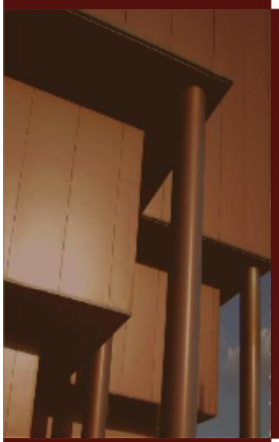
Trabajo Fin de Grado:

Ingeniería en Electrónica, Robótica y Mecatrónica

Título: **Aplicación de automatización distribuida con componentes de aprendizaje automático basada en la herramienta Node-RED**

Autor: Abdoulaye Abou Wade

Tutor: Joaquín Ferruz Melero



Dpto. Ingeniería de Sistemas y Automática

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado

Ingeniería en Electrónica, Robótica y Mecatrónica

Título:

Aplicación de automatización distribuida con componentes de aprendizaje automático basada en la herramienta Node-RED

Autor:

Abdoulaye Abou Wade

Tutor:

Joaquín Ferruz Melero

Dpto. Ingeniería de Sistemas y Automática

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado: Aplicación de automatización distribuida con componentes de aprendizaje automático basada en la herramienta Node-RED

Autor: Abdoulaye Abou Wade

Tutor: Joaquín Ferruz Melero

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

En esta etapa de mi vida me gustaría agradecer a todas aquellas que me han ayudado durante todas las etapas de mi vida, haciendo posible que pueda llegar hasta aquí. En especial a mi familia tanto los que están aquí presentes, en Europa, o en África.

Me gustaría agradecer también a todas aquellas personas que me han podido trasmitir cualquier conocimiento que me ha permitido lograr este objetivo.

Gracias a todos, desde el corazón.

Abdoulaye Abou Wade

Sevilla, 2020

Resumen

El objetivo de este proyecto es realizar una aplicación del aprendizaje automático a la automatización industrial.

La tarea que se ha realizado es el control de una planta industrial simplificada al que aplicamos aprendizaje automático para hacerla más autónoma, es una planta industrial simple a la que le aplicamos aprendizaje automático y está compuesta por un brazo robótico, una cinta transportadora, una Raspberry Pi, una Jetson AGX Xavier de tal forma que llega a través de la cinta un objeto al que hacemos una foto y detectamos qué objeto es para, por último, posicionarlo en un lugar determinado. Para ello hemos utilizado imágenes de distintos objetos y animales (perro, camión, coche, caballo, bicicleta, persona); por otro lado, también detectamos los objetos dependiendo de su color. También explicamos la manera de realizar un aprendizaje automático ("machine learning") desde cero.

Índice general

Notación

Índice de Figuras

Notación

IA	Inteligencia Artificial
GPU	Graphics Processing Unit
DLA	Deep Learning Accelerator
SoC	System-on-a-chip
SD	Secure Digital
CPU	Central Processing Unit
RAM	Random Access Memory
REST	Representational State Transfer
HTTP	Hypertext Transfer Protocol
API	Application Programming Interface
URI	Uniform Resource Identifier
MTML	HyperText Markup Language
Json	Javascript Object Notation
XML	Extensible Markup Language
CNN	Convolutional Neural Network
IOU	Intersection Over Union

I. Introducción

La tarea que se va a realizar en este trabajo es hacer una aplicación mediante la cual controlamos una planta industrial al que aplicamos aprendizaje automático para hacerla más autónoma.

La automatización industrial es la tecnología que se utiliza para permitir el funcionamiento automático de diferentes maquinarias y procesos en una industria sin la necesidad de intervención humana. Dicha automatización es posible gracias a los diferentes sistemas de control que tenemos en la actualidad. Entre dichos sistemas de control podemos nombrar los computadores, los robots y las tecnologías de información entre otras.

La automatización industrial busca fundamentalmente transferir las tomas de decisiones de los seres humanos a las máquinas y también las actividades manuales con el uso de equipos óptimos y comandos lógicos de programación.

1.1 Historia de la automatización industrial

A lo largo de la historia la automatización industrial ha ido evolucionando, sobre todo durante el siglo xx en la que se puede indicar sin duda que se experimentan los avances más importantes que nos han permitidos llegar al nivel que estamos ahora en la industria.

Desde la antigüedad al ser humano siempre le ha fascinado las máquinas; se ha empezado imitando seres como el hombre mecánico entre otros, orientado a la diversión.

Sin embargo, se tuvo que esperar hasta llegar a los siglos XVII y XVIII para llegar a lo que se conoce como la edad de oro de los autómatas, que se caracteriza por el desarrollo de la mecánica de precisión en la fabricación de relojes. En esta época se construyó el primer modelo de telar mecánico y se le aplicaron ideas de autómatas; también es de destacar la construcción del pato mecánico que según la leyenda comía y bebía, entre otros desarrollos.

En el siglo XVIII, siglo de la primera revolución industrial, se empieza a pensar en la importancia que podría suponer aplicar dichas técnicas

de autómatas en la industria para avanzar hacia la automatización de la fabricas: la producción sin intervención humana.

En esos momentos se empezó a desarrollar los elementos base que inciden a la automatización de los procesos industriales:

- Los sistemas de control que permiten gobernar el funcionamiento de las máquinas
- Las máquinas automáticas que realizan las operaciones de producción.

En el siglo XIX la automatización de los procesos industriales da otro salto, se logra el almacenamiento de las secuencias de operaciones y los tiempos de aplicación de las mismas con la programación por levas.

Dichos programas son controlados mediante dispositivos mecánicas que nos permiten realizar y repetir las secuencias con sus mecanismos.

El otro gran avance de ese siglo se le denomina regulación que consiste en la realimentación ("feedback"); es fundamental para la uniformidad, precisión y la calidad en la fabricación de productos.

También apareció el concepto de servomecanismo para el control de grandes máquinas como un elemento adicional de los reguladores. Fue introducido por Joseph Farcot en 1873 para describir un sistema de regulación automático que utiliza motores controlados por dicho sistema.

Aparte de los dispositivos mecánicos para el almacenamiento de programas, se idea en este siglo un sistema basado en cinta o tarjeta perforada para los mismos fines y que en 1950 llevó, mediando investigaciones del Instituto de Tecnología de Massachusetts, a las primeras máquinas de control numérico. En esas máquinas las posiciones de herramientas, mesas de posición, velocidades y alimentadores son indicadas por agujeros en la cinta.

Durante el siglo xx se utilizan todos los avances, todas las técnicas desarrolladas y se mejoran para llevar a la automatización industrial a otro nivel, logrando automatizar plantas enteras de fabricación de productos sin ninguna intervención humana [1].

Introducción

1.2 Actualidad de la automatización industrial

En la actualidad estamos con la cuarta revolución industrial, las industrias están experimentado un proceso de cambio importante. Con la llegada de los robots industriales y las nuevas tecnologías hay una oportunidad importante de automatizar totalmente las tareas en las industrias e incluso de automatizar plantas de trabajo enteras o una fábrica en su totalidad. El avance más importante podemos decir que es el uso de Internet en la industria, Internet de las cosas, que permite a los objetos interconectarse mediante Internet y a grandes distancias. Lo que nos permitirá en un futuro poder interconectar fábricas y aumentar el grado automatización a nivel industrial.

Consecuencias de los avances industriales

- **Ventajas**
  1. Los costes operativos disminuyen considerablemente. Los procesos no requieren de personal que realice tareas mecánicas y, a su vez, otro tipo de gastos relacionados con el stock o suministros se reducen, pues se optimizan los recursos energéticos.
  2. Se eliminan los errores humanos. Como las tareas se automatizan y se programan para determinados patrones, los despistes o fallos provenientes de las personas desaparecen totalmente de la ecuación, dando más eficiencia y fiabilidad a los resultados.
  3. En cuanto al personal de las empresas se pueden dedicar a explotar sus capacidades intelectuales y aportar conocimiento para buscar nuevas estrategias de mejora y crecimiento, lo que hará al negocio mucho más competente de cara al mercado.
  4. Los sistemas trabajan sin horario ni interrupciones y su tráfico de datos ayuda a disponer de mejores análisis, que beneficiarán la comprensión e interpretación de la realidad con respecto a la oferta y demanda de productos y servicios.
  5. El control sobre los procesos que maneja la automatización es más eficiente y los fallos que detecta el sistema se dan a conocer instantáneamente.

- **Desventajas**
  1. Incertidumbre laboral.
     Es cierto que la automatización eliminará ciertos trabajos, pero también automatizar procesos aportará nuevos puestos de trabajo, ya que los empleados pueden abordar muchos más proyectos que antes no alcanzaban a llegar por falta de tiempo. El tipo de empleado es diferente por la necesidad de personal formado para las nuevas tecnologías.

  2. Alta inversión inicial. Antes de que, a nivel económico, esta tecnología comience a ser rentable, hay que realizar un primer esfuerzo económico.

  3. Gestión de cambios. Contar con esta herramienta supone integrar procesos de control de calidad, que suponen cierto grado de esfuerzo y tiempo. Esto implica estar pendiente de actualizaciones y análisis y retroalimentación de lo que aporta este nuevo sistema, para que su utilidad y beneficios no se queden estancados y evolucionen en cuanto a funcionalidades en base a las necesidades de la compañía.

En general podemos indicar que esta tecnología es una ventaja competitiva en el mundo industrial y afectará según las previsiones a todas las áreas y personas [2].

1.3 Introducción sobre la planta industrial simplificada

Como hemos mencionado anteriormente, la tarea que se va a realizar en este trabajo es el control de una planta industrial simplificada al que aplicamos aprendizaje automático. Dicha planta está compuesta por un brazo robótico, una cinta transportadora, una Raspberry Pi, una Jetson AGX Xavier y un portátil de tal forma que llega a través de la cinta un objeto al que hacemos una foto y detectamos qué objeto es para, por último, posicionarlo en un lugar determinado. Para ello, hemos utilizado imágenes de distintos objetos y animales (perro, camión, coche, caballo, bicicleta, persona), por otro lado, también

detectamos los objetos dependiendo de su color. También explicamos la manera de realizar un aprendizaje automático (“machine learning”) desde cero.

- La cinta es controlada por una Raspberry Pi de tal forma que el objeto entra en la planta a través de dicha cinta que le lleva a un lugar predefinido. En este caso dicho objeto es un dado que contiene en cada cara un color o una imagen que puede ser algunas de los objetos o animales nombrados anteriormente.

- Una vez la cinta posiciona el dado o carta, el robot se posiciona para tener una buena perspectiva para la foto que se realiza a continuación a través de una cámara colocada en la muñeca del robot. La cámara es controlada por la Raspberry Pi que la activa para hacer la foto y lo guarda en un archivo y luego pasa la foto a una determinada carpeta de la Jetson AGX Xavier.

- La Jetson AGX Xavier es el dispositivo que utilizamos para realizar los aprendizajes automáticos (“machine learning”) ya que está creado para entre otras tareas realizar aprendizajes automáticos de forma más rápida e eficiente. En nuestro caso aprende a reconocer, por un lado, si el objetos o animal presente en una determinada imagen es una bicicleta, un coche, un caballo, una persona, un camión o un perro. Y, por otro lado, aprende a reconocer el color presente una imagen. La Jetson AGX Xavier en este trabajo recibe como entrada una foto y detecta la imagen en la foto y como salida nos da el nombre del objeto o animal presente en la imagen y el centro de la imagen, o detecta el color del objeto y su centro utilizando algoritmos de la biblioteca de visión artificial OpenCv.

- El portátil recibe el nombre y el centro de la imagen y activa el robot para recoger el objeto y posicionarlo en un determinado lugar para su salida de la planta o futuras operaciones.

- La comunicación entre los distintos dispositivos la hacemos a través de Ethernet por medio de un "router" que nos permite conectar varios dispositivos a la vez. También utilizamos la herramienta Node-RED para realizar una aplicación tal que podemos controlar la planta a través nuestro portátil o teléfono móvil. A través de dicha herramienta indicamos mediante un botón de entrada que queremos realizar una detección; dicha detección puede ser de un objeto o color presente en la imagen de una cara del dado.

  Primero el objeto llega a través de la cinta, cae en una zona de foto, el robot se posiciona y se realiza la foto. El robot recoge el dado o carta y lo lleva a una posición de salida dependiendo de la imagen, color o tipo de carta que tenemos.

El objetivo fundamental de este trabajo es la interconexión de varios dispositivos en una misma planta y aprendizaje automático ("machine learning") ya que son dos cosas que son y van a ser fundamental para el futuro de la automatización industrial y en la inteligencia artificial.

II. Dispositivos y herramientas

La planta descrita anteriormente está compuesta por distintos dispositivos y herramientas para llegar al objetivo fijado.

Todos esos dispositivos y herramientas son fundamentales en el desarrollo del proyecto, sin embargo, es de destacar la Jetson Xavier AGX que explicaremos a continuación junto con los demás dispositivos utilizados tanto a nivel hardware como software.

2.1 HARDWARE

2.1.1 Jetson Xavier AGX

Figura 2.1 Jetson AGX Xavier.

La Jetson AGX Xavier es un dispositivo creado fundamentalmente para dar un avance en la inteligencia artificial y por lo tanto a la automatización industrial. Es un dispositivo ideal para implementar inteligencia artificial avanzada y también para visión computarizada con gran precisión, lo que le permite operar de forma totalmente autónoma sin ninguna intervención humana.

Entre los elementos que componen la Jetson AGX Xavier podemos destacar los dos motores de acelerador de aprendizaje Profundo que descargan la inferencia de las redes neuronales convolucionales de

función fija, estos motores mejoran la eficiencia energética y liberan a la GPU para ejecutar redes más complejas y tareas dinámicas implementadas por el usuario. Como su nombre indica permiten acelerar el aprendizaje automático de un dispositivo combinando rendimiento alto y eficiencia energética.

Hoy en día, la Jetson AGX Xavier es uno de los dispositivos más utilizado en la inteligencia artificial. Eso es debido a su alto rendimiento que permite realizar aprendizajes automáticos ("machine learning"), manejar la odometría visual, la fusión de sensores, la localización y el mapeo, la detección de obstáculos y los algoritmos de planificación de caminos críticos para esta y la próxima generación de robots. [3]

En este trabajo el Jetson AGX Xavier lo utilizamos para implementar e ejecutar los aprendizajes automáticos necesarios para lograr los objetivos. Debido a su gran capacidad de cálculos su velocidad de cálculo y sus motores DLA tarda mucho menos en realizar dicha tarea que un computador normal. Tarda horas donde un computador tarda días en realizar un aprendizaje automático.

2.1.2 Raspberry Pi

Figura2.2 Raspberry Pi

La Raspberry Pi es la placa de un ordenador simple compuesto por un SoC, CPU, memoria RAM, puertos de entrada y salida de audio y vídeo, conectividad de red, ranura SD para almacenamiento, reloj, una toma para la alimentación y conexiones para periféricos de bajo nivel. La Raspberry Pi es un ordenador de bajo coste y tamaño

reducido, tanto es así que cabe en la palma de la mano. Se le puede conectar un monitor y un teclado para interactuar con ella exactamente igual que con cualquier otra computadora. Eso sí, no tiene interruptor para encenderlo o apagarlo**.** Para ponerlo en marcha tenemos que conectar periféricos de entrada y salida para poder interactuar como una pantalla, un ratón y un teclado y grabar un sistema operativo para la Raspberry Pi en la tarjeta SD [4].

En este trabajo se utiliza para conectarle una cámara y controlar las fotos desde la Raspberry Pi y por otro lado controlar el motor paso a paso que a su vez activa y desactiva a la cinta transportadora.

2.2.4 Brazo robótico

Figura 2.3 Brazo robótico Dobot Magician

Dobot magician es el brazo robótico que utilizamos en este trabajo, permite realizar interesantes funciones como la impresión

3D, el grabado láser y hacer tareas propias de un brazo robótico entre otras cosas. Soporta un desarrollo por medio de 13 interfaces extensibles y más de 20 lenguajes de programación, lo que nos permite con nuestra creatividad e imaginación darle mucha más funcionalidad. Dobot Magician tiene un buen desempeño tanto en el diseño de hardware como en la aplicación de software.

El brazo robótico se utiliza en este trabajo para manejar objetos, recogerlos y posicionarlos según la imagen, color o carta que tenemos.

2.2.5 Módulo de cámara para rapberry pi

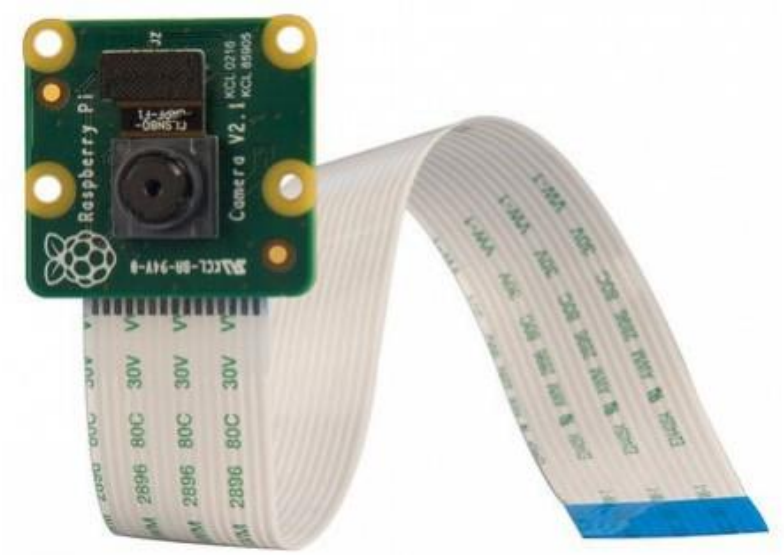


Figura 2.4 Módulo de cámara

Es un dispositivo adaptado para la Raspberry Pi y que tiene una tarea muy simple, la de realizar fotos. Para dicha tarea lo utilizamos en este trabajo; las fotos las guardamos directamente la Raspberry Pi para a posteriori tratar las imágenes.

2.2 SOFTWARE

2.2.1 Rest

El término REST ("Representational State Transfer") se originó en el año 2000, descrito en la tesis de Roy Fielding, padre de la especificación HTTP. Un servicio REST no es una arquitectura software, sino un conjunto de restricciones con las que podemos crear un estilo de arquitectura software, la cual podremos usar para crear aplicaciones web utilizando HTTP. REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON.

Hoy en día la mayoría de las empresas utilizan aplicaciones basados en REST para crear servicios. Esto se debe a que es un estándar lógico y eficiente para la creación de servicios web. Por poner algún ejemplo tenemos los sistemas de identificación de Facebook, la autenticación en los servicios de Google (hojas de cálculo, Google Analytics, …).

Las restricciones que definen a un sistema REST son los siguientes:

"Cliente-servidor": esta restricción mantiene al cliente y al servidor débilmente acoplados. Esto quiere decir que el cliente no necesita conocer los detalles de implementación del servidor y el servidor se "despreocupa" de cómo son usados los datos que envía al cliente.

"Sin estado": aquí decimos que cada petición que recibe el servidor debería ser independiente, es decir, no es necesario mantener sesiones.

"Cache": debe admitir un sistema de almacenamiento en caché. La infraestructura de red debe soportar una caché de varios niveles. Este almacenamiento evitará repetir varias conexiones entre el servidor y el cliente para recuperar un mismo recurso.

"Interfaz uniforme": define una interfaz genérica para administrar cada interacción que se produzca entre el cliente y el servidor de manera uniforme, lo cual simplifica y separa la arquitectura. Esta restricción indica que cada recurso del servicio REST debe tener una única dirección URI.

Sistema de capas: el servidor puede disponer de varias capas para su implementación. Esto ayuda a mejorar la escalabilidad, el rendimiento y la seguridad.

Las operaciones más importantes que nos permitirán manipular los recursos son cuatro: GET para consultar y leer, POST para crear, PUT para editar y DELETE para eliminar.

El uso de "hypermedia" (término que define el conjunto de procedimientos para crear contenidos que contengan texto, imagen, vídeo, audio y otros métodos de información) para permitir al usuario navegar por los distintos recursos de una API REST a través de enlaces HTML.

Para terminar, comentar que lo más importante a tener en cuenta al crear nuestro servicio basándonos en REST no es el lenguaje en el que se implemente, sino que las respuestas a las peticiones se hagan en XML o JSON, ya que lo que realmente nos interesa es poder intercambiar información entre dos dispositivos [5].

En este trabajo se utiliza las operaciones GET, POST e PUT para transferir informaciones entre los distintos dispositivos mediante el formato JSON utilizando la restricción de interfaz uniforme.

2.2.2 Node-red

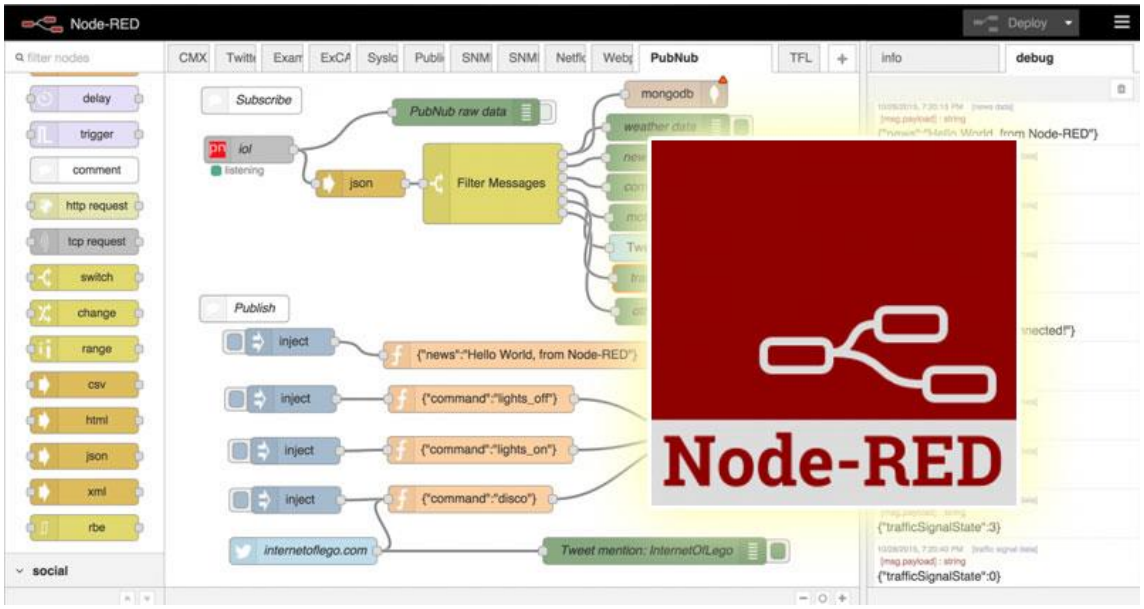


Figura 2.5 ejemplo de una aplicación a través de Node-Red

Node-RED es una herramienta visual de programación que nos permite conectar dispositivos de hardware, permitiendo de esa forma desarrollar aplicaciones complejas con varios dispositivos interconectadas de forma sencilla basado en la REST. El lenguaje de programación que se utiliza es Javascript, sin embargo una de las cosas más interesante que ofrece, razón por la cual lo utilizamos es este trabajo, es que permite desarrollar una aplicación sin escribir código sino interconectando simplemente los dispositivos con sus respectivos IPs e indicando las direcciones y nombre de los programas que queremos ejecutar mediante un diagrama de flujo. Es utilizado fundamentalmente para el internet de las cosas, pero también para cualquier aplicación en general.

Proporciona un editor basado en un navegador que facilita la conexión de los flujos utilizando la amplia gama de nodos de la paleta que pueden desplegarse en su tiempo de ejecución.

Aquí lo utilizamos para crear una aplicación en la que interconectamos todos los dispositivos mencionados anteriormente y realizar las tareas indicada con solo una orden desde dicha aplicación. Instalamos Node-RED en nuestro portátil.

2.2.3 Ubuntu

Es un sistema operativo de código abierto y gratuito basaba en Unix, además Ubuntu es altamente personalizable y tiene un centro de software lleno de aplicaciones. Entre los desarrolladores

es el S.O más utilizado debido a que, como hemos indicado antes, es de código abierto y de software libre. Permite a los desarrolladores elegir su código y mejorarlo para crear productos nuevos o realizar otras tareas.

En este proyecto la Jetson AGX Xavier tiene instalada Ubuntu 18.04 y además el Raspberry Pi utiliza un sistema, Debian, basado igual que Ubuntu en Unix. Por lo tanto, aparte de las ventajas dichos anteriormente también nos facilita la comunicación entre esos dispositivos [13].

2.2.4 OpenCV

La visión artificial es un subcampo de la inteligencia artificial, es fundamental dentro de la inteligencia artificial ya que dota a las máquinas la capacidad de ver imitando la visión humana. La visión artificial trabaja con imágenes que obtenemos a través de cámaras y posteriormente se procesan dichas imágenes con un software para extraer conclusiones. Dependiendo del software utilizado podemos obtener diferentes tipos de clasificaciones, según el color o forma del objeto presente en la imagen entre otras.

En este trabajo el software que utilizamos es OpenCV, es una librería que contiene varios módulos para realizar diferentes tareas, a continuación, definimos los fundamentales:

- OpenCv_core: contiene las funcionalidades principales de la biblioteca como son las estructuras de datos básicas y las funciones aritméticas.

- OpenCv_imgproc: contiene las principales funciones para manipulación de imágenes.

- OpenCv_highgui: contiene las funciones para la lectura y escritura tanto de imagen fija como de vídeo, así como otras funciones de interfaz de usuario.

- OpenCv_features2d: es un módulo con el que se consigue la detección de puntos y descriptores y además posee un "framework" para la unión de esos puntos.

- OpenCv_calib3d: módulo que se encarga de la calibración de la cámara; realiza la tarea de estimación de la geometría para vistas estereoscópicas.

- OpenCv_video: contiene la estimación de movimiento y, funciones y clases para el seguimiento de objetos y la extracción de fondos.

- OpenCv_objdetect: este módulo se encarga de la detección de objetos, como caras y personas.

- Por último, nombramos image procesing que utilizamos en este trabajo y que nos permite entre otras cosas poder detectar colores de una imagen en 4 pasos.
    1) Obtenemos la imagen y la convertimos a espacio de color HSV
    2) Después, creamos los rangos de los distintos colores que queremos detectar
    3) Se crean las máscaras para cada color con una función

Para terminar, obtenemos como salida, la imagen con los colores detectados marcados. En este trabajo marcamos el objeto con un circulo sobre la imagen e indicamos el color [12].

III. Aprendizaje automático ("machine learning")

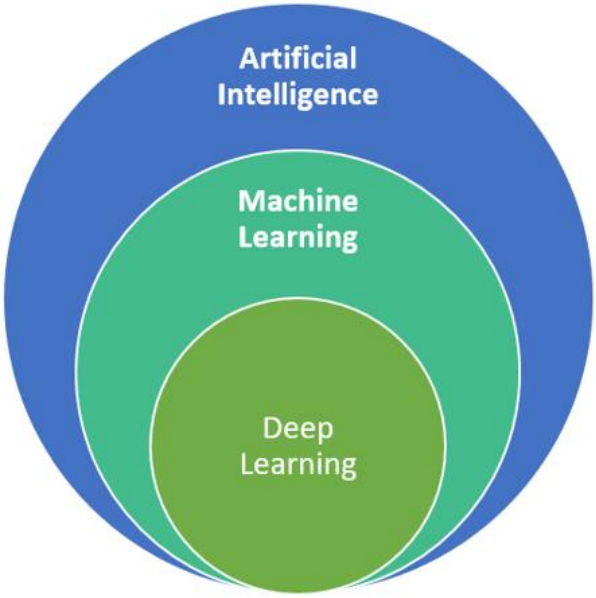


Figura 3.1 Relación entre machine learning, Deep learning e inteligencia artificial

El aprendizaje automático ("machine learning") es una derivación de inteligencia artificial que crea sistemas que aprenden de manera automatizada, es decir, identifican patrones complejos en millones de datos para predecir comportamientos futuros mediante algoritmos y que además de todo son capaces de mejorarse de manera independiente con el tiempo. El objetivo fundamental del aprendizaje automático es generar una regla, que permite predecir el futuro, anteriormente desconocidos mediante ejemplos conocidos.

En cuanto a los estudios. se puede decir que el aprendizaje automático es la mezcla de varias áreas de conocimientos como las estadísticas, las matemáticas y la ciencia de computación entre otras.

Existen dos tipos de aprendizaje automático ("machine learning"), supervisado y no supervisado; en este trabajo usamos el aprendizaje supervisado para lograr los objetivos. El aprendizaje supervisado es aquel aprendizaje al que previamente le han incluido etiquetas en la información y a base de dicha información detecta los objetos.

En el aprendizaje no supervisado, el algoritmo se modifica automáticamente, actualizándose cada vez que se reciben datos, por lo tanto, con el tiempo va mejorando. No contiene conocimiento a priori, es decir, ningún dato ha sido etiquetado, saca sus conclusiones de acuerdo al algoritmo tomando factores de manera indistinta que no tienen ni nombre, ni orden, ni marca.

La mayoría de las aplicaciones de aprendizaje automático son supervisadas y su secreto básicamente consiste en utilizar **aprendizaje profundo ("deep learning")** que funciona por medio de **redes neuronales** que imitan al cerebro humano usando hasta miles de millones de neuronas artificiales o unidades computacionales que se organizan en capas y cada capa aprende patrones una de la otra por lo que en conjunto se desarrollan patrones de definiciones, conducta, acciones, colores, objetos o simplemente luce algo.

El aprendizaje automático es muy importante en la vida del ser humano actualmente y se está desarrollando, se prevé que el futuro lo será todavía más importante. Toca a diferentes sectores y cada vez más:

- **Análisis de compradores y consumidores**:
  Hoy en día, tenemos la capacidad de hacer una previsión, mediante el aprendizaje automático, de los clientes que tiene una posibilidad de cerrar una venta o no de un determinado producto estudiando su comportamiento, entre otras cosas.
- **Seguridad de la empresa:**
  En una empresa se puede revisar las personas que entran y salen con reconocimiento e identificar si pertenece a dicha empresa o no, o verificar determinados materiales en toda una planta, entre otras cosas.
- **Conteo de piezas en video o imagen:**
  Mediante el aprendizaje automático, podemos hacer inventarios incluso si los objetos están distribuidos de forma aleatoria en un entorno, conocer todos los números en tiempo real y dar señales en caso necesario, por ejemplo, si se pasa de los límites permitidos.

- **Optimización de tiempos:**
  Podemos conseguir con el aprendizaje automático sistemas capaces de determinar cuándo realizar una tarea, cuándo parar, que tiempo se necesita y organizar plantas e industrias para que funciones de la forma más óptima posible.

- **Atención a clientes por medio de voz y texto:**
  Con el aprendizaje automático podemos estudiar las distintas preguntas y respuestas de los distintos clientes que llaman a una determinada empresa y crear una base y tener sistemas capaces de realizar el rol de atención al cliente totalmente.

- **Seguridad de la información y antifraudes:**
  Sobre todo, en internet donde se observa muchos fraudes, los avances del aprendizaje automático ayudan a reducirlos ya que hay sistemas capaces de detectar el comportamiento de posibles intrusos y podemos mediante patrones identificar ataques y alertar al personal antes que suceda.

3.1 Aprendizaje profundo ("deep learning")

Como hemos dicho anteriormente, el aprendizaje automático ("machine learning") supervisado utiliza básicamente el aprendizaje profundo ("deep learning"). Por lo tanto, podemos decir que el aprendizaje profundo ("deep learning") está incluido en el aprendizaje automático ("machine learning") que a su vez es parte de la inteligencia artificial.
Básicamente es un conjunto de algoritmos de aprendizaje automático que usan técnicas de redes neuronales convolucionales y que en cada capa procesan información útil para una siguiente capa de forma que es capaz de obtener como resultado un aprendizaje minucioso de la información de entrada y así poder extraer conclusiones relevantes.

El objetivo del aprendizaje profundo es concretamente el de describir descriptores suficientemente discriminantes para poder identificar el contenido de una imagen de forma automática. Para ello tenemos que tener en cuenta todos los factores que pueden influir en la percepción de una imagen como la luminosidad, los ángulos, resolución etc.

Para entender mejor el funcionamiento del aprendizaje profundo observamos la siguiente imagen.

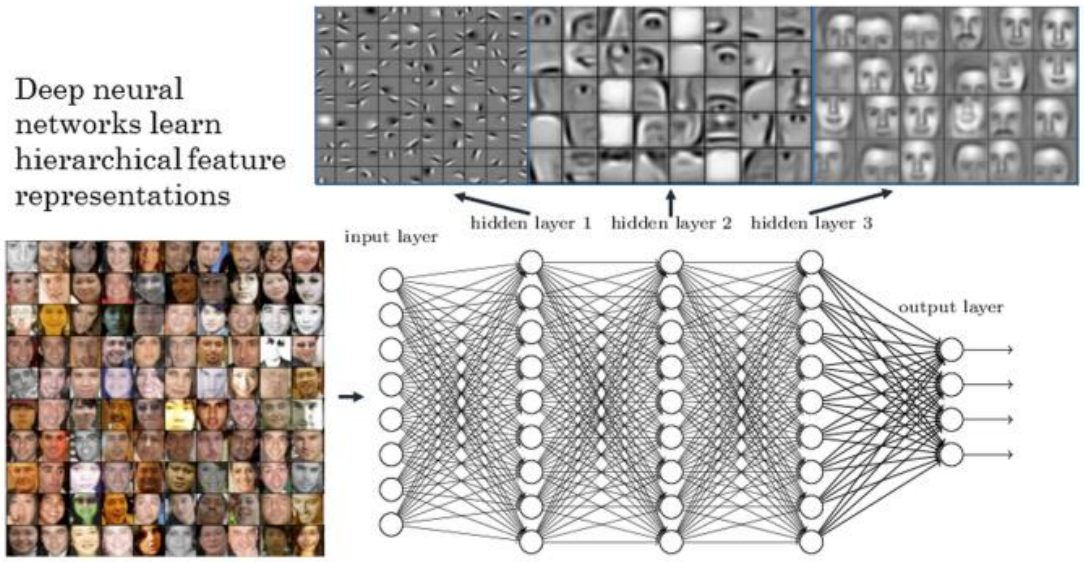


Figura 3.2 resumen aprendizaje profundo (“deep learning”)

Se observa que después de recibir la imagen pasa en una primera capa donde se divide la imagen en pixeles suficientes para poder identificar las partes, luego esa información pasa a una segunda capa donde se hace una primera clasificación observando partes de los seres (ojo, nariz, oreja, boca …). Esa información a su vez pasa también a una capa superior en la que se pueden reconocer caras. Se observa como el aprendizaje profundo (“deep learning”) en cada capa describe claramente discriminantes de forma automática que nos van llevando al objetivo final [6].

3.2 Redes neuronales convolucionales (CNN)

El aprendizaje profundo funciona por medio de redes neuronales convolucionales. El CNN es un tipo de red neuronal artificial que imita básicamente la visión del ser humano. Está constituido de varias capas especializadas y de forma jerárquica, desde capas que pueden detectar elementos sencillos de una imagen como líneas, curvas hasta capas capaces de detectar elementos más complejos, como rostros de animales entre otros.

El CNN necesita una en su entrada múltiples de imágenes para cada uno de los elementos que se quiere detectar para realizar el aprendizaje automático. Hay ejemplos en los que se necesitan procesar más de 10000 imágenes para llegar a los objetivos, por ejemplo, para detectar el rostro de animales, eso nos da una idea de la importancia de la velocidad del dispositivo que realiza el aprendizaje profundo ("deep learning"). Esa es la principal razón por la que trabajamos en este proyecto con la Jetson AGX Xavier, un dispositivo preparado para el aprendizaje profundo, en vez de con un computador corriente.

El CNN toma como entrada una matriz de pixeles de las imágenes, para tener el número de neuronas en la entrada multiplicamos los pixeles por los canales de color. Una vez obtenida esas entradas se hace un **pre-procesamiento** que consiste una transformación del valor cada pixel a un valor entre 0 y 1, dividiendo dicho valor por 255. Podemos observar un ejemplo de pre-procesamiento en la siguiente figura.

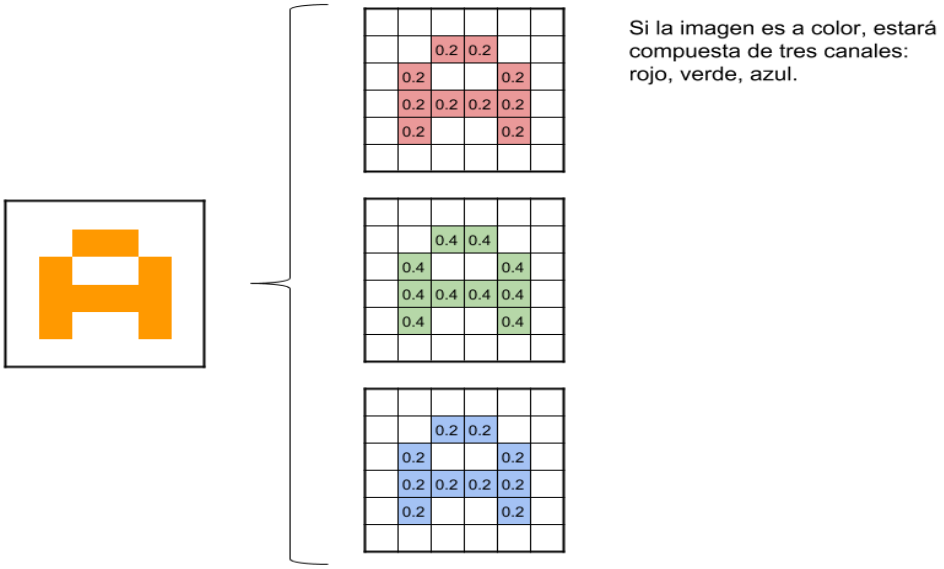


Figura 3.3 Transformación de pixel en valores entre 0 y 1

Después de pre-procesamiento, el siguiente paso son las **convoluciones** que consiste en generar una nueva matriz que es la entrada para la siguiente capa. Dicha matriz se obtiene recorriendo nuestra matriz de entrada en la capa y haciendo el

producto escalar con otra denominada matriz de "kernel" de 3x3 en este caso. Hay varias matrices de "kernel", por lo tanto, tenemos tantas matrices de salida como matrices tenemos. A esas matrices de salida se les aplica la función de activación RELU, f(x)=max (0, x).
En la siguiente imagen se puede observar un ejemplo de una convolución.

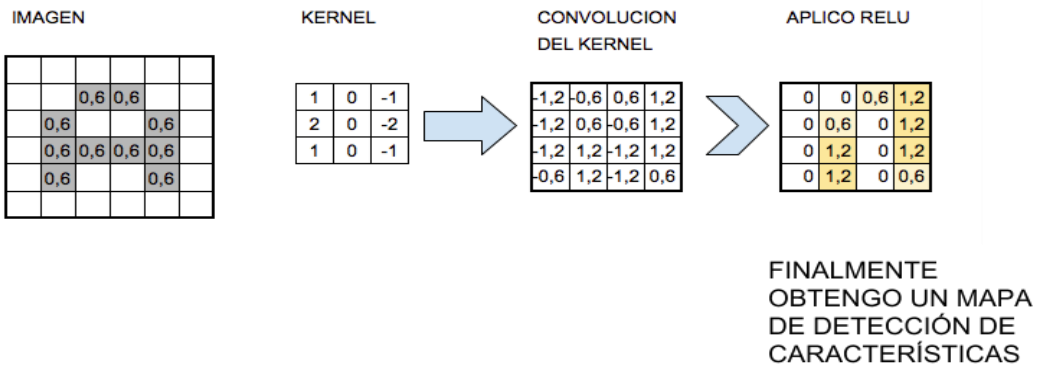


Figura 3.4 Ejemplo de convolución

Tras la convolución se observa que tenemos muchas neuronas, por lo tanto, es necesario reducir dicha cantidad antes de pasar a la siguiente capa (realizar la siguiente convolución). Dicho paso se denomina **submuestreo,** aquí lo más importante es quedarse con la información relevante. El submuestreo más utilizadoes el submuestreo con "max-pooling" de 2x2 que nos permiten reducir a la mitad las dimensiones de las matrices de entrada a la siguiente capa.

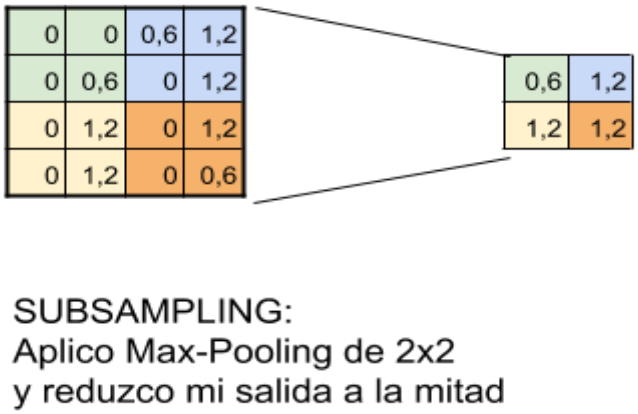


Figura 3.5 Ejemplo de submuestreo

En la siguiente figura se puede observar el ejemplo de los pasos que hacemos en una convolución de una imagen de 28x28 pixeles en la entrada.

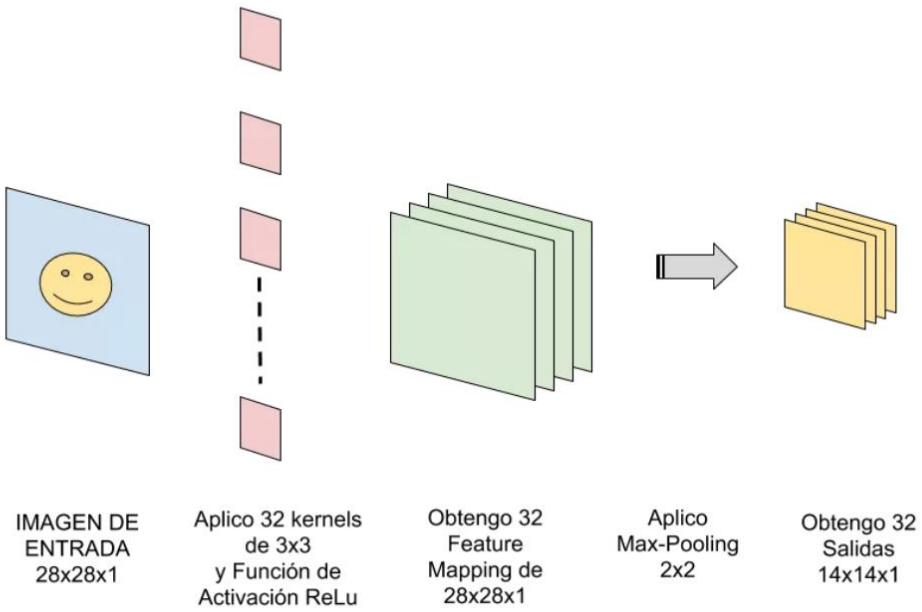


Figura 3.6 Pasos de convoluciones

Se realizará las convoluciones (capas) necesarias hasta tener una imagen de 3x3 pixeles y realizamos nuestra última convolución ya el matriz de "kernel" es de 3x3 teniendo nuestra última capa.

Tras la última convolución y el submuestreo, dicha capa se le realiza lo que denominamos aplanar y conectar a una capa de neuronas que consiste en pasarla de lo que se dice tridimensional en este caso a una capa de neuronas. Por último, a dicha capa de neuronas se le aplica la función llamada "**softmax"** que nos indica la probabilidad de que la figura, elemento o ser contenido en la imagen sea una cosa u otra (gato, perro, persona, coche…).

En la siguiente figura se puede observar un resumen de todos los procesos por los que pasa una imagen en un aprendizaje profundo mediante CNN [7].

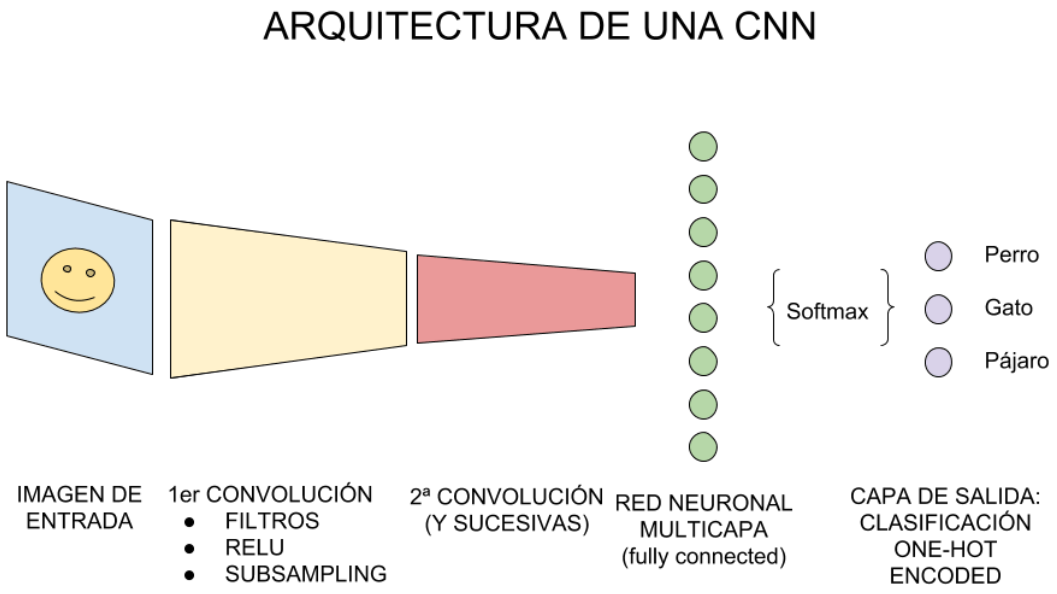


Figura 3.7 Pasos de una red neuronal convolucional

IV. Detección de objeto en una imagen usando Yolo

Una vez hecho el aprendizaje automático ("machine learning"), el dispositivo está capacitado para reconocer y clasificar si se encuentra en el futuro con uno de los objetos, personas, animales que ha aprendido a reconocer en una imagen. Sin embargo, Hay otros dos problemas para detectar un objeto que son la localización del objeto en una imagen y en caso de que la imagen tenga más de un objeto, se debe ser capaz de detectar todos los objetos.

El algoritmo Yolo ("you only look once") es un código abierto que hace uso de una red neuronal única para la detección de objetos de forma precisa y a tiempo real que nos permite solucionar los problemas de detección de objeto. Es capaz de localizar todos los objetos de una imagen dividiéndola en regiones que son sometidas a una predicción con el aprendizaje automático ya realizado y tras dicha predicción agrupar las regiones que pertenecen a un mismo objeto.

A continuación, se explica el funcionamiento de Yolo para detectar un objeto desde que recibe la imagen.

Yolo ("you only look once") mira la imagen solamente una vez, pero de forma inteligente, de ahí su nombre.

1. Tras mirar la imagen la divide en una cuadrícula de RxR.

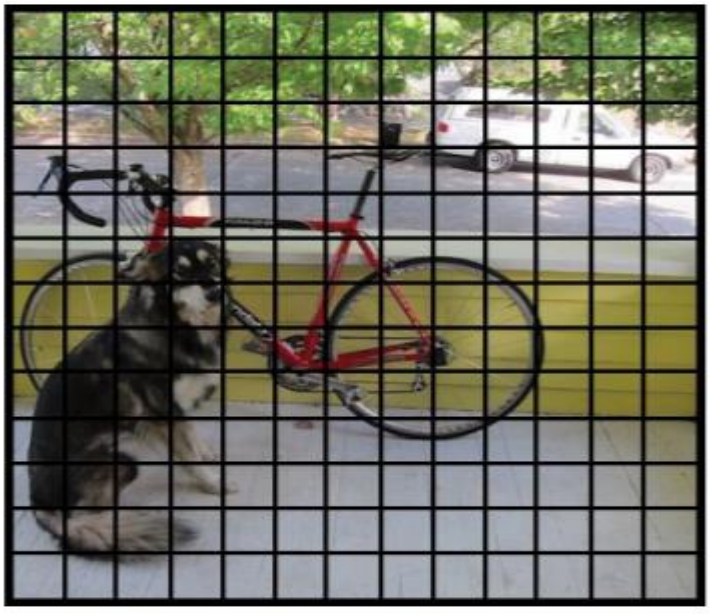


Figura 4.1 división de la imagen

2. En segundo lugar, se describe x rectángulos delimitadores que encierran los objetos mediante la búsqueda selectiva**:**

   La búsqueda selectiva consiste en un algoritmo que es capaz de detectar todos los posibles objetos en una imagen y encerrarlos con rectángulos, mediante segmentación y combinación de regiones.
   Primero divide la imagen en regiones muy pequeñas de pixeles y luego va juntando las regiones con más probabilidad de pertenecer a un determinado objeto ya sea por similitud de textura, de color, de intensidad luminosa etc. Así va comparando y juntando regiones cada vez más grandes hasta encontrar los posibles objetos en una imagen [8].

Figura 4.2 Pasos del selective search

3. Para cada una de las RxR celdas se ejecuta el **algoritmo de predicción de pertenencia o de localización** a un objeto que explicaremos a continuación. De esa ejecución obtenemos un mínimo de cinco salidas en forma de vector para cada uno de los x rectángulos delimitadores u objetos descritos, por lo tanto, tenemos un vector de longitud 5 * x.

   A continuación, se describen esas cinco salidas

   - **P_i** indica el nivel de confianza de que dicha celda pertenezca al i-ésimo objeto o rectángulo descrito cuyo centro y anchos coincide con bx, by, bw, bh.

   - bx_i, by_i, bw_i y bh_i pertenecen al i-ésimo objeto o rectángulo descrito. **bx_i** y **by_i** indican el centro de dicho objeto, **bw_i** y **bh_i** representan respectivamente el ancho y el alto del objeto o rectángulo descrito.

- También podemos obtener N otras salidas o características del objeto si queremos (C_1, …, C_N, en este caso, la salida es un vector de 5 * x * N para cada. Donde C_{ni} representa la característica n-ésima del i-ésimo objeto descrito.

$$y = \begin{bmatrix} p_1 \\ b_{x_1} \\ b_{y_1} \\ b_{w_1} \\ b_{h_1} \\ c_1 \\ . \\ . \\ c_n \\ p_2 \\ b_{x_2} \\ b_{y_2} \\ b_{w_2} \\ b_{h_2} \\ c_1 \\ . \\ . \\ c_n \end{bmatrix}$$

Figura 4.3 Vector de salida para cada celda

Y tenemos una imagen como la siguiente que nos confirma que los x rectángulos que tenemos están bien descritos, pero no nos dan información sobre qué objeto tenemos.

Figura 4.4 ejemplo de rectángulos descritos

4. Luego se ejecuta el algoritmo que detecta que clase de objeto (coche, gato, pero, caballo etc) tenemos en la imagen para cada uno de los x rectángulos descrito anteriormente, basándose en el aprendizaje automático ya ejecutado. En la salida tenemos para cada rectángulo descrito, todas las clases que hemos sometido al aprendizaje automático y su respetivo porcentaje o nivel de confianza de que el rectángulo pertenezca a esa clase.

   A este nivel tenemos otro problema ya que un mismo objeto lo pueden detectar varios rectángulos. Para solucionar ese problema primero eliminamos todos los rectángulos con una probabilidad inferior a un valor mínimo y después se aplica la técnica conocido como **"**non — max suppression".

   "Non — max suppression" es la técnica que nos permite detectar cada objeto una sola vez, Para eso nos ayudamos de las intersecciones entre los rectángulos **IOU** ("intersection over union") que detectan un mismo objeto y del rectángulo cuyo porcentaje es el superior.

   1. observamos el IOU de todos los rectángulos de detectan dicho objeto con respeto de mayor porcentaje.

   2. Todos los rectángulos cuyo IOU son superiores a 0,5 se eliminan, así eliminando a todo los tienen un grado alto de superposición [9].

5. Finalmente obtenemos el resultado de cada objeto detectado solamente una vez [10].

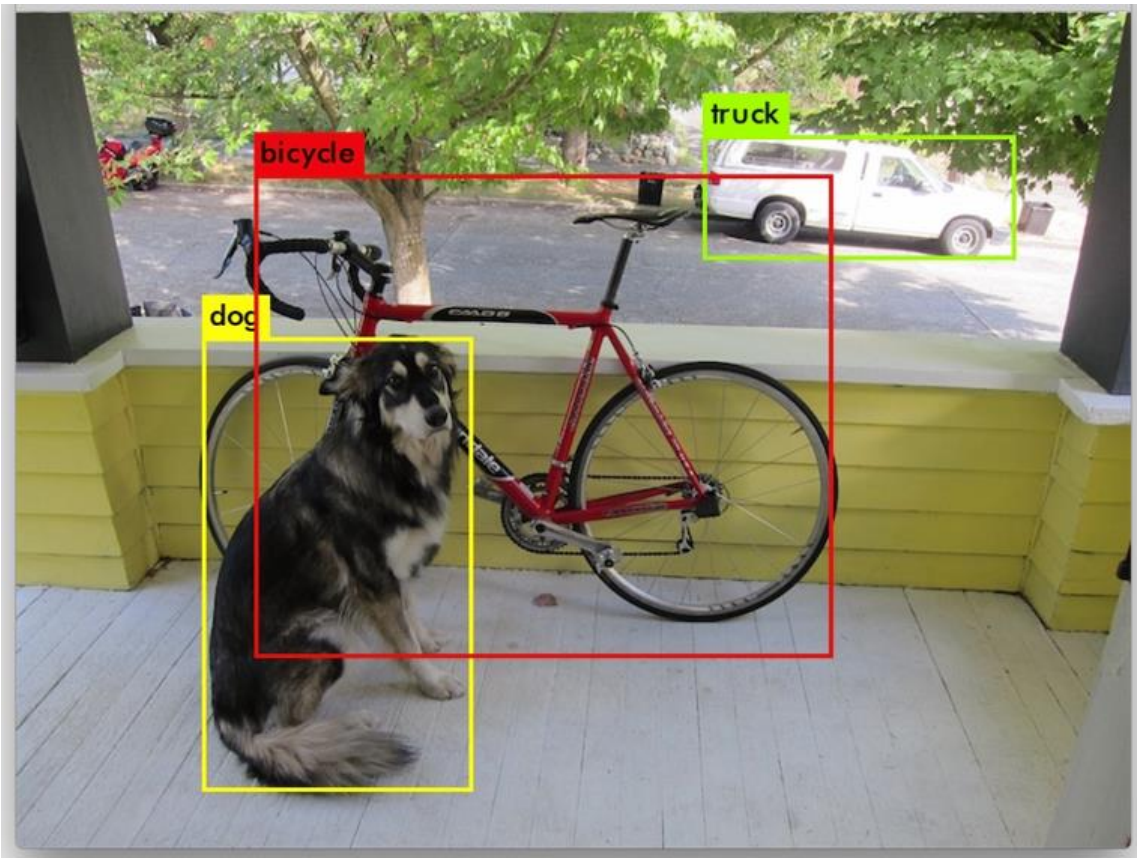


Figura 4.5 Resultado de la detección

V. Realización de la aplicación para el control de la planta

Figura 5.1 Planta (distintas a lo largo del trabajo)

Como hemos indicado anteriormente en este proyecto hacemos una aplicación mediante la cual controlamos una planta industrial al que aplicamos aprendizaje automático para hacerla más autónoma). En dicha planta recibimos un objeto a través de una cinta transportadora y clasificamos dicho objeto:

1. Según el color
2. Según sea una de los siguientes: caballo, perro, coche, camión, bicicleta o persona. Anteriormente se instala en la Jetson un pre-modelo de aprendizaje automático que permite detectar dichos objetos.

También se explica cómo podemos realizar un aprendizaje automático para cualquier objeto.

La planta la controlamos a través de una aplicación que he creado en Node-RED en la que utilizamos el protocolo de transferencia "HTTP" para la comunicación en dispositivos.

El funcionamiento de la planta se puede entender a través del siguiente esquema en el que se observa todos los pasos del dado u objeto desde que entra en la planta hasta su salida.

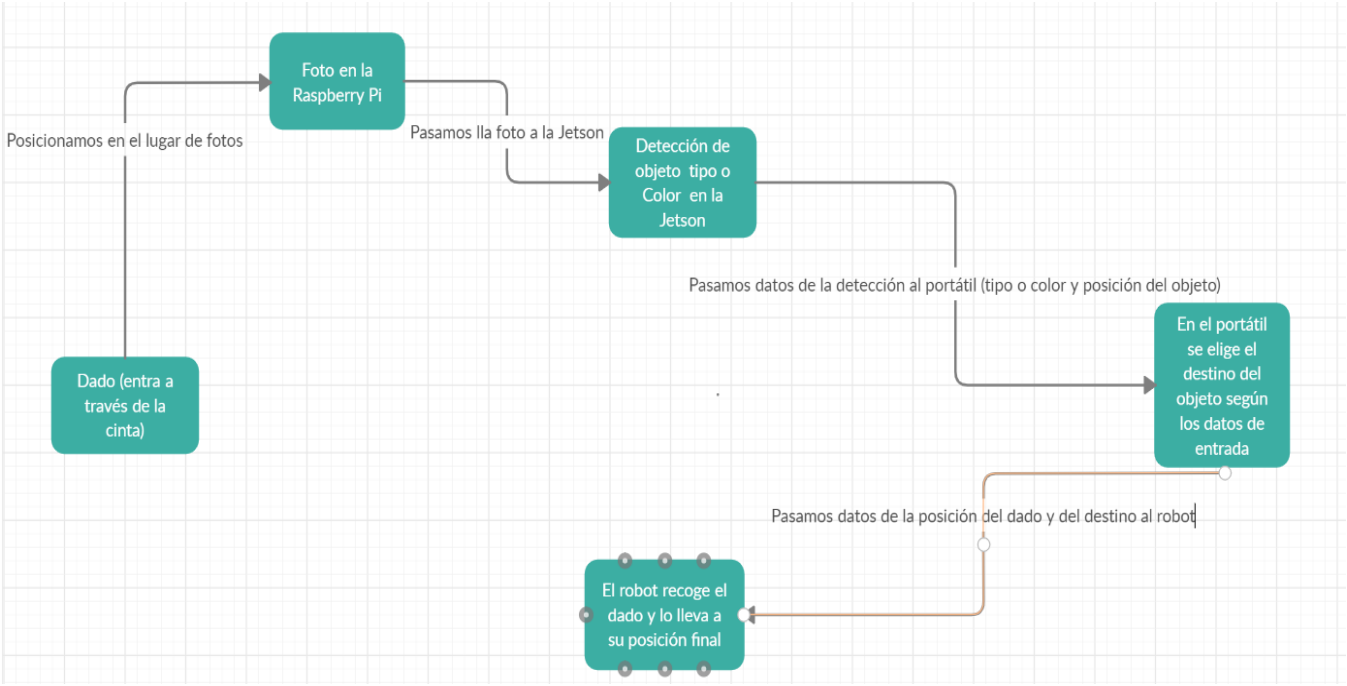


Figura 5.2 Resumen de las distintas fases que va pasando el objeto en la planta

5.1 Cinta trasportadora

A través de esa cinta, que percibimos en la imagen Figura 4.5, nos llega nuestro objeto que puede ser en este caso un dado con un color diferente en cada cara, un dado con uno de las siguientes imágenes: caballo, perro, coche, camión, bicicleta, persona en cada cara o un color o una carta.

Primero arrancamos la cinta con un programa sencillo en Python de denominamos start en este caso. Después podemos elegir si la cinta gira por un sentido u otro dependiendo da la posición de la cinta en la planta para que el objeto caiga en la zona querida, esos programas se denominan "clockwise" y "conterclockwise", y para parar la cinta en cualquier momento tenemos un programa llamado "stop".

Todos esos programas descritos a su vez son parte de otro programa más grande denominado "conveyor". El programa es el que ejecutamos y una vez en ejecución podemos llamar todos los demás relacionadas con la cinta a través de ella.

5.2 Clasificación según el color del objeto

En esta parte del proyecto utilizamos básicamente la visión artificial mediante el software OpenCV para dar la capacidad de poder reconocer distintos colores a nuestro dispositivo Jetson AGX Xavier y clasificarlos.

Lo vamos a explicar basándonos en la aplicación denominada planta.json creado en Node-RED que podemos observar en la imagen Figura 5.3, en esta aplicación utilizamos distintos protocolos de REST para la comunicación entre los dispositivos y enviamos las informaciones en formato json.

Antes de todo ejecutamos los siguientes programas que hemos desarrollados en los distintos dispositivos para el correcto funcionamiento da la aplicación, para que una vez reciban información a través de la aplicación se pueda llevar al cabo tarea deseada

- Conveyor en la Raspberry Pi para la cinta
- Photo en la rasberry pi también para poder realizar la foto
- Decoder_color en la Jetson AGX para poder reconocer el color imagen
- Dobot-service2 en el portátil para poder dar órdenes al robot.
- Position_dice_OpenCv o position_dice_OpenCv_distance (solamente puede estar en ejecución una de las dos) en la computadora para controlar el movimiento del robot.
  Los dos programas, position_dice_OpenCv o position_dice_OpenCv_distance, tienes la misma función: indicarle al robot en qué posición está el objeto.
  La diferencia es que con la position_dice_OpenCv hacemos un control en bucle abierto en la que cambiamos directamente el sistema de coordenadas de la imagen recibida al sistema de coordenada del robot mediante cálculos que permiten tener la posición exacta del robot.
  Con la position_OpenCv_yolo_distance hacemos un control en bucle cerrado en la que el robot se va

acercando al objeto con la información que recibe de position_dice_OpenCv_distance, hace una foto y devuelve esa información a la position_dice_OpenCv_distance que va calculando la distancia del robot con el objeto. Así hasta llegar a una distancia máxima permitida, de esa forma el robot encuentra la posición del objeto.

A continuación, procedo a explicar el funcionamiento de la aplicación para entender la detección del color del objeto basándome en la aplicación siguiente:

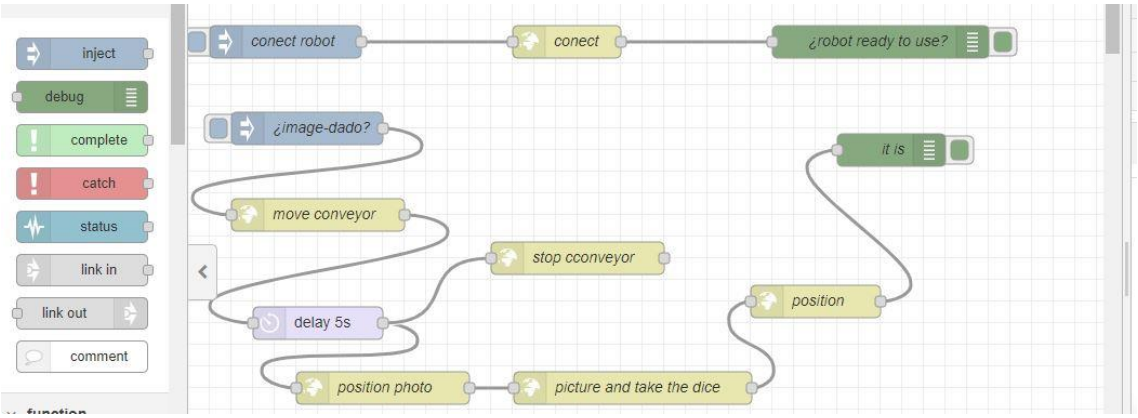


Figura 5.3 aplicación para detectar objetos o colores del dado

Por un lado, tenemos el botón conect robot que arranca el robot simplemente para que pueda ejecutar las ordenes que le llegan. Para ellos se envía automáticamente a través de la función conect "http://localhost:8000/dobot/api/v1.0/connect" desde la aplicación hasta el programa Dobot-service2 del robot que se encarga de ejecutar la orden.

Por otro lado, podemos observar un botón de entrada image-dado que pulsamos para arrancar el funcionamiento de la planta indicando en este caso que queremos identificar el color del dado. La pulsación de ese botón significa que ha llegado un objeto en la planta, todo lo demás se hace de forma automática para obtener el resultado final.

1. Se mueve la cinta transporta durante 5 segundos, tiempo suficiente para llevar el objeto a su destino final. Para ello, enviamos al Raspberry Pi una señal "conterclockwise" mediante la función move conveyor que se recibe en el programa conveyor para ejecutar el programa conterclockwise. Si posicionamos la cinta diferente podemos

ejecutar el programa clockwise que realiza la misma tarea lo único que cambia es el sentido de giro de la cinta.

2. Después se para la cinta enviando la señal "stop" al Raspberry Pi mediante la función stop conveyor y el robot se posiciona para realizar la foto ya que el robot lleva la foto junto con la herramienta para escoger el objeto. Para ellos desde la aplicación se envía la señal "position-photo" a través de la función position photo que recibe el programa position_dice_OpenCv o position_dice_OpenCv_distance desde el computador dependiendo de cuál está en ejecución, que a su vez ordena al robot que se mueve a una posición estándar desde la que hacemos la foto.

3. En tercer lugar, se envía al programa position_dice_OpenCv o position_dice_OpenCv_distance una señal "coger-dado" mediante la función picture and take the dice. Una vez esa señal enviada se hace varias cosas de forma interna entre los programas:

    - Primero se envía una señal "decoder" al programa decoder_color de la Jetson AGX.

    - El programa decodor_color a su vez manda una señal "foto" al Raspberry Pi para realizar la foto, una vez realizada la foto el programa decodor_color lo recibe en formato json en la Jetson AGX.

    - Una vez recibido la foto en la Jetson se procede a detectar el color a través del programa detect_color1 en la que usamos OpenCv. Y recibimos el color del objeto y su localización.

    - Esa información obtenida se recibe en formato json en el programa position_dice_OpenCv o position_dice_OpenCv_distance.

    - Finalmente se pasa la información de la localización del objeto al robot y se le ordena ir a cogerlo y quedarse a una posición intermedia.

Figura 5.4 robot en movimiento tras coger el dado

4. Por último, mandamos a través de la aplicación la señal "position-dado" mediante la función position que se recibe en el programa position_dice_OpenCv o position_dice_OpenCv_distance, a través de ese programa se indica al robot de llevar el objeto detectado a su posición final dependiendo del color (en la aplicación ponemos los colores en inglés) que hemos detectado y en la aplicación recibimos como salida el color que se ha detectado.

Figura 5.5 Robot posición el dado en su posición final según color

Figura 5.6 salida del color en la aplicación

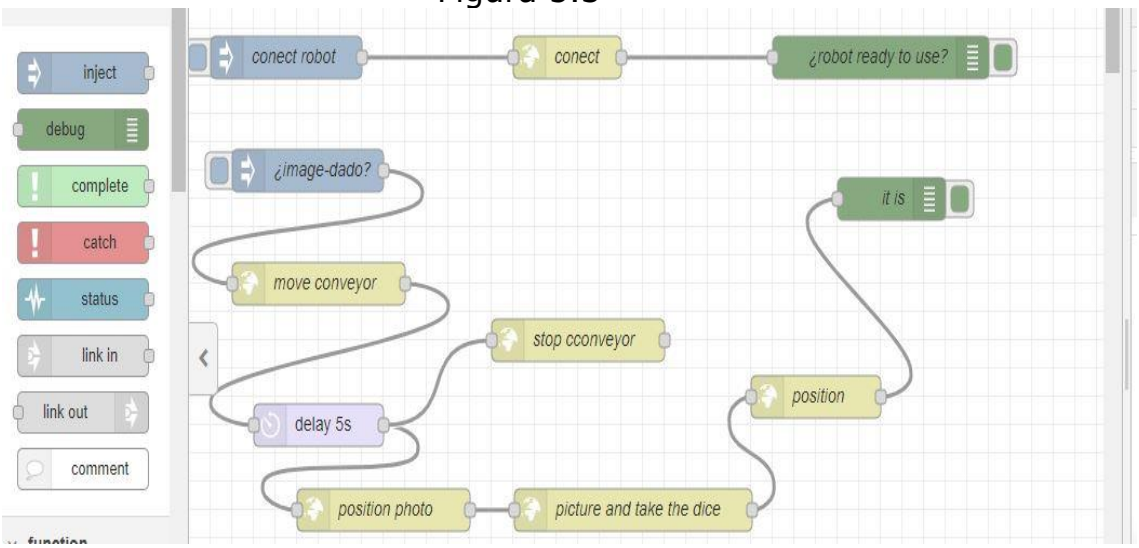
5.3 Clasificación según sea caballo, perro, coche, camión, bicicleta o persona

En esta parte del proyecto utilizamos básicamente un aprendizaje automático ya hecho que instalamos en la Jetson. Para ello utilizamos un pre-modelo yolo3.weights que descargamos a través de darknet que es un marco de red neural de código abierto escrito en C y CUDA. Aquí lo compilamos con CUDA para procesar las imágenes mucho más rápido. Por lo tanto, primero instalamos darnet y después podemos utilizar la subrutina yolo para detectar varios objetos o animales mediante una simple llamada, como se puede observar en la siguiente imagen, indicando el pre-modelo. Entre esos objetos e animales que podemos detectar, es decir que están sometido al aprendizaje automático en el pre-modelo yolo3.weights están todos los que queremos clasificar en este apartado del trabajo (caballo, perro, coche, camión, bicicleta o persona) [11].

Figura 5.7 comando para detectar con el pre-modelo yolov3.weights

En cuanto al procedimiento para detectar la imagen que tenemos en la cara del dado es exactamente igual que para detectar un color explicado en el apartado anterior. De hecho, se utiliza la misma aplicación planta.json que podemos observar en la imagen Figura 5.3. Siendo la diferencia que en la Jetson en vez de ejecutar el programa position_dice_OpenCv o position_dice_OpenCv_distance se ejecuta position_dice_yolo o position_dice_yolo_distance y en vez de decoder_OpenCv se ejecuta decoder_image que nos permiten detectar si la imagen que tenemos es un caballo, perro, coche, camión, bicicleta o persona, en vez de detectar el color.

Figura 5.3



- Por lo tanto, igual que el apartado anterior empezamos por arrancar los distintos programas que interconectan desde la aplicación en los distintos dispositivos:
    - Conveyor en la Raspberry Pi para la cinta
    - Photo en la rasberry pi también para poder realizar la foto
    - Decoder_image en la Jetson AGX para poder reconocer el color imagen
    - Dobot-service2 en la computadora para poder dar órdenes al robot.
    - Position_dice_yolo o position_dice_yolo_distance (solamente puede estar en ejecución una de las dos) en la computadora para controlar el movimiento del robot.
      Los dos programas, position_dice_OpenCv o position_dice_OpenCv_distance, tienes la misma función el de indicarle al robot en qué posición está el objeto.
      La diferencia es que con la position_dice_OpenCv hacemos un control en bucle abierto en la que cambiamos directamente el sistema de coordenadas de la imagen recibida al sistema de coordenada del robot mediante cálculos que permiten tener la posición exacta del robot.

Con la position_dice_yolo_distance hacemos un control en bucle cerrado en la que el robot se va acercando al objeto con la información que recibe de position_dice_yolo_distance, hace una foto y devuelve esa información a la position_dice_yolo_distance que va calculando la distancia del robot con el objeto. Así hasta llegar a una distancia máximas permitidas, de esa forma el robot encuentra la posición del objeto.

- Después de arrancar los programas, arrancamos el robot con el botón connect robot de la aplicación nuestro robot.

- A continuación, tenemos el botón "¿image-dado?" que pulsamos al llegar el dado en la planta.

1. Se mueve la cinta transporta durante 5 segundo, tiempo suficiente para llevar el objeto a su destino final. Para ello, enviamos al Raspberry Pi una señal "conterclockwise" mediante la función move_conveyor que se recibe en el programa conveyor para ejecutar el programa conterclockwise. Si posicionamos la cinta diferente podemos ejecutar el programa clockwise que realiza la misma tarea lo único que cambia es el sentido de giro de la cinta.

2. Después se para la cinta enviando la señal "stop" al Raspberry Pi mediante la función stop conveyor y el robot se posiciona para realizar la foto ya que el robot lleva la foto junto con la herramienta para escoger el objeto. Para ellos desde la aplicación se envía la señal "position-photo" a través de la función position photo que recibe el programa position_dice_yolo o position_dice_yolo_distance desde el computador dependiendo de cuál está en ejecución, que a su vez ordena al robot que se mueve a una posición estándar desde la que hacemos la foto.

3. En tercer lugar, se envía al programa position_dice_yolo o position_dice_yolo_distance una señal "coger-dado" mediante la función picture and take the dice. Una vez esa

señal enviada se hace varias cosas de forma interna entre los programas:

- Primero se envía una señal "decoder" al programa decoder_image de la Jetson AGX.

- El programa decoder_image a su vez manda una señal "foto" al Raspberry Pi para realizar la foto, una vez realizada la foto el programa decoder_image lo recibe en formato json en la Jetson AGX.

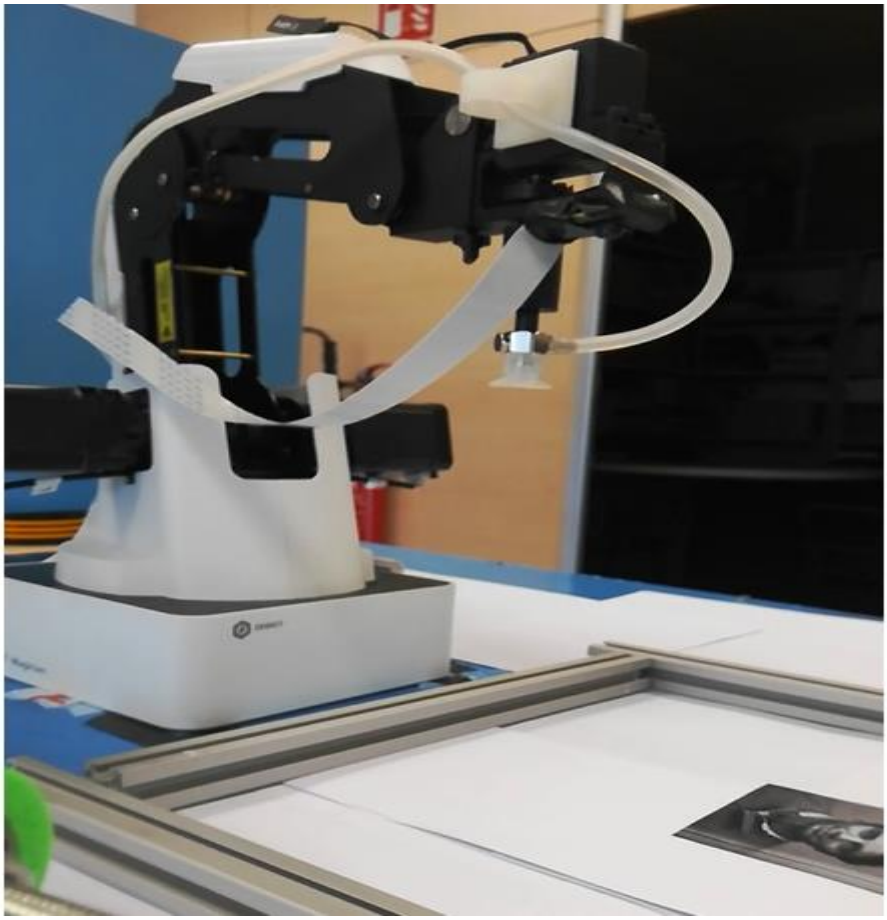


Figura 5.8 robot en posición de foto para reconocer la imagen

- Una vez recibido la foto en la Jetson se procede a detectar el tipo de objeto a través del programa detect en la que usamos yolo. Y recibimos el nombre del tipo de objeto presente en la imagen y su localización.

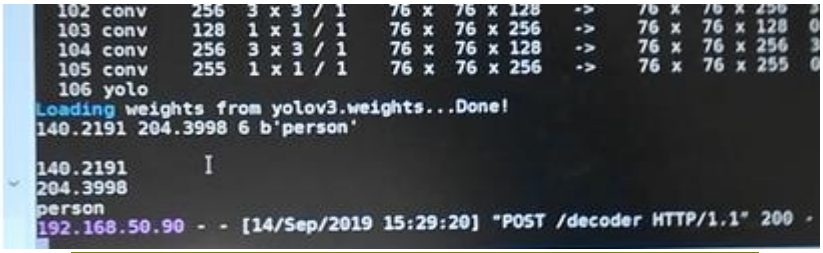


Figura 5.9 salida del Jetson tras localizar y reconocer el imagen

- Esa información obtenida se recibe en formato json en el programa position_dice_yolo o position_dice_yolo_distance.

- Finalmente se pasa la información de la localización del objeto al robot y se le ordena ir a cogerlo y quedarse a una posición intermedia.

Por último, mandamos a través de la aplicación la señal "position-dado" mediante la función position que se recibe en el programa position_dice_yolo o position_dice_yolo_distance, a través de ese programa se indica al robot que lleve el objeto detectado a su posición final dependiendo del tipo que hemos detectado y en la aplicación recibimos como salida el tipo de objeto que se ha detectado. Tenemos los nombres en inglés en la aplicación: horse, cat, dog, truck, car o person.

Figura 5.10 salida del tipo en la aplicación

5.4 Aprendizaje automático de objetos presente en una imagen

En este último apartado se va proceder a explicar cómo se entrena la Jetson AGX Xavier para la detección de un objeto utilizando "tensorflow-gpu".

Tensorflow es una biblioteca de código abierto para aprendizaje automático capaz de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos. Por lo tanto, se puede decir que realmente es el encargado que realiza el aprendizaje automático para nuestra Jetson Xavier. El Tensorflow-gpu realiza las tareas anteriormente descrito y nos permite ahorrar mucho tiempo ya que realiza el aprendizaje automático con mucho menos tiempos comparado con otros [14].

por otro lado, instalamos la herramienta LabelImg que nos permite guardar los datos en punto XML.

Por lo tanto, lo primero que haremos es instalar el tensorflow-gpu, luego se descarga el paquete Python necesario para su correcto funcionamiento. Dicho paquete es el Faster-RCNN-Inception-V2-COCO**,** se utiliza el faster-rnn para la detección de rostros y también para el reconocimiento de rostros. La arquitectura de inception V2 se utiliza debido a que tiene una alta precisión entre las arquitecturas de las redes neuronales convolucionales y COCO es una base de datos que tiene como objetivo localizar los puntos o zonas claves para la detección del objeto.

Después, situamos todos los scripts y ficheros necesarios dentro de un directorio del tensorflow. Para ellos creamos:

- Una carpeta de imágenes que contiene un fichero para las imágenes de prueba. Una vez terminado el entrenamiento, lo utilizamos para probar el correcto funcionamiento. También contiene otro fichero para las imágenes de entrenamiento.
- Y otra carpeta entreno para los archivos de configuración del modelo y los mapas de etiquetas que explicaremos posteriormente en este apartado.

Una vez terminado las instalaciones estamos listos para lanzar nuestro entrenamiento.

Para el entrenamiento lo primero que haremos es realizar un cierto número de fotos de cada tipo de objeto de queremos detectar, en varias posiciones, oscuridades, entorno entre otras cosas diferentes y

las guardamos en los ficheros de prueba (alrededor del 30%) y de entrenamiento (alrededor de un 80%) nombrados anteriormente. La herramienta LabelImg nos permite guardar en un archivo en formato punto XML las imágenes de los objetos que queremos identificar en cada foto mediante recortes rectangulares que permite la herramienta y le asociamos el nombre del objeto en cada recortado. Esta tarea se realiza manualmente; por lo tanto tarda mucho tiempo ya que para cada foto puede incluir varios objetos. Este archivo en formato XML es el que se utiliza para ejecutar un script que nos permite obtener una etiqueta de entrenamiento en un archivo en formato CSV y otra de prueba con el mismo formato en la carpeta imágenes. Con dichos archivos en formato CSVs creamos mediante un comando lo que se denomina archivo TFRecord, un archivo TFRecord almacena datos como una secuencia de cadenas binarias, que utilizamos como entrada para el entreno o aprendizaje con el tensorflow.

Luego se crea el mapa de etiquetas dentro de la carpeta de entreno que indica lo que es cada objeto definiendo un mapa de nombres de clases con números de identificaciones de las clases, es un archivo en formato TXT.

Por último y antes de ejecutar en aprendizaje automático, hay que configurar, en la carpeta entreno, el entreno de detección de objetos definiendo qué modelo y qué parámetros se utilizarán y las rutas de los archivos [15].

VI. Conclusión

En conclusión, podemos indicar que hemos logrado el objetivo propuesto en este trabajo, ya que hemos logrado simular una planta simplificada pero totalmente autónoma desde que entra una pieza u objeto hasta su salida, utilizando aprendizaje automático, detección de objetos y distintos tipos de comunicaciones entre los dispositivos mediante cable, es decir medio físico.

En cuanto al impacto de este trabajo, puede con algunas mejorar llevarse en la vida real y así automatizar plantas de producciones enteras, lo cual es algo muy importante en las industrias ahora mismo debido a que baja los costes y lo será todavía más importante en el futuro debido a los avances que espera del IoT.

Terminaré este trabajo proponiendo dos posibles mejoras sobre le simulación de la planta:

Se puede incluir sensores de tal forma que no hay una entrada concreta, sino que se detecta la presencia de un objeto en la planta y el robot lo recoge directamente. Después, se verifica si es el objeto esperado y se somete al procedimiento para detectar el tipo o color. En caso contrario se considera una pieza caída y se aparta dicha pieza. Se lograría una mejora importante en la organización de la planta y sobre todo es un paso más para la automatización ya que es un error que la planta corrige de forma autónoma.

Se puede realizar el mismo trabajo utilizando conexiones no físico entre los dispositivos, por ejemplo, mediante wifi o bluetooth. Permitiendo una mejora significativa para organizar la planta y un ahora en cables e incluso su conexión con el exterior. Por otro lado, estos tipos de conexiones son fundamentales para conectar la planta con otros.

VII. Anexos

Aquí incluimos los distintos códigos empleados para realizar este trabajo.

7.1 Códigos para el funcionamiento de la Cinta transportadora

- **Start** (para activar la cinta)

```
import RPi.GPIO as gpio
import sys
import time


gpio.setmode(gpio.BOARD)
#define los pines que vamos a usar como salidas
gpio.setmode(gpio.BOARD)
steppins =[7,11,13,15] # pines de raspberry
print ('enciende cinta')
for pin in steppins:
   gpio.setup(pin, gpio.OUT)
   gpio.output(pin, True)
   time.sleep(1)
```

- **Stop** (para activar la cinta)

```
import RPi.GPIO as gpio
import sys
import time


gpio.setmode(gpio.BOARD)
#define los pines que vamos a usar como salidas
gpio.setmode(gpio.BOARD)
steppins =[7,11,13,15]  # pines de raspberry
print ('stop')
for pin in steppins:
   gpio.setup(pin, gpio.OUT)
   gpio.output(pin, False)
   time.sleep(1)
```

- **Clockwise** (para mover la cinta en sentido horario)

```
#importamos las libreria necesaria
import RPi.GPIO as gpio
import time
import sys

gpio.setmode(gpio.BOARD)
#define los pines que vamos a usar como salidas
gpio.setmode(gpio.BOARD)

steppins =[7,11,13,15]   # pines de raspberry

for pin in steppins:
    gpio.setup(pin, gpio.OUT)
    gpio.output(pin, False)

# sequencia a seguir
seq = [
[1,0,0,0],
[1,1,0,0],
[0,1,0,0],
[0,1,1,0],
[0,0,1,0],
[0,0,1,1],
[0,0,0,1],
[1,0,0,1]]


stepcount = len(seq)
stepdir = -1 # sentido antihorario 1, horario -1

#inicializar contador
stepconter=0

while True:

    for pin in range(0,4):
        xpin = steppins[pin]  #escogemos el pin que vamos a
activar

        if seq[stepconter][pin]!=0:
            gpio.output(xpin,True)
        else:
            gpio.output(xpin,False)

    stepconter = stepconter + stepdir

    if (stepconter >=stepcount):
        stepconter = 0

    if (stepconter < 0):
        stepconter = stepcount + stepdir

    time.sleep(0.001)
```

- **Conterclockwise** (para mover la cinta en sentido anti-horario)

```
#importamos las libreria necesaria
import RPi.GPIO as gpio
import time
import sys

gpio.setmode(gpio.BOARD)
#define los pines que vamos a usar como salidas
gpio.setmode(gpio.BOARD)

steppins =[7,11,13,15]  # pines de raspberry

for pin in steppins:
    gpio.setup(pin, gpio.OUT)
    gpio.output(pin, False)

# sequencia a seguir
seq = [
[1,0,0,0],
[1,1,0,0],
[0,1,0,0],
[0,1,1,0],
[0,0,1,0],
[0,0,1,1],
[0,0,0,1],
[1,0,0,1]]


stepcount = len(seq)
stepdir = 1 # sentido antihorario 1, horario -1

#inicializar contador
stepconter=0

while True:

    for pin in range(0,4):
        xpin = steppins[pin]  #escogemos el pin que vamos a
activar

        if seq[stepconter][pin]!=0:
            gpio.output(xpin,True)
        else:
            gpio.output(xpin,False)

    stepconter = stepconter + stepdir

    if (stepconter >=stepcount):
        stepconter = 0
```

```
    if (stepconter < 0):
        stepconter = stepcount + stepdir

    time.sleep(0.001)
```

- **Conveyor** (da la dirección del programa y ip del dispositivo que lo ejecuta al lanzar cualquiera de programas anteriores en este apartado)

```
import os
import subprocess
import time
from flask import Flask,jsonify,abort,request
import threading
import signal
import json

app = Flask(__name__)
port=8000
deployhost='0.0.0.0'

aux=0
aux1=0
aux2=0

@app.route('/start',methods=['POST'])
def start():

   global aux1
   global aux2
   global aux

   if (aux1>0):
         os.killpg(aux1, signal.SIGTERM)
         aux1=0
   if (aux2>0):
         os.killpg(aux2, signal.SIGTERM)
         aux2=0
   proc = subprocess.Popen("python3 start.py",shell= True,
preexec_fn=os.setsid)
   aux = proc.pid
   return 'encender motor'

@app.route('/clockwise',methods=['POST'])
def horario():

   global aux1
   global aux2
   global aux

   if (aux1>0):
         os.killpg(aux1, signal.SIGTERM)
         aux1=0
```

```
   if (aux>0):
         os.killpg(aux, signal.SIGTERM)
         aux=0

   proc = subprocess.Popen("python3 clockwise.py",shell= True,
preexec_fn=os.setsid)
   aux2 = proc.pid
   print ('pid2: ' + str(aux2))
   return 'movimiento horario'


@app.route('/conterclockwise',methods=['POST'])
def antihorario():

   global aux1
   global aux2
   global aux

   if (aux2>0):
         os.killpg(aux2, signal.SIGTERM)
         aux2=0
   if (aux>0):
         os.killpg(aux, signal.SIGTERM)
         aux=0

   proc = subprocess.Popen("python3 conterclockwise.py",shell=
True, preexec_fn=os.setsid)
   aux1 = proc.pid
   print ('pid1: ' + str(aux1))
   return 'movimiento antihorario'

@app.route('/stop',methods=['POST'])
def terminar():
   global aux1
   global aux2
   global aux
   print (aux1)

   print (aux2)
   if (aux1>0):
         print ('pid1: ' + str(aux1))
         os.killpg(aux1, signal.SIGTERM)
         proc = subprocess.Popen("python3 stop.py",shell= True,
preexec_fn=os.setsid)
         aux1=0


   if (aux2>0):
         print ('pid2: ' + str(aux2))
         os.killpg(aux2, signal.SIGTERM)
         proc = subprocess.Popen("python3 stop.py",shell= True,
preexec_fn=os.setsid)
         aux2=0

   if (aux>0):
```

```
        print ('pid0: ' + str(aux))
        os.killpg(aux, signal.SIGTERM)
        proc = subprocess.Popen("python3 stop.py",shell= True,
preexec_fn=os.setsid)
        aux=0

   return 'se apaga la cinta'


if __name__ == '__main__':
   app.run(host='0.0.0.0',port=8000,debug=True)
```

7.2 Códigos para realizar fotos en la Raspberry Pi

- **Foto** (para realizar la foto)

```
import picamera
import time

with picamera.PiCamera() as picam:
   picam.start_preview()
   time.sleep(1)
   picam.capture('photo_'+'.jpg')
   picam.stop_preview()
   picam.close()
```

- **Photo** (para ordenar la raspberry hacer la foto)

```
import threading
import signal


app = Flask(__name__)
port = 8000
deployhost = '0.0.0.0'

data = {}

@app.route('/foto',methods=['POST'])
def photo():
   #os.killpg(aux, signal.SIGTERM)
   proc = subprocess.Popen("python3 foto.py", shell=True,
preexec_fn=os.setsid)
   #aux = proc.pid
   global data

   proc.wait()

   with open('photo_.jpg',mode='rb') as f:
        img = f.read()
```

```
    data['img'] = base64.b64encode(img)
    #print(json.dumps(data))

    return jsonify(data)

if __name__== '__main__':
    app.run(host='0.0.0.0',port=8000,debug=True)
```

7.3 Códigos para realizar la detección el tipo de objeto o color en la Jetson AGX Xavier

- **Decoder color** (para ordenar a la Jetson de realizar la detección de color)

```
#identificacion del color (se carga la imagen y se indentifica
el color programa detect_color1.py)
import os
import json
import base64
import subprocess
import time
from flask import Flask, jsonify, abort, request
import requests
import threading
import signal

import cv2
import numpy as np

app = Flask(__name__)
port = 8000
deployhost = '0.0.0.0'


@app.route('/decoder',methods=['POST'])

def decoder():
  r = requests.post("http://192.168.50.88:8000/foto", data='$$')

  json_data = json.loads(r.text)

  #print(json_data)


  #print data
  imgdata = base64.b64decode(json_data['img'])
  filename = 'some_image.jpg'  # I assume you have a way of
picking unique filenames
  with open(filename, 'wb') as f:
    f.write(imgdata)
```

```
  #proc = subprocess.Popen("python3
/home/nvidia/Documents/experiments/cards_002/yolov3_gpu/darknet/
detect.py", shell=True, preexec_fn=os.setsid)
  #proc.wait()


  process = subprocess.Popen(["python",
"/home/nvidia/Documents/experiments/cards_002/yolov3_gpu/darknet
/detect_color1.py"], stdout=subprocess.PIPE)
  process.wait()
  aux= process.communicate()[0]
  print(aux)
  longi=(len(aux))
  r2=0
  x = aux[2]+aux[3]+aux[4]+aux[5]+aux[6]
  y = aux[11]+aux[12]+aux[13]+aux[14]+aux[15]
  w = aux[19]+aux[20]+aux[21]
  h = aux[24]+aux[25]+aux[26]
  r1= aux[30]+aux[31]+aux[32]+aux[33]
  r1=float(r1)
  long_name = int(aux[37])
  i=1
  j=42
  es= aux[41]
  while(i<(long_name)):
    es = es + aux[j]
    i=i+1
    j=j+1

  j=j+5
  if(longi>60):
    long_name2 = int(aux[j+35])
    r2= aux[j+28]+aux[j+29]+aux[j+30]+aux[j+31]
    r2=float(r2)
    print(r1,r2)


  if(r2>r1):  #cojemos el color que mas espacio abarca
    x = aux[j]+aux[j+1]+aux[j+2]+aux[j+3]+aux[j+4]
    y = aux[j+9]+aux[j+10]+aux[j+11]+aux[j+12]+aux[j+13]
    print(x,y)
    i=1
    k=j+40
    es= aux[j+39]
    while(i<(long_name2)):
      es = es + aux[k]
      i=i+1
      k=k+1

  """print(longi)
  print(long_name)
  print(long_name2)
  print(r1,r2)
  print(x)
  print(y)
```

```
  print(w)
  print(h)
  print(es)
  x= float(x)
  y= float(y)
  w= float(w)
  h= float(h)
  es= str(es)"""
  print(x,y,es)
  data =(x,y,es,w,h)

  data_string = json.dumps(data)
  return data_string

if __name__== '__main__':
   app.run(host='0.0.0.0',port=port,debug=True)
```

- **Detect color1** (realiza la detección del color utilizando OpenCv)

```
import numpy as np
import cv2

# define the lower and upper boundaries of the colors in the HSV
color space
lower = {'red':(0,80,80),
         'green':(49,50,50),
         'blue':(100,65,75),
         'yellow':(23,59,119),
         'black':(0,0,0),
         'brown' :(10, 100, 20)
         }

upper = {'red':(20,255,255),
         'green':(80, 255, 255),
         'blue':(130, 255, 255),
         'yellow':(43,255,255),
         'black':(40,40,40),
         'brown' :(20,255,200)
         }

# define standard colors for circle around the object
colors = {'red':(0,0,255),
          'green':(45, 87, 44),
          'blue':(255,0,0),
          'yellow':(0,255,217),
          'black':(0,0,0),
          'brown' :(42,59,102)
          }

width = 720/2
height = 480/2
```

```
font = cv2.FONT_HERSHEY_SIMPLEX



    # grab the current frame
frame = cv2.imread('some_image.jpg')
cv2.circle(frame,(int(round(width)),int(round(height))),20,(255,
255,255),1)

blurred = cv2.GaussianBlur(frame,(11,11),0)
hsv = cv2.cvtColor(blurred,cv2.COLOR_BGR2HSV)
    #for each color in dictionary check object in frame
for key, value in upper.items():
  kernel = np.ones((9,9),np.uint8)
  mask = cv2.inRange(hsv,lower[key],upper[key])
  mask = cv2.morphologyEx(mask,cv2.MORPH_OPEN,kernel)
  mask = cv2.morphologyEx(mask,cv2.MORPH_CLOSE,kernel)

        #Calculate percentage of pixel colors
  output = cv2.countNonZero(mask)
  res = np.divide(float(output),mask.shape[0]*int(mask.shape[1]
/ 128))
  percent_colors = np.multiply((res),400) / 10000
  percent=(np.round(percent_colors*100,2))

  cnts =
cv2.findContours(mask.copy(),cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMP
LE)[-2]
  center = None

  if len(cnts) > 0:
    c = max(cnts, key=cv2.contourArea)
    ((x,y), radius) = cv2.minEnclosingCircle(c)
    M = cv2.moments(c)
    center = (int(M["m10"] / M["m00"]),int(M["m01"] / M["m00"]))

    if radius > 0.5:

cv2.circle(frame,(int(x),int(y)),int(radius),colors[key],2)

cv2.circle(frame,(int(round(x)),int(round(y))),5,colors[key],2)

      if(radius>=100):
        radius=99
      r=round(radius,1)
      r=str(r)
      r=r.zfill(4)
      x = float(x)
      x = round(x,1)
      x=str(x)
      x= x.zfill(5)
      y = float(y)
      y = round(y,1)
```

```
      y=str(y)
      y=y.zfill(5)
      longi=len(key)
      print(x,y,width,height,r,longi,key)
      #cv2.putText(frame,str(percent) + '% ' +key,(int(x-
radius),int(y-radius)),font,0.6,colors[key],2)

  cv2.imwrite('Colors.jpg', frame)
  cv2.imshow("Frame",frame)
  key = cv2.waitKey(1)
  if key == ord("q"):
    break

#camera.release()
cv2.destroyAllWindows()
```

- **Decoder yolo** (para ordenar a la Jetson de realizar la detección del tipo de objeto)

```
#identificacion de la imagen enel Jetson en el progama detect.py
import os
import json
import base64
import subprocess
import time
from flask import Flask, jsonify, abort, request
import requests
import threading
import signal

import cv2
import numpy as np

app = Flask(__name__)
port = 8000
deployhost = '0.0.0.0'


@app.route('/decoder',methods=['POST'])

def decoder():
  r = requests.post("http://192.168.50.88:8000/foto", data='$$')

  json_data = json.loads(r.text)

  #print(json_data)


  #print data
  imgdata = base64.b64decode(json_data['img'])
  filename = 'some_image.jpg'  # I assume you have a way of
picking unique filenames
```

```
  with open(filename, 'wb') as f:
    f.write(imgdata)

  #proc = subprocess.Popen("python3
/home/nvidia/Documents/experiments/cards_002/yolov3_gpu/darknet/
detect.py", shell=True, preexec_fn=os.setsid)
  #proc.wait()


  process = subprocess.Popen(["python3",
"/home/nvidia/Documents/experiments/cards_002/yolov3_gpu/darknet
/detect.py"], stdout=subprocess.PIPE)
  process.wait()
  aux= process.communicate()[0]
  print (aux)
  #print (len (aux))
  x = aux[0]+aux[1]+aux[2]+aux[3]+aux[4]+aux[5]+aux[6]+aux[7]
  y =
aux[9]+aux[10]+aux[11]+aux[12]+aux[13]+aux[14]+aux[15]+aux[16]
  long_name = int(aux[18])
  i=1
  j=23
  es= aux[22]
  while(i<(long_name)):
    es = es + aux[j]
    i=i+1
    j=j+1



  print(x)
  print(y)
  print(es)
  data = (x,y,es)
  data_string = json.dumps(data)
  return data_string

if __name__== '__main__':
   app.run(host='0.0.0.0',port=port,debug=True)
```

- **Detect** (realiza la detección del tipo de objeto utilizando yolo)

```
net = Detector(bytes("cfg/yolov3.cfg", encoding="utf-8"),
bytes("yolov3.weights", encoding="utf-8"), 0,
bytes("cfg/coco.data",encoding="utf-8"))

img =
cv2.imread(os.path.join(darknet_home,"/home/nvidia/Documents/exp
eriments/cards_002/yolov3_gpu/darknet/region.jpg"))
img2 = Image(img)
# r = net.classify(img2)
```

```
results = net.detect(img2)
#print(results)
#print(type(results))
longi=len(results)
#print(longi)
name__object= results[0][0]
#print(results[0][2][0])
#print (type(results[0][2][0]))
x = results[0][2][0]
y = results[0][2][1]
for i in range(longi-2):
   if (results[i][1]< results[i+1][1]):
         name__object= results[i+1][0]
         x = results[i+1][2][0]
         y = results[i+1][2][1]
#print(name__object)
long_name = len (name__object)
x = float(x)
x = round(x,4)
x=str(x)
x= x.zfill(8)
y = float(y)
y = round(y,4)
y=str(y)
y=y.zfill(8)
print (x,y,long_name,name__object)
```

7.4 Códigos para controlar los movimientos del robot

- **Position Photo** (para mover el robot a una posición concreta, por lo tanto, la cámara y realizar la foto)

```
import os
import json
import base64
import subprocess
import time
from flask import Flask, jsonify, abort, request
import requests
import threading
import signal



app = Flask(__name__)
port = 8003
deployhost = '0.0.0.0'


home_color = {
    'x': 180.7058,
    'y': -49.1664,
    'z': 140.7473,
```

```
    'r': 0,
    'a': 0
}

home1 = {
    'x': 170.2699,
    'y': -40.4981,
    'z': 100,
    'r': -1.0588,
    'a': 0
}


@app.route('/position-photo',methods=['GET'])
def position():
  data=home1
  r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)

  return '0'


if __name__== '__main__':
  app.run(host='0.0.0.0',port=port,debug=True)
```

- **Position dice OpenCv** (para llevar el dado a su posición de salida, según el color, haciendo un control en bucle abierto)

```
#calculamos y enviamos al robot las posiciones en las que va ir
moviendo y le indicamos con la variable a si se agarra o no el
dado
#es un control en blucle abierto
import os
import json
import base64
import subprocess
import time
from flask import Flask, jsonify, abort, request
import requests
import threading
import signal
import math



app = Flask(__name__)
port = 8002
deployhost = '0.0.0.0'

home = {
    'x': 300,
    'y': -3.4981,
```

```
    'z': 100,
    'r': 0,
    'a': 0
}
home1 = {
    'x': 300,
    'y': -3.4981,
    'z': 100,
    'r': 0,
    'a': 1
}

homeaux0 = {
    'x': 250,
    'y': -200,
    'z': 100,
    'r': 22.8971,
    'a': 0
}

homeaux1 = {
    'x': 250,
    'y': 0,
    'z': 100,
    'r': 0,
    'a': 1
}



dice = {
    'x': 276.5108,
    'y': 116.7860,
    'z': -46.4538,
    'r': 0,
    'a': 1
}


inter0 = {
    'x': 108.6552,
    'y': -241.8352,
    'z': 100,
    'r': 0,
    'a': 0
}

inter1 = {
    'x': 108.6552,
    'y': -241.8352,
    'z': 100,
    'r': 0,
    'a': 1
}
posaux  = {
```

```
    'x': 9.4321,
    'y': -307.1571,
    'z': 100,
    'r': 0,
    'a': 0
}

red = {
    'x': -67.8236,
    'y': -305.7152,
    'z': -45.4538,
    'r': 0,
    'a': 0
}
black = {
    'x': -67.8236,
    'y': -226.0775,
    'z': -45.4538,
    'r': 0,
    'a': 0
}

blue = {
    'x': 9.4321,
    'y': -305.7152,
    'z': -45.4538,
    'r': 0,
    'a': 0
}

brown = {
    'x': 9.4321,
    'y': -226.0775,
    'z': -45.4538,
    'r': 0,
    'a': 0
}

yellow = {
    'x': 75.2628,
    'y': -295.7152,
    'z': -45.4538,
    'r': 0,
    'a': 0
}
green = {
    'x': 75.2628,
    'y': -215.0775,
    'z': -45.4538,
    'r': 0,
    'a': 0
}
```

```
#posision (0,0) de imagen
ix=315.4689
iy=80
image = 1

@app.route('/coger-dado',methods=['GET'])
def coger():
  global image
  image=1
  r=requests.post('http://192.168.50.52:8000/decoder',data='$$')
  json_data = r.json()

  x = float(json_data[0])
  y = float(json_data[1])
  es= json_data[2]
  print(es)


  auxx=0
  auxy=0
  if (x<360.0 and y<240.0):
    auxy=0
    auxx=4

  if (x<360 and y>240):
    auxy=-8
    auxx=3
  if (x>360 and y>240):
    auxx=-2
    auxy=-5
  if (x>360 and y<240):
    auxy=-8
    auxx=4
  if(x>=330 and x<=390 and y>=230 and y<=250):
    auxy=0
    auxx=0



  """if(es == 'person'):
    image=6
  if(es == 'car'):
    image=5
  if(es == 'truck'):
    image=4
  if(es == 'bicycle'):
    image=3
  if(es == 'dog'):
    image=2
  if(es == 'horse'):
    image=1"""

  if(es == 'yellow'):
    image=1
  if(es == 'green'):
```

```
    image=2
  if(es == 'blue'):
    image=3
  if(es == 'brown'):
    image=4
  if(es == 'red'):
    image=5
  if(es == 'black'):
    image=6

  escala = 0.31392774831
  c= 0.026125767907
  s=-1.028260259116



  print (x)
  print (y)
  print (es)

  dice['x'] =     ix+auxx-((c*x - s*y))*escala
  dice['y'] =     iy+auxy+((c*y + s*x))*escala
  homeaux0['x'] = ix+auxx-((c*x - s*y))*escala
  homeaux0['y'] = iy+auxy+((c*y + s*x))*escala
  homeaux1['x'] = ix+auxx-((c*x - s*y))*escala
  homeaux1['y'] = iy+auxy+((c*y + s*x))*escala
  homeaux0['z'] = 100
  homeaux1['z'] = 100


  print (dice)
  print (homeaux0)
  print (homeaux1)
  if((image >0) and (image <7) ):
    data=homeaux0
    r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)
    data = dice
    r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)
    data = homeaux1
    r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)
    data = home1
    r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)
  return '0'

@app.route('/position-dado',methods=['GET'])
def imagen():
  data=inter1
```

```
  r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)

  if(image == 1):
    data = yellow
    aux= 'yellow'
  elif(image == 2):
    data = green
    aux= 'green'
  elif(image == 3):
    data = blue
    aux= 'blue'
  elif(image == 4):
    data = brown
    aux = 'brown'
  elif(image == 5):
    data = red
    aux= 'red'
  elif(image == 6):
    data = black
    aux= 'black'

  r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)
  data=posaux
  r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)
  data=inter0
  r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)
  data = home
  r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)
 #
  return jsonify(aux)

if __name__== '__main__':
   app.run(host='0.0.0.0',port
```

- **Position dice yolo** (para llevar el dado a su posición de salida, según el tipo, haciendo un control en bucle abierto)

```
import os
import json
import base64
import subprocess
import time
from flask import Flask, jsonify, abort, request
import requests
```

```
import threading
import signal



app = Flask(__name__)
port = 8001
deployhost = '0.0.0.0'

home = {
    'x': 300,
    'y': -3.4981,
    'z': 100,
    'r': 0,
    'a': 0
}
home1 = {
    'x': 300,
    'y': -3.4981,
    'z': 100,
    'r': 0,
    'a': 1
}

homeaux0 = {
    'x': 250,
    'y': -200,
    'z': 100,
    'r': 22.8971,
    'a': 0
}

homeaux1 = {
    'x': 250,
    'y': 0,
    'z': 100,
    'r': 0,
    'a': 1
}



dice = {
    'x': 276.5108,
    'y': 116.7860,
    'z': -46.4538,
    'r': 0,
    'a': 1
}


inter0 = {
    'x': 108.6552,
    'y': -241.8352,
    'z': 100,
```

```
    'r': 0,
    'a': 0
}

inter1 = {
    'x': 108.6552,
    'y': -241.8352,
    'z': 100,
    'r': 0,
    'a': 1
}
posaux  = {
    'x': 9.4321,
    'y': -307.1571,
    'z': 100,
    'r': 0,
    'a': 0
}



horse = {
    'x': 9.4321,
    'y': -305.7152,
    'z': -45.4538,
    'r': 0,
    'a': 0
}

car = {
    'x': 9.4321,
    'y': -226.0775,
    'z': -45.4538,
    'r': 0,
    'a': 0
}

bicycle = {
    'x': 75.2628,
    'y': -295.7152,
    'z': -45.4538,
    'r': 0,
    'a': 0
}
person = {
    'x': 75.2628,
    'y': -215.0775,
    'z': -45.4538,
    'r': 0,
    'a': 0
}


image = 4
ix=315.4689
```

```
iy=80

@app.route('/coger-carta',methods=['GET'])
def coger():
  global image
  image=1
  r=requests.post('http://192.168.50.52:8000/decoder',data='$$')
  json_data = r.json()

  x = float(json_data[0])
  y = float(json_data[1])
  es= json_data[2]
  print(es)


  auxx=0
  auxy=0
  if (x<360.0 and y<240.0):
    auxy=0
    auxx=4

  if (x<360 and y>240):
    auxy=-8
    auxx=3
  if (x>360 and y>240):
    auxx=-2
    auxy=-5
  if (x>360 and y<240):
    auxy=-8
    auxx=4
  if(x>=330 and x<=390 and y>=230 and y<=250):
    auxy=0
    auxx=0



  if(es == 'person'):
    image=1
  if(es == 'car'):
    image=2

  if(es == 'bicycle'):
    image=3

  if(es == 'horse'):
    image=4



  escala = 0.31392774831
  c= 0.026125767907
  s=-1.028260259116
```

```
  print (x)
  print (y)
  print (es)

  dice['x'] =     ix+auxx-((c*x - s*y))*escala
  dice['y'] =     iy+auxy+((c*y + s*x))*escala
  homeaux0['x'] = ix+auxx-((c*x - s*y))*escala
  homeaux0['y'] = iy+auxy+((c*y + s*x))*escala
  homeaux1['x'] = ix+auxx-((c*x - s*y))*escala
  homeaux1['y'] = iy+auxy+((c*y + s*x))*escala
  homeaux0['z'] = 100
  homeaux1['z'] = 100


  print (dice)
  print (homeaux0)
  print (homeaux1)
  if((image >0) and (image <7) ):
    data=homeaux0
    r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)
    data = dice
    r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)
    data = homeaux1
    r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)
    data = home1
    r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)
  return '0'


@app.route('/position-carta',methods=['GET'])
def imagen():
  data=inter1
  r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)

  if(image == 1):
    data = person
    aux= 'person'
  elif(image == 2):
    data = car
    aux= 'car'
  elif(image == 3):
    data = bicycle
    aux= 'bicycle'
  elif(image == 4):
    data = horse
```

```
    aux = 'horse'


  r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)
  data=posaux
  r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)
  data=inter0
  r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)
  data = home
  r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)
 #
  return jsonify(aux)


if __name__== '__main__':
   app.run(host='0.0.0.0',port=port,debug=True)
```

- **Position dice OpenCv distance** (para llevar el dado a su posición de salida, según el tipo, haciendo un control en bucle cerrado)

```
#calculamos y enviamos al robot las posiciones en las que va ir
moviendo y le indicamos con la variable a si se agarra o no el
dado

import os
import json
import base64
import subprocess
import time
from flask import Flask, jsonify, abort, request
import requests
import threading
import signal
import math



app = Flask(__name__)
port = 8002
deployhost = '0.0.0.0'

home = {
    'x': 300,
    'y': -3.4981,
    'z': 100,
```

```
    'r': 0,
    'a': 0
}
home1 = {
    'x': 300,
    'y': -3.4981,
    'z': 100,
    'r': 0,
    'a': 1
}

homeaux0 = {
    'x': 250,
    'y': -200,
    'z': 100,
    'r': 22.8971,
    'a': 0
}

homeaux1 = {
    'x': 250,
    'y': 0,
    'z': 100,
    'r': 0,
    'a': 1
}



dice = {
    'x': 276.5108,
    'y': 116.7860,
    'z': -46.4538,
    'r': 0,
    'a': 1
}


inter0 = {
    'x': 108.6552,
    'y': -241.8352,
    'z': 100,
    'r': 0,
    'a': 0
}

inter1 = {
    'x': 108.6552,
    'y': -241.8352,
    'z': 100,
    'r': 0,
    'a': 1
}
posaux  = {
    'x': 9.4321,
```

```
    'y': -307.1571,
    'z': 100,
    'r': 0,
    'a': 0
}

red = {
    'x': -67.8236,
    'y': -305.7152,
    'z': -45.4538,
    'r': 0,
    'a': 0
}
black = {
    'x': -67.8236,
    'y': -226.0775,
    'z': -45.4538,
    'r': 0,
    'a': 0
}

blue = {
    'x': 9.4321,
    'y': -305.7152,
    'z': -45.4538,
    'r': 0,
    'a': 0
}

brown = {
    'x': 9.4321,
    'y': -226.0775,
    'z': -45.4538,
    'r': 0,
    'a': 0
}

yellow = {
    'x': 75.2628,
    'y': -295.7152,
    'z': -45.4538,
    'r': 0,
    'a': 0
}
green = {
    'x': 75.2628,
    'y': -215.0775,
    'z': -45.4538,
    'r': 0,
    'a': 0
}



image = 1
```

```
home_color = {
    'x': 220.9367,
    'y': -58.4711,
    'z': 140.7473,
    'r': 0,
    'a': 0
    }
#posision (0,0) de imagen
posx=180.7058
posy=-49


@app.route('/coger-dado',methods=['GET'])
def coger():
  global posx
  global posy
  global image
  image=1


  i=1
  j=1

  while (i!=0 or j!=0):

    escalax = 0.3430769
    escalay = 0.4385964

r=requests.post('http://192.168.50.52:8000/decoder',data='$$')
    json_data = r.json()

    y = float (json_data[0])
    x = float (json_data[1])
    es= (json_data[2])
    h= float (json_data[3])
    w = float (json_data[4])



    distx=w-x
    disty=h-y

    posx=posx+distx*escalax*i
    posy=posy+disty*escalay*j
    auxx=posx
    auxy=posy
    if(distx<5 and distx>-5):
      i=0

    if(disty<5 and disty>-5):
      j=0


    print(x,y,w,h)
```

```
    print(distx,disty)
    print(posx,posy)
    print(i,j)
    data = {
      'x': posx,
      'y': posy,
      'z': 140.7473,
      'r': 0,
      'a': 0
    }

    print(data)
    r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)


  print (es)
  if(es == 'yellow'):
    image=1
  if(es == 'green'):
    image=2
  if(es == 'blue'):
    image=3
  if(es == 'brown'):
    image=4
  if(es == 'red'):
    image=5
  if(es == 'black'):
    image=6
    #time.sleep(10)

  if (i==0 and j==0):
    print("sssssssssssssssssssssssssssssssssssss")
    posx = auxx+50
    posy = auxy
    data['x'] = posx
    data['y'] = posy
    dice['x'] =    posx
    dice['y'] =    posy
    homeaux0['x'] = posx
    homeaux0['y'] = posy
    homeaux1['x'] = posx
    homeaux1['y'] = posy
    homeaux0['z'] = 100
    homeaux1['z'] = 100
  print(data)
  r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)

  if((image >0) and (image <7) ):
    data=homeaux0
```

```
    r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)
    data = dice
    r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)
    data = homeaux1
    r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)
    data = home1
    r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)
  return '0'

@app.route('/position-dado',methods=['GET'])
def imagen():
  data=inter1
  r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)

  if(image == 1):
    data = yellow
    aux= 'yellow'
  elif(image == 2):
    data = green
    aux= 'green'
  elif(image == 3):
    data = blue
    aux= 'blue'
  elif(image == 4):
    data = brown
    aux = 'brown'
  elif(image == 5):
    data = red
    aux= 'red'
  elif(image == 6):
    data = black
    aux= 'black'

  r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)
  data=posaux
  r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)
  data=inter0
  r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)
  data = home
```

```
    r =
requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords",
json=data)

    return jsonify(aux)

if __name__== '__main__':
    app.run(host='0.0.0.0',port=port,debug=True)
```

- **Position dice yolo distance** (para llevar el dado a su posición de salida, según el tipo, haciendo un control en bucle cerrado)

```
#calculamos y enviamos al robot las posiciones en las que va ir moviendo y le indicamos con la variable a si se agarra o no el dado

import os
import json
import base64
import subprocess
import time
from flask import Flask, jsonify, abort, request
import requests
import threading
import signal
import math



app = Flask(__name__)
port = 8002
deploythost = '0.0.0.0'

home = {
    'x': 300,
    'y': -3.4981,
    'z': 100,
    'r': 0,
    'a': 0
}
home1 = {
    'x': 300,
    'y': -3.4981,
    'z': 100,
    'r': 0,
    'a': 1
}

homeaux0 = {
    'x': 250,
    'y': -200,
```

```
    'z': 100,
    'r': 22.8971,
    'a': 0
}

homeaux1 = {
    'x': 250,
    'y': 0,
    'z': 100,
    'r': 0,
    'a': 1
}



dice = {
    'x': 276.5108,
    'y': 116.7860,
    'z': -46.4538,
    'r': 0,
    'a': 1
}


inter0 = {
    'x': 108.6552,
    'y': -241.8352,
    'z': 100,
    'r': 0,
    'a': 0
}

inter1 = {
    'x': 108.6552,
    'y': -241.8352,
    'z': 100,
    'r': 0,
    'a': 1
}
posaux  = {
    'x': 9.4321,
    'y': -307.1571,
    'z': 100,
    'r': 0,
    'a': 0
}

horse = {
    'x': 9.4321,
```

```
    'y': -305.7152,
    'z': -45.4538,
    'r': 0,
    'a': 0
}

car = {
    'x': 9.4321,
    'y': -226.0775,
    'z': -45.4538,
    'r': 0,
    'a': 0
}

bicycle = {
    'x': 75.2628,
    'y': -295.7152,
    'z': -45.4538,
    'r': 0,
    'a': 0
}
person = {
    'x': 75.2628,
    'y': -215.0775,
    'z': -45.4538,
    'r': 0,
    'a': 0
}



image = 1

photo = {
    'x': 220.9367,
    'y': -58.4711,
    'z': 140.7473,
    'r': 0,
    'a': 0
    }
#posision (0,0) de imagen
posx=180.7058
posy=-49


@app.route('/coger-dado',methods=['GET'])
def coger():
  global posx
  global posy
```

```
global image
image=1


i=1
j=1

while (i!=0 or j!=0):

 escalax = 0.3430769
 escalay = 0.4385964
 r=requests.post('http://192.168.50.52:8000/decoder',data='$$')
 json_data = r.json()

 y = float (json_data[0])
 x = float (json_data[1])
 es= (json_data[2])
 h= float (json_data[3])
 w = float (json_data[4])



 distx=w-x
 disty=h-y

 posx=posx+distx*escalax*i
 posy=posy+disty*escalay*j
 auxx=posx
 auxy=posy
 if(distx<5 and distx>-5):
  i=0

 if(disty<5 and disty>-5):
  j=0


 print(x,y,w,h)
 print(distx,disty)
 print(posx,posy)
 print(i,j)
 data = {
  'x': posx,
  'y': posy,
  'z': 140.7473,
  'r': 0,
  'a': 0
 }

 print(data)
```

```
  r = requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords", json=data)


 print (es)
 if(es == 'person'):
  image=1
 if(es == 'car'):
  image=2

 if(es == 'bicycle'):
  image=3

 if(es == 'horse'):
  image=4

 if (i==0 and j==0):

  posx = auxx+50
  posy = auxy
  data['x'] = posx
  data['y'] = posy
  dice['x'] =    posx
  dice['y'] =    posy
  homeaux0['x'] = posx
  homeaux0['y'] = posy
  homeaux1['x'] = posx
  homeaux1['y'] = posy
  homeaux0['z'] = 100
  homeaux1['z'] = 100
 print(data)
 r = requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords", json=data)

 if((image >0) and (image <7) ):
  data=homeaux0
  r = requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords", json=data)
  data = dice
  r = requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords", json=data)
  data = homeaux1
  r = requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords", json=data)
  data = home1
  r = requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords", json=data)
 return '0'

@app.route('/position-dado',methods=['GET'])
def imagen():
 data=inter1
 r = requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords", json=data)

 if(image == 1):
```

```
    data = person
    aux= 'person'
  elif(image == 2):
    data = car
    aux= 'car'
  elif(image == 3):
    data = bicycle
    aux= 'bicycle'
  elif(image == 4):
    data = horse
    aux = 'horse'

  r = requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords", json=data)
  data=posaux
  r = requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords", json=data)
  data=inter0
  r = requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords", json=data)
  data = home
  r = requests.put("http://192.168.50.90:8000/dobot/api/v1.0/coords", json=data)

  return jsonify(aux)

if __name__== '__main__':
        app.run(host='0.0.0.0',port=port,debug=True)
```

7.5 Aplicación planta.json en Node-RED

se puede observar la aplicación hecho importando en Node-RED el siguiente código:


```
[{"id":"eee63f67.8e106","type":"tab","label":"Flow
4","disabled":false,"info":""},{"id":"4ba48ac8.9df0b4","type":"inject","z":"eee63f67.8e106","n
ame":"¿image-
dado?","topic":"","payload":"","payloadType":"date","repeat":"","crontab":"","once":false,"on
ceDelay":0.1,"x":110,"y":140,"wires":[["19048379.73c44d"]]},{"id":"19048379.73c44d","type":
"http request","z":"eee63f67.8e106","name":"move
conveyor","method":"POST","ret":"txt","paytoqs":false,"url":"http://192.168.50.87:8000/cont
erclockwise","tls":"","persist":false,"proxy":"","authType":"","x":120,"y":220,"wires":[["95967b
b9.38d9d8"]]},{"id":"50f5e405.5a93ec","type":"debug","z":"eee63f67.8e106","name":"it is
","active":true,"tosidebar":true,"console":false,"tostatus":false,"complete":"payload","targetT
ype":"msg","x":650,"y":160,"wires":[]},{"id":"95967bb9.38d9d8","type":"delay","z":"eee63f67.
8e106","name":"","pauseType":"delay","timeout":"5","timeoutUnits":"seconds","rate":"1","n
bRateUnits":"1","rateUnits":"second","randomFirst":"1","randomLast":"5","randomUnits":"se
conds","drop":false,"x":120,"y":320,"wires":[["2dedcca8.9d3ff4","5151c71e.672028"]]},{"id":"
97bb04a6.1a3808","type":"http request","z":"eee63f67.8e106","name":"picture and take the
dice","method":"GET","ret":"txt","paytoqs":false,"url":"http://localhost:8002/coger-
dado","tls":"","persist":false,"proxy":"","authType":"","x":410,"y":380,"wires":[["48369494.ef9
edc"]]},{"id":"48369494.ef9edc","type":"http request","z":"eee63f67.8e106","name":"
position","method":"GET","ret":"txt","paytoqs":false,"url":"http://localhost:8002/position-
```

dado","tls":"","persist":false,"proxy":"","authType":"","x":580,"y":300,"wires":[["50f5e405.5a93ec"]]},{"id":"5151c71e.672028","type":"http request","z":"eee63f67.8e106","name":"stop cconveyor","method":"POST","ret":"txt","paytoqs":false,"url":"http://192.168.50.87:8000/stop","tls":"","persist":false,"proxy":"","authType":"","x":360,"y":260,"wires":[[]]},{"id":"10647e12.f5ddf2","type":"inject","z":"eee63f67.8e106","name":"conect robot","topic":"","payload":"","payloadType":"date","repeat":"","crontab":"","once":false,"onceDelay":0.1,"x":90,"y":60,"wires":[["a783abcd.c4e108"]]},{"id":"a783abcd.c4e108","type":"http request","z":"eee63f67.8e106","name":"conect","method":"POST","ret":"txt","paytoqs":false,"url":"http://localhost:8000/dobot/api/v1.0/connect","tls":"","persist":false,"proxy":"","authType":"","x":350,"y":60,"wires":[["1014a950.d639e7"]]},{"id":"1014a950.d639e7","type":"debug","z":"eee63f67.8e106","name":"¿robot ready to use? ","active":true,"tosidebar":true,"console":false,"tostatus":false,"complete":"payload","targetType":"msg","x":640,"y":60,"wires":[]},{"id":"2dedcca8.9d3ff4","type":"http request","z":"eee63f67.8e106","name":" position photo","method":"GET","ret":"txt","paytoqs":false,"url":"http://localhost:8003/position-photo","tls":"","persist":false,"proxy":"","authType":"","x":180,"y":380,"wires":[["97bb04a6.1a3808"]]}]

VIII. Referencias

[1]http://members.tripod.com/iua_informatica.ar/informatica_industrial/download/historia.pdf

[2] https://mrhouston.net/blog/pros-y-contras-automatizacion-de-procesos/

[3] https://devblogs.nvidia.com/nvidia-Jetson-agx-xavier-32-teraops-ai-robotics/?ncid=so-fac-mdjngxxrmllhml-69163

[4] https://www.xataka.com/makers/cero-maker-todo-necesario-para-empezar-raspberry-pi

[5] https://www.idento.es/blog/desarrollo-web/que-es-una-api-rest/

[6] https://biblioteca.unirioja.es/tfe_e/TFE004595.pdf

[7] https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/

[8]http://vision.stanford.edu/teaching/cs231b_spring1415/slides/ssearch_schuyler.pdf

[9] https://medium.com/@pratheesh.27998/object-detection-part1-4dbe5147ad0a

[10] https://www.youtube.com/watch?v=4eIBisqx9_g

[11] https://pjreddie.com/darknet/yolo/

[12]https://OpenCv.org/

[13] https://es.wikipedia.org/wiki/Ubuntu

[14] https://es.wikipedia.org/wiki/TensorFlow

[15] https://github.com/suvrat29/Card_detection