY * accY + accZ * accZ)) * RAD_TO_DEG;
```

```
49
50      kalmanX.setAngle(roll); // Establecer el angulo inicial en el que se encuentra el MPU6050
51      kalmanY.setAngle(pitch);
52
53      timer = micros();
54  }
55
56  void loop() {
57      while (i2cRead(0x3B, i2cData, 14)); // Leer los datos del MPU6050
58      accX = (int16_t)((i2cData[0] << 8) | i2cData[1]);
59      accY = (int16_t)((i2cData[2] << 8) | i2cData[3]);
60      accZ = (int16_t)((i2cData[4] << 8) | i2cData[5]);
61
62      gyroX = (int16_t)((i2cData[8] << 8) | i2cData[9]);
63      gyroY = (int16_t)((i2cData[10] << 8) | i2cData[11]);
64      gyroZ = (int16_t)((i2cData[12] << 8) | i2cData[13]);;
65
66      double dt = (double)(micros() - timer) / 1000000; // Calcular delta time
67      timer = micros();
68
69      //atan2 devuelve valores de -pi hasta pi (radianes) ver http://en.wikipedia.org/wiki/Atan2
70      // Se convierte el valor a grados
71
72      double roll  = atan2(accY, accZ) * RAD_TO_DEG; // Calcular el angulo usando solo el
            acelerometro
73      double pitch = atan(-accX / sqrt(accY * accY + accZ * accZ)) * RAD_TO_DEG;
74
75      double gyroXrate = gyroX / 131.0; // Convertir a deg/s
76      double gyroYrate = gyroY / 131.0; // Convertir a deg/s
77
78      kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt); // Calcular el angulo usando el Filtro
                                                           //de Kalman
79                                                          // en donde se incorporan acelerometro
                                                               //, giroscopio y el tiempo delta
80      kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt);
81
82      /* Escribir los datos por consola */
83
84      //if (cont < 50){ // Descomentar el if para recibir datos solo durante 2,5 segundos
85          Serial.print(accX * (9.81/16384.0)); Serial.print(",");
86          Serial.print(accY * (9.81/16384.0)); Serial.print(",");
87          Serial.print(accZ * (9.81/16384.0)); Serial.print(",");
88
89          Serial.print(gyroX / 131.0); Serial.print(",");
90          Serial.print(gyroY / 131.0); Serial.print(",");
91          Serial.print(gyroZ / 131.0); Serial.print(",");
92
93          Serial.print(kalAngleX); Serial.print(",");
94
95          Serial.print(kalAngleY);
96
97          Serial.print("\r\n");
98      //}
99      delay(2);
100     cont++;
101 }
```

Apéndice B

Código de implementación del Ruido Gaussiano aplicado a datos de serie temporal

```
import os
import numpy as np
import csv
from random import randrange

data_dir = 'C:/Users/penad/Desktop/MUIA/TFM/Logger'
tag_dir = 'C:/Users/penad/Desktop/MUIA/TFM/Logger/Data'
fname = os.path.join(tag_dir, 'LabelsDataAug.csv')
fnameData = os.path.join(data_dir, 'DataArduino.csv')
fnameTags = os.path.join(data_dir, 'LabelsArduino.csv')
fnameWrite = os.path.join(tag_dir, 'DataAug.csv')
MaxPages = 200000
MaxExamples = 280

def get_sensor_data(path, posSup, posInf): #Obtener los datos de un ejemplo a partir de su
    ubicacion en el csv
    f = open(path)
    sensordata = f.read()
    f.close()
    sensorlines = sensordata.split('\n')
    sensorheader = sensorlines[0].split(',')
    sensorlines = sensorlines[0:]
    sensor_float_data = np.zeros((50, len(sensorheader)))
    i = posInf
    while (i <= posSup):
        values = [float(x) for x in sensorlines[i].split(',')]
        sensor_float_data[i - posInf, :] = values
        i += 1
    return sensor_float_data

def get_rand_and_data(randomTag): #Obtener los datos de un ejemplo a partir de la direccion de
     su etiqueta
    posSup = (randomTag * 50) - 1
    posInf = posSup - 49
    csvsensor = get_sensor_data(fnameData, posSup, posInf)
    return csvsensor

def get_neg():
    if(randrange(2) == 1):
        neg = -1
    else:
        neg = 1
    return neg
```

```
42
43 def get_div():
44     if (randrange(2) == 1):
45         div = 10
46     else :
47         div = 100
48     return div
49
50 def get_std(path):  #Obtener desviacion estandar del conjunto de datos
51     f = open(path)
52     data = f.read()
53     f.close()
54     lines = data.split('\n')
55     header = lines[0].split(',')
56     lines = lines[0:]
57
58     float_data = np.zeros((len(lines)-1, len(header)))
59     for i, line in enumerate(lines):
60         if(len(line) > 0):
61             values = [float(x) for x in line.split(',')]
62             float_data[i, :] = values
63     mean = float_data.mean(axis=0)
64     float_data -= mean
65     std = float_data.std(axis=0)
66     return std
67
68
69 f = open(fname) #Leer datos obtenidos con el sensor
70 data = f.read()
71 f.close()
72 lines = data.split('\n')
73 header = lines[0].split(',')
74 lines = lines[0:]
75
76 float_data = np.zeros((len(lines), len(header)))
77
78 for i, line in enumerate(lines):
79     values = [float(x) for x in line.split(',')]
80     float_data[i, :] = values
81
82 f = open(fnameTags) #Leer datos de las etiquetas de los ejemplos
83 data = f.read()
84 f.close()
85 lines = data.split('\n')
86 header = lines[0].split(',')
87 lines = lines[0:]
88
89 Tags_data = np.zeros((len(lines), len(header)))
90
91 for i, line in enumerate(lines):
92     values = [float(x) for x in line.split(',')]
93     Tags_data[i, :] = values
94
95
96 cont = 0
97 std = get_std(fnameData) #Obtener desviacion estandar de los datos
98 while (cont < MaxPages):    #Obtener 200000 ejemplos nuevos aplicando ruido gaussiano a un
       ejemplo aleatorio
99     randomTag = randrange(MaxExamples)
100    tag = float_data[0][cont]
101    num = Tags_data[0][randomTag]
102    while (num != tag):
103        randomTag = randrange(MaxExamples)
104        num = Tags_data[0][randomTag]
105    print(cont)
106    cont += 1
107    csvsensor = get_rand_and_data(randomTag + 1)
108    rowcont = 0
109    f = open(fnameWrite, 'a+', newline='')
110    writer = csv.writer(f)
```

Código de implementación del Ruido Gaussiano aplicado a datos de serie temporal

```
111     while (rowcont < 50):
112         row = csvsensor[rowcont]
113         col = 0
114         while (col < len(csvsensor[rowcont])):
115             row[col] = round(row[col] + (get_neg() * randrange(int(round(std[col],2))*100))/
                    randrange(100,1001), 2)
116             col += 1
117         writer.writerow(row)
118         rowcont += 1
119     f.close()
```

Apéndice C

Código de entrenamiento y validación del modelo de reconocimiento de gestos

C.1. Procesado de datos

```
import numpy as np

data_dir = './Data/DataAug.csv' #Directorio y fichero en donde se encuentran los datos
                                #obtenidos con el Data Augmentation

pageSize = 50       #Numero de filas en un ejemplo
maxPages = 200000   #Numero de ejemplos guardados

f = open(data_dir)  #Abrir el fichero y leer todos los datos

data = f.read()
f.close()
lines = data.split('\n')
header = lines[0].split(',')
lines = lines[0:]

float_data = np.zeros((len(lines)-1, len(header)))
final_data = np.zeros((maxPages, pageSize, len(header)))  #La estructura donde se guardan los
                                                          #datos es en un tensor de dimension
                                                          # 200000 X 50 X 8

for i, line in enumerate(lines):
  if(len(line) > 0):
    values = [float(x) for x in line.split(',')]
    float_data[i, :] = values

page = 0
while (page < maxPages):
  i = 0
  while (i < pageSize):
    values = [(x) for x in lines[page * pageSize + i].split(',')]
    final_data[page, i, :] = values
    i += 1
  page += 1

mean = float_data.mean(axis=0)  #Se normalizan los datos restandole la media y dividiendolos
                                #por la varianza
float_data -= mean

```

```
vari = float_data.var(axis=0)
float_data /= vari

final_data -= mean
final_data /= vari


array_labels = './Data/LabelsDataAug.csv'    #Directorio y fichero que contiene las etiquetas
                                             #de los ejemplos anteriores

f = open(array_labels)    #Abrir el fichero y leer todos los datos

data = f.read()
f.close()
lines = data.split('\n')
header = lines[0].split(',')
lines = lines[0:]

array_data = np.zeros((len(lines), len(header)))

for i, line in enumerate(lines):
    if(len(line) > 0):
        values = [float(x) for x in line.split(',')]
        array_data[i, :] = values
```

C.2. Modelo de reconocimiento de gestos

C.2.1. Datos de entrenamiento y validación

```
x_train = final_data[:160000]           #Se divide el conjunto de datos en 2: Entrenamiento y
    validacion
y_train = array_data[0][:160000]        #En total se tienen 200000 ejemplos, de los cuales
                                        #160000 son de entrenamiento
                                        #40000 son de validacion
x_test = final_data[160000:160000+40000]
y_test = array_data[0][160000:160000+40000]
```

C.2.2. Generacion y entrenamiento del modelo

```
import tensorflow as tf
from tensorflow import keras
from sklearn.preprocessing import StandardScaler
import numpy as np

model = keras.Sequential()  #Se define el modelo neuronal LSTM y sus hiperparametros
model.add(keras.layers.LSTM(64, input_shape = (x_train.shape[1:]), activation='relu',
    return_sequences=True))
model.add(keras.layers.LSTM(32, activation='relu'))
model.add(keras.layers.Dense(32, activation='relu'))
model.add(keras.layers.Dense(6, activation='softmax'))

opt = keras.optimizers.Adam(learning_rate=1e-4, decay=1e-5)

#La funcion de perdida es la sparse_categorical_crossentropy debido a que se trata de un
            #problema de clasificacion y
#porque las etiquetas no estan codificadas con el metodo One-Hot, sino que se tratan
#de numeros enteros
model.compile(loss='sparse_categorical_crossentropy', optimizer=opt, metrics=['accuracy'])

#Se entrena el modelo
history = model.fit(x_train, y_train, epochs=4, validation_data=(x_test, y_test))
```

C.2.3. Rendimiento del modelo

```
import pandas as pd
import matplotlib.pyplot as plt
results=pd.DataFrame(history.history)
results.plot(figsize=(8, 5))
plt.grid(True)
plt.xlabel ("Epochs")
plt.ylabel ("Accuracy - Mean Log Loss")
plt.gca().set_ylim(0, 1.01)
plt.show()  #Visualizar el rendimiento del modelo
```

C.3. Validación del modelo y exportación

C.3.1. Procesado de datos

```
Ftest_dir = './DataArduino.csv'  #Leer los datos obtenidos manualmente con el Arduino, se
                                 #cargan en un tensor de dimension 280 X 50 X 8
                                #Se normalizan los datos de la misma forma que ene el caso
                                    anterior
pageSize = 50
maxPages = 280

f = open(Ftest_dir)

data = f.read()
f.close()
lines = data.split('\n')
header = lines[0].split(',')
lines = lines[0:]

float_data = np.zeros((len(lines)-1, len(header)))
final_data = np.zeros((maxPages, pageSize, len(header)))

for i, line in enumerate(lines):
  if(len(line) > 0):
    values = [float(x) for x in line.split(',')]
    float_data[i, :] = values

page = 0
while (page < maxPages):
  i = 0
  while (i < pageSize):
    values = [(x) for x in lines[page * pageSize + i].split(',')]
    final_data[page, i, :] = values
    i += 1
  page += 1

mean = float_data.mean(axis=0)
float_data -= mean

vari = float_data.var(axis=0)
float_data /= vari

final_data -= mean
final_data /= vari


Farray_labels = './LabelsArduino.csv'      #Leer las etiquetas de los datos obtenidos
                                           #manualmente

f = open(Farray_labels)

data = f.read()
f.close()
```

```
lines = data.split('\n')
header = lines[0].split(',')
lines = lines[0:]

Farray_data = np.zeros((len(lines), len(header)))

for i, line in enumerate(lines):
    if(len(line) > 0):
        values = [float(x) for x in line.split(',')]
        Farray_data[i, :] = values
```

C.3.2. Evaluación del modelo

```
x_Ftest = final_data        #Preparar los ejemplos y las etiquetas
y_Ftest = Farray_data[0]

model.evaluate(x_Ftest, y_Ftest)    #Evaluar el modelo
```

C.3.3. Exportación del modelo aprendido

```
model_json = model.to_json()            #Guardar el modelo LSTM y sus pesos
with open("modelDeepLSTM.json", "w") as json_file:
    json_file.write(model_json)
model.save_weights("modelDeepLSTM.h5")
print("Modelo guardado")
```

Apéndice D

Código del sistema de reconocimiento continuo de gestos aplicado a una interfaz robótica

```
ur3 = UR3(clientID) #Inicializar el robot UR3e

conf = ur3.model.qf #Se empieza por defecto con la configuracion qf
pos = "qf"
ur3.set_robot_joints(conf, blocking=True)


"""Implementacion del metodo de la ventana deslizante. Porcentaje de solapamiento de 80%.
    Primero se lee una ventana de 50 filas de datos, una vez se alcanza el numero maximo, se
    clasifica el gesto y se guarda la etiqueta en la cola. Luego se genera una nueva ventana
    en donde las primeras 40 filas corresponden a las 40 ultimas de la iteracion anterior. Las
     10 siguientes filas leidas son nuevas. Si la cola alcanza su capacidad maxima de 4, se
    saca el primer gesto guardado y se guarda el nuevo gesto clasificado en el ultimo puesto
    ."""

while sim.simxGetConnectionId(clientID) != -1:
            if (cont == 50):    #Si se leen 50 filas se clasifica el gesto
                threeD_data[0] = sensor_float_data
                out = loaded_model.predict(threeD_data)
                out = np.argmax(out,axis=1)
                """Si la cola alcanza el maximo de su capacidad y solamente existe una
                    etiqueta 0 que esta en el primer puesto, se saca la etiqueta mas frecuente
                     siempre que no se trate de un 0"""
                if (len(q)==4):
                    if (q[0] == 0 and most_frequent(q[1:]) != 0 and q.count(0) == 1):
                        print('pred:')
                        pred = most_frequent(q[1:])
                        print(pred)
                        """Se pausa el proceso por aproximadamente 2 segundos para permitir al
                             usuario volver a la posicion neutral con el sensor"""
                        return_to_idle(ser)
                    q.pop(0)    #Sacar el primer elemento
                q.append(out[0]) #Insertar la nueva etiqueta en el ultimo puesto
                print(out[0])
                #Restar 10 en el contador de filas y generar la nueva ventana con las 40 filas
                     anteriores
                cont -= 10
                sensor_float_data[:40] = sensor_float_data[10:10+40]
            getData=str(ser.readline()) #Obtener una fila de datos del sensor
```

```
            data=getData[2:][:-5]
            datalines = data.split('\n')
            datalines = datalines[0:]
            try:
                #Normalizar y guardar la fila obtenida en la ventana
                values = [float(x) for x in datalines[0].split(',')]
                sensor_float_data[cont, :] = values
                sensor_float_data[cont, :] -= mean
                sensor_float_data[cont, :] /= vari
                cont += 1
            except:
                print(".")


            if (pos == "qf"):   #Asignacion de las etiquetas de los gestos a las acciones del
                robot
                if(pred == 3):
                    conf = ur3.model.qd
                    pos = "qd"
                elif(pred == 4):
                    conf = ur3.model.qi
                    pos = "qi"
                elif(pred == 2):
                    conf = ur3.model.qfb
                    pos = "qfb"
            elif (pos == "qi"):
                if(pred == 3):
                    conf = ur3.model.qf
                    pos = "qf"
                elif(pred == 2):
                    conf = ur3.model.qib
                    pos = "qib"
            elif (pos == "qd"):
                if(pred == 4):
                    conf = ur3.model.qf
                    pos = "qf"
                elif(pred == 2):
                    conf = ur3.model.qdb
                    pos = "qdb"
            elif (pos == "qfb"):
                if(pred == 4):
                    conf = ur3.model.qib
                    pos = "qib"
                elif(pred == 3):
                    conf = ur3.model.qdb
                    pos = "qdb"
                elif(pred == 1):
                    conf = ur3.model.qf
                    pos = "qf"
            elif (pos == "qib"):
                if(pred == 3):
                    conf = ur3.model.qfb
                    pos = "qfb"
                elif(pred == 1):
                    conf = ur3.model.qi
                    pos = "qi"
            elif (pos == "qdb"):
                if(pred == 4):
                    conf = ur3.model.qfb
                    pos = "qfb"
                elif(pred == 1):
                    conf = ur3.model.qd
                    pos = "qd"

            # Accion
            if(pred == 5):
                Gblock = closeGripper(pos, Gblock) #Cerrar pinza
            elif(pred == 6):
                openGripper(pos) #Abrir pinza
```

```
98              ur3.set_robot_joints(conf, blocking=True) #Mover las articulaciones del robot
                    segun el gesto clasificado
99              pred = 0    #Se reinicia la etiqueta a 0 despues de la accion para volver al
                    estado idle
```