Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Aplicaciones de “Deep Learning” en entorno ROS

Autor: Andrea Gómez Tejada
Tutor: Joaquín Ferruz Melero

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo de Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Aplicaciones de “Deep Learning” en entorno ROS

Autor:
Andrea Gómez Tejada

Tutor:
Joaquín Ferruz Melero
Catedrático

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Trabajo de Fin de Grado: Aplicaciones de “Deep Learning” en entorno ROS

Autor: Andrea Gómez Tejada

Tutor: Joaquín Ferruz Melero

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A todos los que me acompañaron durante este camino.

Índice

ÍNDICE DE FIGURAS

Figura 35: GarbageCollector Robot 34

1 INTRODUCCIÓN

Actualmente, la robótica está cada vez más presente en nuestro día a día, en industrias, en el hogar, en el trabajo, en personas y en servicios. Por esta razón es una tecnología que se desarrolla muy rápida para cubrir las necesidades que van surgiendo, esto contribuye a que la calidad de vida de los seres humanos mejore.

Debido al auge de este sector, los humanos tendemos a sistematizar y automatizar cada vez más tareas, tanto que en la actualidad se investigan técnicas como el "Deep Learning" o aprendizaje profundo para que los robots sean capaces de aprender una tarea nueva sin la necesidad de tener que ser reprogramados.

Esta técnica es una parte de lo que la mayoría de las personas conocen como Inteligencia Artificial. Intentamos dotar a la máquina capacidades parecidas a las del ser humano, siendo capaces éstas de interactuar con un humano sin necesidad de un intérprete. Se ha conseguido que un robot sea capaz de interpretar la voz y lenguaje no verbal e identificar objetos mediante la visión artificial.

Pueden ayudarnos en tareas como ayudar a una persona sin movilidad, analizar sistemas financieros, detectar en una obra si algún empleado no lleva casco o botas de seguridad y alarmar sobre ello e incluso coger un taxi sin necesidad de ningún chófer. Al igual que algunos trabajos desaparecerán, se crearán nuevos puestos para personas más cualificadas con el fin de desarrollar y mantener a esta tecnología.

El objetivo de este proyecto es desarrollar una aplicación en ROS (Robot Operating System) mediante una simulación con Gazebo que sea capaz de realizar alguna tarea mediante la técnica del "Deep Learning".

2 ESTADO DEL ARTE DEL "DEEP LEARNING"

2.1. Historia del "Deep Learning"

Hablar de la historia del "Deep Learning" no tiene mucho sentido ya que es una técnica relativamente nueva, pero la verdad es que su comienzo se remonta a mitad del siglo pasado desde la aparición de la Inteligencia Artificial, ya que ambos términos están estrechamente relacionados. El origen se debe a tres corrientes desde los años 50 del sigo XX hasta día de hoy [1]:

- Cibernética: desde los años 40 hasta los 60, comenzó con la curiosidad del aprendizaje biológico, en esta etapa se desarrolló el perceptrón, lo que actualmente conocemos como neurona. Los padres de la cibernética fueron McCulloch&Pitts y Hebb.
- Conexionismo: desde 1980 hasta 1995, surgió el concepto de propagación hacia atrás o retropropagación, que actualmente se utiliza con las redes neuronales.
- "Deep Learning": desde 2006 hasta la actualidad, no se pretende similar el mismo funcionamiento que el cerebro de un humano o de un animal, se pretende crear un tipo de aprendizaje. Con esta disciplina se pretende que la máquina sea capaz de aprender mediante unos conceptos que tienen una jerarquía.

Como se ha citado anteriormente, los pioneros fueron Pitts y Warren McCulloch [2], que se unieron para aportar conocimientos sobre la teoría de las redes neuronales, de autómatas y de la computación. El encargado de desarrollar el perceptrón fue Rosenblatt. Las redes neuronales, que ya introducen los pesos de la arquitectura, son capaces de variar dicho peso a través del entrenamiento. realizando una clasificación binaria. Los sistemas lineales no eran capaces de resolver todos los problemas, por lo que se congeló durante un tiempo el estudio de esta disciplina.

Tras la etapa oscura después de la cibernética [3], las redes neuronales recobran protagonismo gracias al conexionismo. Este movimiento pretendía implementar modelos cognitivos, ya que el cerebro humano es muy complejo de entender. Surgieron nuevos conceptos como la retropropagación o "backpropagation" y la representación distribuida. De nuevo la teoría de las redes neuronales decae debido a las promesas que realizaron los científicos en desarrollar ideas que en aquella época eran prácticamente imposibles.

Desde 2006 hasta la actualidad se sigue investigando sobre este tema, lo que actualmente se conoce ya como "Deep Learning". A priori se buscaba desarrollar un aprendizaje no supervisado, la industria está mas interesada en el aprendizaje supervisado.

2.2. Inteligencia Artifical

Esta disciplina se ha desarrollado por la necesidad de aplicar "inteligencia" a la hora de tratar con cierta información. Actualmente no existe una definición exacta de Inteligencia Artificial o IA, es un término que engloba distintas formas de implementar algoritmos, lo podremos definir como *esfuerzo para automatizar las tareas intelectuales que normalmente realizan los seres humanos* [4].

El inicio de la definición de Inteligencia Artificial comienza en 1950, con el Test de Turing. Turing, un matemático inglés publicó en el artículo *Computing machinery and intelligence* que si una máquina puede actuar como un ser humano porqué no se le puede llamar inteligente a la máquina. El Test de Turing consistía en que un humano se comunicaba con una terminal u otro humano en habitaciones contiguas, si la primera persona no es capaz de distinguir con quién se está comunicando, entonces podremos decir que la máquina es inteligente. Se llegó a la conclusión que para dotar de inteligencia a una máquina es necesario:

- Reconocimiento del lenguaje natural.
- Razonamiento.
- Aprendizaje.
- Representación del conocimiento.

Figura 1: Relación humano-máquina

Si una máquina es capaz de procesar el lenguaje natural, dicho lenguaje es muy complejo ya que entran en juego la dificultad de reconocer y construir frases mediante la morfología, la sintaxis y la semántica, entonces es posible la comunicación entre hombre y máquina. El razonamiento lógico se consigue a través de hechos introducidos en el sistema, actualmente se utilizan otros métodos como las redes probabilísticas. Con el aprendizaje automático se busca que la máquina sea capaz de cambiar sus parámetros cuando su entorno sufra algún cambio y ésta se adapte de nuevo al medio, se intenta simular un cerebro a través de las redes neuronales. Otro factor importante para definir una máquina como inteligente es la capacidad de representar el conocimiento una vez que ha tratado la información.

Existe otro test, el Test de Turing Total en el que se dota a la máquina la capacidad de visión artificial para la que es necesario el uso de una cámara y la capacidad de manipular, en el que intervendrá la robótica. Un terminal con estos elementos para ser inteligente debería ser capaz de localizar cierto objeto con la cámara y manipularlo con un brazo robótico por ejemplo teniendo en cuenta que dicho objeto no se puede dañar y tratarlo de una manera determinada; por tanto las máquinas serían capaces de interpretar su entorno.

2.3. “Machine Learning”

La razón por la que se crea este concepto es que actualmente producimos, almacenamos y enviamos millones de datos, y el tratamiento de los mismos es necesario [5]. La gran cantidad de datos ha hecho posible el auge del “Machine Learning” o conocido también como aprendizaje automático. Se puede definir como la técnica que hace posible que las máquinas sean capaces de aprender sin ser programadas de nuevo. Hay diferentes métodos de este tipo de aprendizaje, por refuerzo, algoritmos genéticos, por redes neuronales…

“Machine Learning” es la ciencia que permite que las computadoras aprendan y actúen como lo hacen los humanos, mejorando su aprendizaje a lo largo del tiempo de una forma autónoma, alimentándolas con datos e información en forma de observaciones e interacciones con el mundo real.” — Dan Fagella

Esta tecnología es capaz de implementar una respuesta a una serie de datos de entrada tras un procesamiento basado en modelos predictivos. La forma a través de la que interpretamos los datos es mediante matrices o vectores, cada fila de la matriz es un dato y cada columna es un factor o una característica [6]. Existen diferentes tipos de implementar el aprendizaje automático.

2.3.1 Aprendizaje supervisado

El modelo es entrenado con un conjunto de datos de entrada de los que la salida ya es conocida. Tras aprender los valores de salida, el modelo ajusta los parámetros internos para ser capaz de hacer predicciones con datos que no se han utilizado en el entrenamiento. El objetivo es realzar un mapeo con ciertas entradas y generar salidas. Existen dos tipos de aplicaciones:

- Clasificación: que puede ser binaria o multiclase, el objetivo es desarrollar una función de aproximación y encontrar relaciones entre los puntos con los que se ha entrenado y los nuevos para así clasificarlos en una parte u otra.

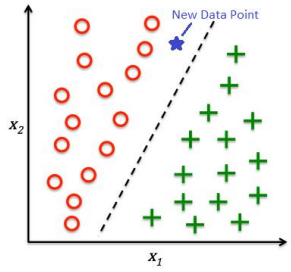

Figura 2: Gráfica aprendizaje por supervisión

- Regresión: Se encarga de aproximar una función continua según los datos. También se pretende buscar una función desconocida de aproximación en el que existirán variables predictoras y una variable de respuesta continua que será el resultado. El objetivo es crear la función, la variable de respuesta continua, mediante la relación entre el resto de variables.

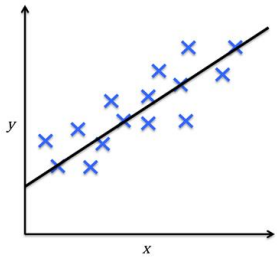

Figura 3: Gráfica aprendizaje por regresión

2.3.2 Aprendizaje no supervisado

La estructura de los datos es desconocida por tanto no hay etiquetas, entonces debemos buscar patrones iguales en la serie de datos de entrada. No se conoce la salida por tanto no es posible estimar el error. Existen dos categorías:

- Agrupamiento: es una técnica que analiza los datos, que clasifica en grupos según la información sin saber su estructura. La finalidad es obtener un número de grupos que clasifiquen por cierta característica.

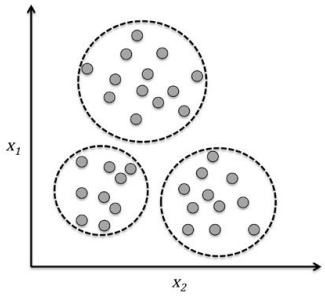

Figura 4: Gráfica apredizaje por agrupamiento

- Reducción dimensional: la posibilidad de trabajar con datos que tengan varias dimensiones es posible. Funciona encontrando similitudes y relaciones entre las características, por lo que alguna información puede ser redundante, ser una dependiente de otra linealmente por ejemplo. Por ello el objetivo es comprimir los datos más importantes en un subespacio más reducido para su tratamiento.

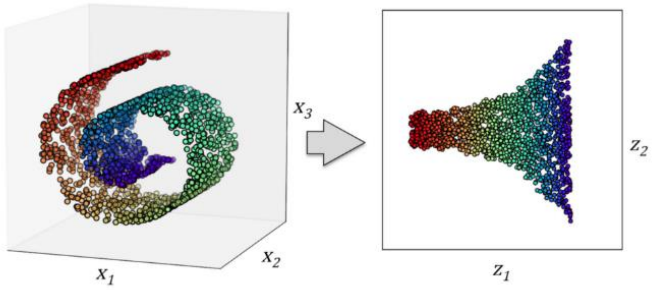

Figura 5: Gráfica aprendizaje por reducción dimensional

2.4. Aprendizaje profundo o “Deep Learning”

Debido a la enorme cantidad de datos con la que se trabaja hoy en día, se han tenido que crear nuevos términos como el Big Data. Este término ha sido el primer paso de todos los que quedan por avanzar en este tipo de tecnología. El “Deep Learning” [7] es el área de investigación más popular de la inteligencia artificial, han logrado procesar el lenguaje natural y la visión por computadora sin supervisión. Es un subcampo del “Machine Learning”; se define como un algoritmo automático estructurado que emula el aprendizaje humano con la finalidad de obtener ciertos conocimientos. Cabe destacar, que las redes neuronales son un modelo computacional basado en las conexiones entre sí de neuronas artificiales, por las que se transmite información y se produce un valor de salida, posteriormente se hará una explicación más profunda.

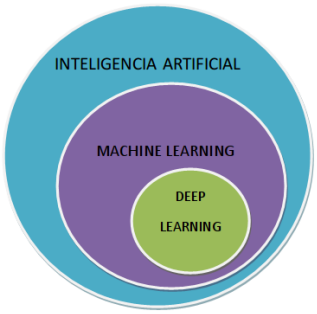

Figura 6: Diagrama Inteligencia Artificial

La finalidad del “Machine Learning” es etiquetar a cada entrada una salida mediante el ensayo de prueba-error de cada entrada y cada salida. Las entradas son mapeadas por las redes neuronales que están ordenadas de forma jerárquica y finalmente obtenemos la salida. El proceso avanza de tal forma que cada capa construye algo más complejo que la anterior. “Lo que cada capa hace” viene establecida por los pesos, por tanto los datos van cambiando. El inconveniente del “Deep Learning” es que una red neuronal puede estar constituidas por millones de parámetros y encontrar todas estas es muy complejo, ya que si modificamos alguno de ellos podemos cambiar los resultados de los demás [8].

Para poder definir algo, es necesario que se pueda observar, por lo que primeramente debemos ver cuánto de lejos se queda la salida de la red neuronal con la salida deseada, este trabajo lo realiza la función de pérdida. Gracias esta función podemos determinar si las predicciones son o no correctas. Con el resultado de esta función retroalimentamos la red para volver a ajustar los pesos, y de esto se encarga la retropropagación y el optimizador.

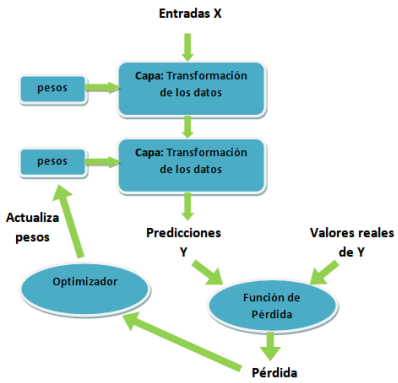

Figura 7: Diagrama entendimiento “Deep Learning”

Una de las razonas por las que se usa el “Deep Learning” es porque tiene la capacidad y la necesidad de operar con muchos datos, por lo que necesitan muchas muestras de datos para desarrollar una aplicación . Otro motivo es que gracias a la gran capacidad de cálculo con la que se trabaja, el tiempo de entrenamiento se ve reducido.

Hay ciertas diferencias a la hora de trabajar con el “Machine Learning” y el “Deep Learning”, entre ellas se encuentra que el ML los procesos elegidos se escogen de manera manual mientras que en el DL son automáticos. En el DL los datos no han sido procesados a diferencia del ML [9].

Por último es posible implementar el “Deep Learning” en algunos sectores como la medicina, es posible detectar células cancerígenas. La conducción autónoma también se está desarrollando, el coche es capaz de detectar un stop e incluso si una persona se interpone en el camino. En la industria también es posible ver si alguna persona está próxima a alguna zona peligrosa o a alguna máquina pesada.

3 REDES NEURONALES

Las redes neuronales es una herramienta que se utiliza actualmente para resolver problemas de aprendizaje supervisado. La semejanza con nuestro sistema nervioso biológico es que en cada una de las partes que lo constituyen, la neurona artificial es el elemento esencial en esta relación y está distribuida en capas. Este conjunto de capas formará una red neuronal y una red neuronal junto a las entradas y salidas dan lugar al sistema global completo [10].

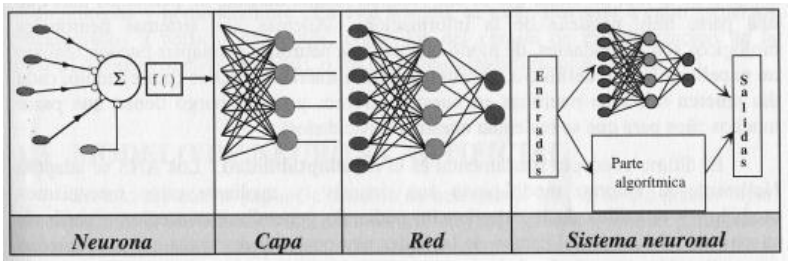

Figura 8: Sistema neuronal

Los conceptos que se intentan emular a la hora de diseñar una red neuronal son varios, entre ellos está el procesamiento en paralelo, la información va a ser tratada por varias neuronas a la misma vez. Existe una redundancia de la información almacenada en las redes ya que es posible la pérdida de ésta por tanto posee una memoria distribuida. Finalmente la adaptabilidad al entorno; las condiciones pueden variar y el modelo debe presentar flexibilidad a la hora de cambiar ciertos parámetro. Esto hace posible que el sistema sea capaz de aprender gracias a la experiencia.

Se puede definir red neuronal como un modelo computacional que consiste en un gran conjunto de neuronas simples, que actúan de forma similar a las neuronas biológicas.

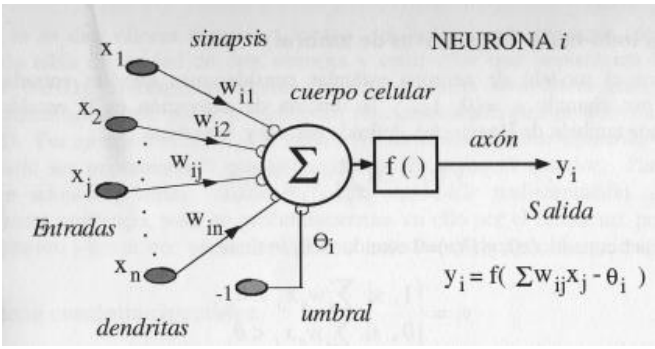

Figura 9: Neurona artificial

- x1, x1, …, xn: datos de entrada a las neuronas, pueden ser también resultados de la salida de otra neurona.
- x0: unidad de sesgo, valor constante que se suma a la entrada de la función de activación de la neurona, normalmente es 1 y dota de flexibilidad para que la neurona aprenda.
- wij: pesos sinápticos para cada entrada
- θi: umbral, se resta a la regla de propagación escogida, que normalmente es el producto de las entradas con los pesos sinápticos.

$$h_i(x_1,\ldots,x_n,w_{i1},\ldots,w_{in)=}\sum_{i=1}^{n} w_{ij}\,x_j-\theta_i$$

- yi: salida de cada neurona.
- f es la función de activación de cada neurona, dota a las redes neuronales de flexibilidad y estima

relaciones complejas entre todos los datos. Puede ser una función lineal o no lineal. Se representa de a siguiente forma:

$$y_i = f_i(h_i) = f_i\left(\sum_{j=0}^{n} w_{ij}\, x_j - \theta_i\right), con\ y_i \in R$$

Algunos ejemplos de función de activación son:

- Neuronas todo-nada: es una función escalonada, los valores de salida son 1 o 0.

$$y_i = f_i(h_i) = f_i\left(\sum_{j=0}^{n} w_{ij}\, x_j - \theta_i\right)$$

$$y_i = \begin{cases} 1 \ \ si \ \ \sum_{j=1}^{n} w_{ij} x_j \geq \theta_i \\ 0 \ si \ \sum_{j=1}^{n} w_{ij} x_j \leq \theta_i \end{cases}, con\ y_i \ \in [0,1]$$

- Neurona continua sigmoidea: se utiliza cuando queremos obtener una salida continua, son diferenciables y se suelen utilizar en los modelos con multicapas y con la técnica de retropropagación que se explicará más adelante.

$$y_i = \frac{1}{1 + e^{-(\sum_{j=1}^{n} w_{ij} x_j - \theta_i)}}, con\ y_i \ \in [0,1]$$

$$y_i = \frac{e^{(\sum_{j=1}^{n} w_{ij} x_j - \theta_i)} - e^{-(\sum_{j=1}^{n} w_{ij} x_j - \theta_i)}}{e^{(\sum_{j=1}^{n} w_{ij} x_j - \theta_i)} + e^{-(\sum_{j=1}^{n} w_{ij} x_j - \theta_i)}}, con\ y_i \ \in [-1,1]$$

La arquitectura interna de las redes neuronales ha sido nombrada anteriormente, pero se distribuye en diferentes tipos de capas. La capa de entrada es la encargada de recibir los datos procedentes del exterior, la capa de salida que da la respuesta y las capas ocultas que dan grados de libertad a la hora de procesar la información. Existen varios tipos de redes neuronales.

3.1. Redes neuronales prealimentadas

Las primeras en desarrollo y con el modelo más sencillo. La información se transmite hacia delante, los elementos principales son las neuronas (perceptrón) y las capas (perceptrón multicapa). Se utilizan para problemas de clasificación sencillos.

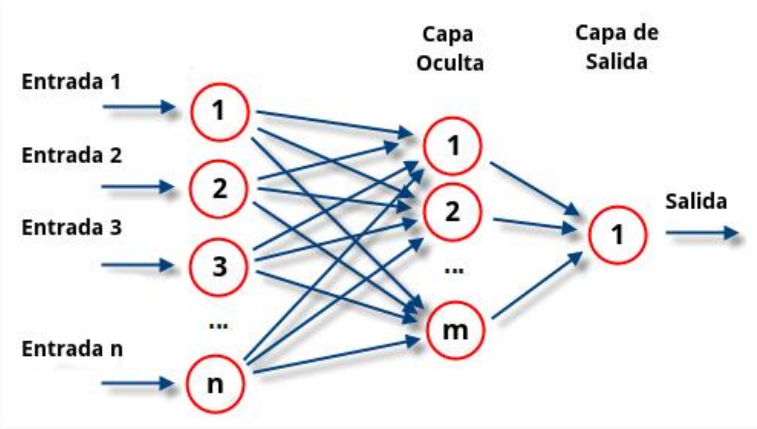

Figura 10: Red neuronal prealimentada (“feedforward”)

3.2. Redes neuronales convolucionales

Son capaces de detectar características simples como por ejemplo detección de bordes, líneas, etc y obtener las características más complejas hasta detectar lo buscado, es muy potente para el análisis de imágenes. Una red neuronal convolucional es una red multicapa que consta de capas convolucionales y de reducción alternadas, y al finalmente tiene capas de conexión total como una red perceptrón multicapa. La convolución consiste en una serie de operaciones de productos y sumas entre la capa de partida y el filtro que genra un mapa de características. La reducción disminuye la cantidad de parámetros al quedarse con las características más comunes [11].

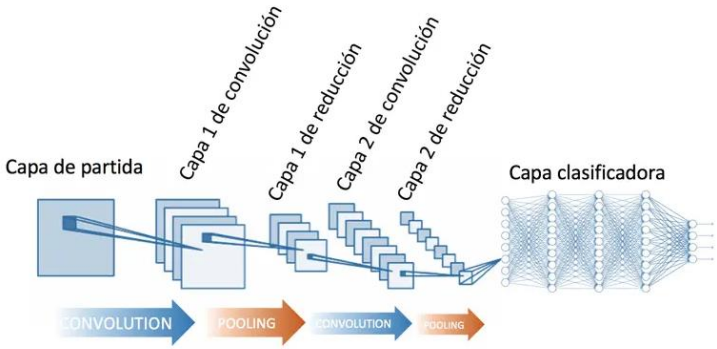

Figura 11: Red neuronal convolucional

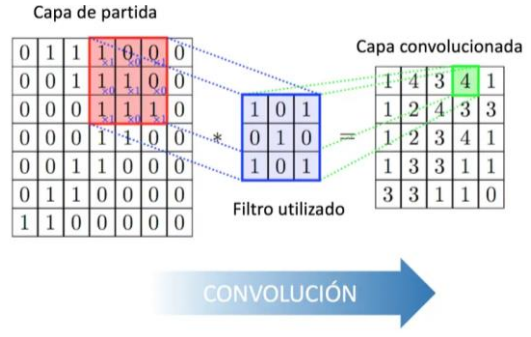

Figura 12: Convolución

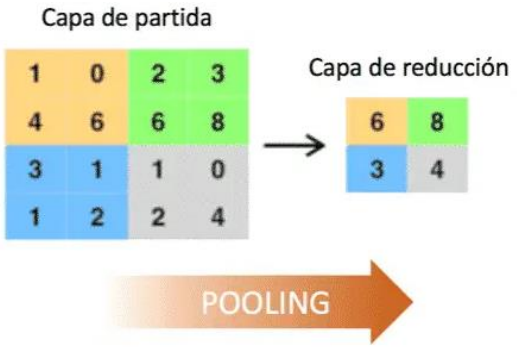

Figura 13: Reducción

3.3. Redes neuronales recurrentes

Son redes con bucles de retroalimentación que hacen posible que parte de la información se mantenga o persista. Consiste en una red con múltiples copias de ella misma, que envían un dato a un sucesor. Se utilizan para trabajar con secuencias y listas. Ha resuelto problemas como reconocimiento de voz, modelado de lenguaje, traducción, subtítulos de imágenes y la lista continúa. Existe un tipo de red neuronal recurrente, la LSTM, permite que la red pueda aprender en largo plazo. Este tipo de red está formada por 4 capas, que interactúan entre ellas.

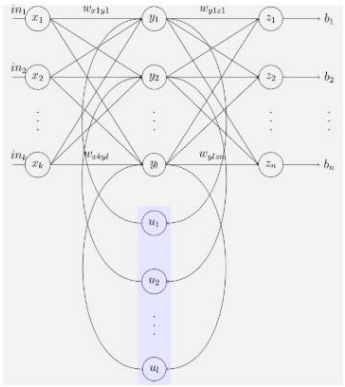

Figura 14: Red neuronal recurrente

3.4. Auto-codificadores (autoencoders)

Redes neuronales de tres capas, sólo una es oculta. Aprende a dar de salida la misma entrada, entrada y salida deben tener el mismo número de neuronas. Se puede dividir la red en dos, compresor y descompresor, porque la capa oculta comprime información de la entrada, se descubre una forma de codificar la información, es un aprendizaje no supervisado [12].

3.5. Auto-codificadores dispersos

¿Qué ocurriría si la capa oculta tiene más neuronas que la capa de entrada y salida? Es posible que la red no aprenda nada. Si forzamos que las neuronas de la capa oculta se activen pocas veces, entonces la red tendría que aprender nuevamente un código alternativo, y además, disperso. Cada neurona se especializa en un patrón de entrada. Qué hacer para no copiar la entrada a la salida:

- Añadir un factor de dispersión
- Introducir un poco de ruido en el vector de entrada y dejar al de salida sin ruido

4 RETROPROPAGACIÓN

Para hacer que un sistema tenga la capacidad de aprender es necesario utilizar un método de cálculo como el backpropagation o la propagación hacia atrás. Este proceso consta de varias fases, en la primera la red neuronal recibe un patrón de entrada, estos datos pasan por todas las capas por la que esté formada la red generando una salida que se compara con la deseada calculando también su error. La segunda fase consiste en que todos los errores de las salidas se propagan hacia atrás, pero las capas ocultas solo reciben una parte de la señal errónea. Este proceso se repite en cada capa hasta que todas las neuronas reciban con cuánto error han contribuido a la primera salida. En esta etapa los pesos pueden modificar su valor [13].

De esta forma se entrena a nuestro modelo para minimizar el error y que en futuras entradas sea capaz de dar un resultado correcto. Las neuronas de las capas intermedias tienen la capacidad de distribuirse a si mismas para reconocer las características de las nuevas entradas. En la última etapa, tras el entrenamiento, cuando la red reciba una entrada con cierto patrón igual al entrenado las neuronas intermedias reconocerán dicho patrón.

Para explicar de forma matemática la propagación hacia atrás explicaremos cómo funciona la salida de cada una de las neuronas:

$$y_j^l = f(\sum_k w_{jk}^l \, x_k^{l-1} + \theta_j^l)$$

- y_j^l: salida de la neurona j-ésima en la capa l.
- f: función de activación neuronal.
- w_{jk}^l : peso que conecta la neurona k-ésima de la capa l-1 con la neurona j-ésima de la capa l.
- θ_j^l: umbral de la neurona j-ésima en la capa l.

Durante la etapa de aprendizaje, el objetico es minimizar el error entre la salida obtenida por la red y la salida deseada. La función de error a minimizar para cada patrón p es la siguiente [14]:

$$E^p = \frac{1}{2}\sum_{k=1}^{M}(d_k^p - y_k^p)^2$$

Donde d_k^p es la salida deseada para la neurona de salida k ante la presentación del patrón p. Para obtener una medida general del error se aplica la siguiente fórmula:

$$E = \sum_{p=1}^{P} E^p$$

El algoritmo en el que se basa la retropropagación para modificar el gradiente se conoce como gradiente decreciente. E^p es función de cada uno de los pesos de la red, por tanto el gradiente de E^p es un vector igual a la derivada parcial de E^p respecto a cada uno de sus pesos. Se escoge la derivada negativa ya que determina el error más rápido.

$$-\sum_{p=1}^{P}\frac{\partial E^p}{\partial w_{ij}}$$

Por tanto para calcular el error asociado a una neurona oculta j viene determinado por la suma de los errores de las k neuronas de salidas que reciben como entrada la salida de la neurona oculta j:

$$\partial_j^p = f(\sum_{i=1}^{n} w_{ji}\, x_i + \theta_j)\sum_{k=1}^{M}\partial_k^p w_{kj}$$

$$\partial_k^p = (d_k^p - y_k^p) f'(w_{ji} x_i + \theta_{j)}$$

5 “FRAMEWORKS”

5.1. Tensorflow

Figura 15: Logo Tensorflow

Es el marco de aprendizaje más popular en el “Deep Learning”, muchas marcas como Google, NVIDIA, IBM, Airbus… lo utilizan y además es una biblioteca de código abierto. El lenguaje recomendado para utilizar este “framework” es Python y necesita mucha codificación. Soporta muchas plataformas y entorno. Funciona con un gráfico de cálculo estático. Es ejecutable tanto en CPU como en GPU [15].

5.2. Pythorch

Figura 16: Logo Pytorch

Es la segunda herramienta de software más utilizada tras TensorFlow. Se desarrolló principalmente para Facebook. Soporta gráficos computacionales dinámicos, es decir se pueden hacer cambios en la arquitectura del proceso. Es adecuado para proyectos pequeños o prototipos y se ejecuta en Python. Una de las principales ventajas es que presenta modelos previamente entrenados [16].

5.3. Chainer

Figura 17: Logo Chainer

Es un marco de aprendizaje potente, dinámico e intuitivo desarrollado para Python. Es posible modificar las redes durante el tiempo de ejecución. Es el “framework” más rápido en comparación con otros y mejor rendimiento.

5.4. Keras

Figura 18: Logo Keras

Capaz de desarrollar el “Deep Learning” y para un gran volumen de datos. Se puede utilizar como una API con

TensorFlow. La creación de modelos se puede reducir en una sola línea, por lo que el código resultante es conciso. Está a un nivel superior que el resto de "frameworks".

5.5. DEEPLEARNING4J

Figura 19: Logo DEEPLEARNING4J

Está escrito para Java, el entrenamiento se realiza de forma paralela y es de código abierto. Se puede ejecutar en GPUs y CPUs. Se puede administrar sobre Hadoop y Spark. Es mucho más eficiente que los que están desarrollados en Python [17].

6 ROS Y HERRAMIENTAS DE DESARROLLO

6.1. Introducción de ROS

ROS (Robot Operating System) [18] es un “framework” que desarrolla el software, basado en Ubuntu, para robots que provee la funcionalidad de un sistema operativo en un clúster heterogéneo. También se puede definir como una colección de herramientas, bibliotecas y convenciones que tienen como objetivo simplificar la tarea de crear un comportamiento robusto y complejo en una amplia variedad de plataformas robóticas. ROS se desarrolló originariamente en 2007 bajo el nombre de switchyard por el Laboratorio de Inteligencia Artificial de Stanford para dar soporte al proyecto del Robot con Inteligencia Artificial de Stanford (STAIR2). Desde 2008, el desarrollo continuó principalmente en Willow Garage, un instituto de investigación robótico con más de veinte instituciones colaborando en un modelo de desarrollo federado.

Figura 20: Logo ROS

A pesar de no ser un sistema operativo, ROS provee los servicios estándar de uno de estos tales como la abstracción del hardware, el control de dispositivos de bajo nivel, la implementación de funcionalidad de uso común, el paso de mensajes entre procesos y el mantenimiento de paquetes. Está basado en una arquitectura de grafos donde el procesamiento toma lugar en los nodos que pueden recibir, mandar y multiplexar mensajes de sensores, control, estados, planificaciones y actuadores, entre otros. La librería está orientada para un sistema UNIX (Ubuntu -Linux-) aunque también se está adaptando a otros sistemas operativos como Fedora, Mac OS X, Arch, Gentoo, OpenSUSE, Slackware, Debian o Microsoft Windows, considerados a día de hoy como ‘experimentales’.

ROS tiene dos partes básicas: la parte del sistema operativo, ros, y ros-pkg. Esta última consiste en una suite de paquetes aportados por la contribución de usuarios (organizados en conjuntos llamados pilas o en inglés “stacks”) que implementan las funcionalidades tales como localización y mapeo simultáneo, planificación, percepción, simulación, etc.

ROS es software libre bajo términos de licencia BSD. Esta licencia permite libertad para uso comercial e investigador. Las contribuciones de los paquetes en ros-pkg están bajo una gran variedad de licencias diferentes.

Aunque está en proceso de desarrollo de aplicaciones, las áreas que ya incluye ROS son:

- Un nodo principal de coordinación.
- Publicación o subscripción de flujos de datos: imágenes, estéreo, láser, control, actuador, contacto, etc.
- Multiplexación de la información.
- Creación y destrucción de nodos.
- Los nodos están perfectamente distribuidos, permitiendo procesamiento distribuido en múltiples núcleos, multiprocesamiento, GPUs y clústeres.
- Login.
- Parámetros de servidor.

- Testeo de sistemas.

Las siguientes áreas ya son posibles de utilizar en ROS para las aplicaciones de procesos:

- Percepción
- Identificación de Objetos
- Segmentación y reconocimiento
- Reconocimiento facial
- Reconocimiento de gestos
- Seguimiento de objetos
- Egomoción
- Comprensión de movimiento
- Estructura de movimientos (SFM)
- Visión estéreo: percepción de profundidad mediante el uso de dos cámaras
- Movimientos
- Robots móviles
- Control
- Planificación
- Agarre de objetos

Una de las características de ROS es que fue diseñado para ser los mas distribuido y modular posible. La modularidad permite seleccionar las partes que son útiles y qué partes se quieres implementar. La distribución de ROS fomenta una gran comunidad de paquetes contribuidas por usuarios que añaden un gran valor a la parte superior del núcleo del sistema ROS. En el último recuento se contaron más de 3000 paquetes que son públicos. La comunidad de usuarios de ROS construye sobre la parte superior de una infraestructura común para ofrecer acceso a controladores de hardware, robots genéricos, herramientas de desarrollo, bibliotecas externas útiles, y mucho más.

6.2. Historia

ROS es un gran proyecto con muchos antepasados y contribuyentes. La necesidad de un marco de colaboración abierto se sintió por muchas personas en la comunidad de investigación en robótica, y se han creado muchos proyectos para este objetivo.

Varios esfuerzos en la Universidad de Stanford a mediados de la década de 2000 que involucraron inteligencia artificial integrada, como el robot STanford AI (STAIR) y el programa Personal Robots (PR), crearon prototipos internos de sistemas de software flexibles y dinámicos destinados al uso de la robótica. En 2007, Willow Garage, una incubadora de robótica visionaria, proporcionó recursos significativos para extender estos conceptos mucho más lejos y crear implementaciones bien probadas. El esfuerzo fue impulsado por innumerables investigadores que contribuyeron con su tiempo y experiencia tanto a las ideas centrales de ROS como a sus paquetes de software fundamentales. En todo momento, el software se desarrolló de forma libre utilizando la licencia permisiva de código abierto BSD, y gradualmente se ha convertido en una plataforma ampliamente utilizada en la comunidad de investigación en robótica.

Desde el principio, ROS se desarrolló en múltiples instituciones y para múltiples robots, incluidas muchas instituciones que recibieron robots PR2 de Willow Garage. Aunque hubiera sido mucho más simple para todos los contribuyentes colocar su código en los mismos servidores, a lo largo de los años, el modelo "federado" se ha convertido en una de las grandes fortalezas del ecosistema ROS. Cualquier grupo puede iniciar su propio repositorio de código ROS en sus propios servidores, y mantienen la propiedad y el control completos de este. No necesitan el permiso de nadie. Si eligen hacer que su repositorio esté disponible públicamente, pueden recibir el reconocimiento y el crédito que se merecen por sus logros, y beneficiarse de comentarios técnicos específicos y mejoras como todos los proyectos de software de código abierto.

6.3. Comunidad activa y colaborativa

Los usuarios de la robótica investigan fuera del laboratorio cada vez más, hay una adopción de ROS en el sector comercial, robótica industrial y de servicios. La comunidad de ROS es muy activa, consta de más de 1500 participantes, más de 3300 usuarios en la wiki de documentación y unos 5700 usuarios en la web de preguntas y respuestas de ROS. El sitio web de preguntas y respuestas tiene 13,000 preguntas hechas hasta la fecha, con una tasa de respuesta del 70%.

ROS por sí mismo ofrece un gran valor a la mayoría de los proyectos de robótica, pero también presenta una oportunidad para establecer contactos y colaborar con los expertos de robots mundiales que forman parte de la comunidad ROS. Una de las filosofías centrales en ROS es el desarrollo compartido de componentes comunes.

6.4. Conceptos básicos

Para entender el funcionamiento de ROS se hará una breve explicación de los conceptos básicos con los que funciona este frameware, nodos, mensajes, topics y servicios.

Un nodo realmente no es mucho más que un archivo ejecutable dentro de un paquete ROS, utilizan una biblioteca de cliente ROS para comunicarse con otros nodos. Los nodos pueden publicar o suscribirse a un tema y proporcionar o usar un servicio. Un sistema se compone de muchos nodos. Uno de los beneficios es la tolerancia a fallos, ya que los bloqueos de nodos son individuales, cada nodo se encarga de una tarea en concreto.

Los nodos se comunican entre si por paso de mensajes. Un mensaje es una estructura de datos simple, que puede ser de distinto tipo (entero, flotante, booleano). Los mensajes pueden estar compuestos de otros mensajes, y matrices de otros mensajes, anidados de forma profunda.

Los topics son los nombres que identifican el contenido de un mensaje donde los nodos se clasifican de dos formas, publicador y suscriptor. Un nodo que está interesado en un determinado tipo de datos se suscribe al tema correspondiente. Puede haber varios editores y suscriptores concurrentes a un mismo tema, y un único nodo puede publicar y suscribirse a múltiples temas. En general, los editores y suscriptores no son conscientes de la existencia de los demás.

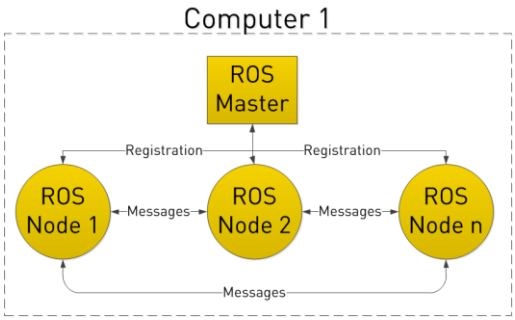

Figura 21: Sistema Básico ROS

Para las transacciones síncronas no es conveniente utilizar la opción de publicación-suscripción de topics, para ello utilizamos los servicios. Es el tipo de arquitectura encargada de realizar la comunicación entre nodos, utilizan dos tipos de mensajes, uno para la solicitud, y otro que es la respuesta dada por el otro nodo a esa petición, por tanto nos podemos encontrar nodos servidores y nodos clientes. Tras realizar la solicitud, el nodo servidor se queda en modo espera hasta recibir respuesta por parte del nodo cliente, y en el caso de que sea el cliente el que realiza una petición, el nodo servidor la procesará y responderá al cliente con la información requerida.

Una de las grandes ventajas de ROS es que hace posible la programación en varios lenguajes, como Python o C++, permitiendo que diferentes nodos estén programados en distintos lenguajes.

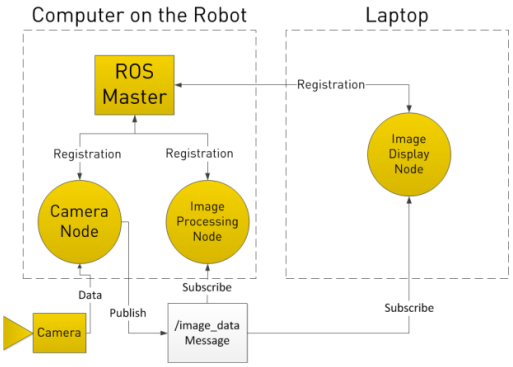

Figura 22: Sistema visión ROS

6.5. "Deep Learning" en ROS

Actualmente buscamos que los robots sean capaces de adaptarse a nuevas situaciones en el entorno, para ello los ingenieros deben desarrollar modelos matemáticos para controlar su funcionamiento. Los sistemas realizados a mano pueden ser rígidos y difícil para adaptarse, por eso este TFG va enfocado a las técnicas de aprendizaje automático, a partir de ciertos datos. El aprendizaje por refuerzo es una subrama del "Machine Learning" donde un agente aprende a optimizar su comportamiento de tal forma que maximiza la suma de una señal de recompensa con el tiempo. La mayor parte de investigación de esta tecnología esta basada en la decisión de Markov [19], que consiste en un proceso de decisión donde hay estados y transiciones entre ellos. Se caracteriza por no garantizar que una acción resulte en un estado particular: en cambio, hay múltiples estados posibles que pueden seguir a una acción, caracterizados por una distribución de probabilidad. Esto explica la observabilidad parcial y estocástica de factores que pueden ser imposibles de predecir con precisión. Para empezar a resolver el problema hay que enfrentarse con estos antecedentes:

- Agente: Esta es la entidad inteligente que estamos tratando de entrenar. Un agente tiene percepciones, una función de recompensa, una política y posibles acciones. Los detalles adicionales dependen de la implementación.
- Percepciones: son entradas para el agente. Suelen ser observaciones recopiladas del mundo. Idealmente, describirían completamente el entorno, pero esto a menudo no es posible, obligando al agente a dar cuenta de un mundo parcialmente observable.
- Secuencia de Percepción: Esta es una secuencia de todas las percepciones observadas desde el comienzo de la vida del agente hasta el instante actual t, denotado como st. Teóricamente, todas las acciones deben determinarse a partir de la secuencia de percepción, por lo que también podemos llamarlo el estado actual del agente.
- Estado del mundo: define el estado del tal como existe objetivamente, independientemente de cómo el agente lo percibe. La secuencia de estados del mundo se denotará como w, con el estado mundial actual siendo wt.
- Acción: este es un comando para realizar. Las acciones pueden ser discretas, lo que indica que una acción debe ser realizada o no, y a continuación indicando la "fuerza" con el que aplicar una acción (por ejemplo, seleccionar un buen ángulo de dirección). Si lo permite el sistema, una política puede devolver múltiples acciones para que ocurran simultáneamente.
- Modelo: Esto representa una reducción del mundo real a un problema simplificado, lo que hace la tarea del agente es más fácil al hacer que aprendan un problema más simple del que es.

6.6. Herramientas de ROS

6.6.1 Gazebo

Es uno de los simuladores más populares compatibles con ROS [20]. Proporciona simulación física completa en soporte 3D, que ofrece una selección de back-end de física, así como emulación de sensor y consulta de estado interfaz. Gazebo ofrece la posibilidad de simular con precisión y eficiencia poblaciones de robots en entornos interiores y exteriores complejos. Genera tanto la realimentación de los sensores como las interacciones físicas entre objetos tratándolos como cuerpos rígidos, a través de gráficos de alta calidad e interfaces programables.

Figura 23: Gazebo

6.6.2 Keras

Keras es una biblioteca para el modelado y entrenamiento de redes neuronales para Python [21] . Tensorflow es uno de sus posibles “backends”. Su propósito es simplificar la creación de redes neuronales sofisticadas, poniendo énfasis el en diseño intuitivo y la facilidad de uso, y para permitir la creación rápida de prototipos. El rendimiento se puede lograr a través de la aceleración con GPUs.

- Modularidad: un modelo puede entenderse como una secuencia o un gráfico solo. Todos los elementos de un modelo de aprendizaje profundo son componentes discretos que se pueden combinar de manera arbitraria.
- Minimalismo: la biblioteca proporciona lo suficiente para lograr un resultado, sin lujos y maximizando la legibilidad.
- Extensibilidad: los nuevos componentes son intencionalmente fáciles de agregar y usar dentro del marco-trabajo, destinado a desarrolladores para probar y explorar nuevas ideas.
- Python: no hay archivos de modelo separados con formatos de archivo personalizados. Todo es Python.

6.6.3 RVIZ

RVIZ (ROS Visualization) [22] es un visualizador 3D que permite la visualización de datos de sensor e información de estado del sistema de ROS. Usando RVIZ, se puede ver la configuración actual en un modelo virtual del robot, además de las representaciones en vivo de los valores de sensor publicados en los topics de ROS, incluyendo datos de cámara, mediciones de sensores de distancia infrarrojos, datos de sonar, etc.

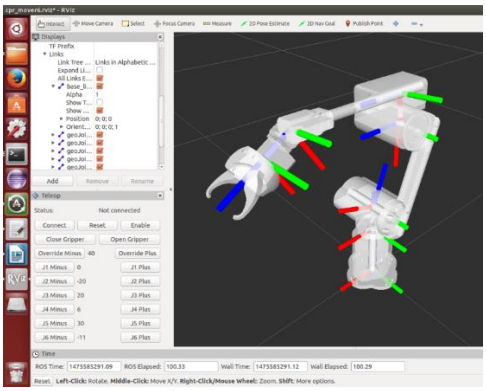

Figura 24: RVIZ

7 PLANTEAMIENTO DETECCIÓN Y RECONOCIMIENTO DE OBJETOS

En este capítulo se plantea todos los pasos a seguir para conseguir la detección y el reconocimiento de objetos. El primer paso es conseguir que se reconozca al robot Mira, mediante un entrenamiento con varias imágenes del objeto. Posteriormente el propio robot reconocerá a los robots de su mismo tipo (Mira) entre otros objetos [23].

7.1. Descripción

Principalmente el objetivo es el reconocimiento del robot Mira. Este robot fue creado por Pixar, tiene un ingenioso sistema de movimiento de pitch, roll y yaw. Mira se utiliza para fines sociales y de diversión

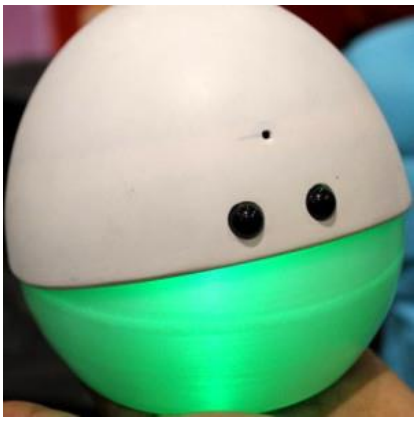

Figura 25: Robot Mira

En primer lugar se debe configurar el espacio de trabajo para trabajar con TensorFlow y Python 3, que serán el “framework” y el lenguaje de programación que se utilizarán. La versión de ROS instalada es Kinetic en el sistema operativo Ubuntu 16.04. Para ello se deben ejecutar los siguientes comandos en la consola:

#Activa el entorno de Python*

cd ~/catkin_ws/src

mkdir -p my_catkin_ws_python3/src

cd ~/catkin_ws/src/my_catkin_ws_python3

source ~/.py3venv/bin/activate

source /home/user/.catkin_ws_python3/devel/setup.bash

catkin_make -DPYTHON_EXECUTABLE:FILEPATH=/home/user/.py3venv/bin/python

source devel/setup.bash

rospack profile

Para crear el paquete en el que se trabajará con dependencias de Python, y de mensajes std_msgs y sensor_msgs se ejecutará:

cd ~/catkin_ws/src/my_catkin_ws_python3/src

#Crea paquete de ROS

catkin_create_pkg my_tf_course_pkg rospy std_msgs sensor_msgs

```
cd ..;source devel/setup.bash;rospack profile
roscd my_tf_course_pkg
#Crea las carpetas launch y scripts dentro del paquete
mkdir launch
mkdir scripts
```

7.2. Entrenamiento del modelo

Ya creado el paquete, para entrenar nuestro propio modelo en Tensorflow se deben seguir los siguientes pasos:

- Etiquetar imágenes.
- Preparar el paquete para el entrenamiento.
- Entrenar al modelo y monitorearlo mediante TensorBroad.

7.2.1 Generación y etiquetado de imágenes

Esta es una de las tareas más lentas, se debe tener un número de imágenes para empezar el entrenamiento. Estas imágenes se han extraído de un git, donde las copia en una carpeta, para ello se deben ejecutar las siguientes sentencias:

```
#Activa entorno de Python
cd ~/catkin_ws/src/my_catkin_ws_python3
source ~/.py3venv/bin/activate
source /home/user/.catkin_ws_python3/devel/setup.bash
catkin_make -DPYTHON_EXECUTABLE:FILEPATH=/home/user/.py3venv/bin/python
source devel/setup.bash
rospack profile
roscd my_tf_course_pkg
#Elimina archivos si hay restos de otros entrenamientos
rm -rf course_tflow_image_student_data
#Descarga conjunto de fotos
git clone https://bitbucket.org/theconstructcore/course_tflow_image_student_data.git
#Elimina y luego crea la carpeta data
rm -rf data
mkdir data
#Elimina y luego crea la carpeta images
rm -rf images
mkdir images
#Copia las imágenes descargada en la carpeta images
cp -a course_tflow_image_student_data/images_1_labels/. images
```

Se generan varias carpetas, de las que se destacarán:

- images_1_labels: solo tiene UNA etiqueta [mira_robot]

- images_2_labels_complete: tiene DOS etiquetas [mira_robot, object]. Todos los objetos en la escena fueron etiquetados con el objeto etiqueta, mientras que todos los Mira Robots fueron etiquetados con mira_robot

Para etiquetar las imágenes, se usará un programa llamado LabelImg, que genera archivos .xml basado en imágenes y cómo los etiqueta.

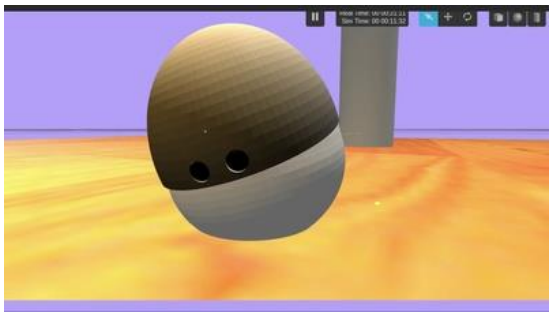

Figura 26: Imagen robot Mira etiquetada por LableImg

Figura 27: Imagen robot Mira-prueba 1

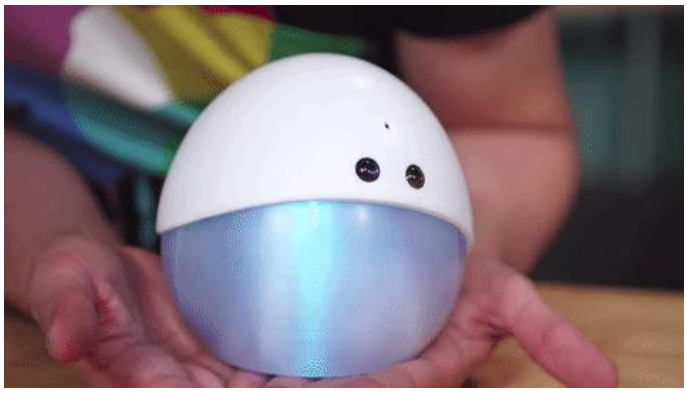

Figura 28: Imagen robot Mira-prueba 2

Para separar las imágenes en test y entrenamiento se utilizará el código separate_dataset_train_test.py, un 90% se destinará a entrenar y el 10% restante a realizar las pruebas. Este número es decisión propia, para estar seguro si el modelo funciona, se pueden colocar más imágenes en la carpeta de prueba. Un requisito fundamental es que las imágenes de entrenamiento no estén en la de pruebas, ya que no tendría sentido evaluar una imagen ya entrenada.

7.2.2 Preparar datos para el entrenamiento

TensorFlow no utiliza archivos .xml. En su lugar, utiliza un tipo de archivo llamado .records. Entonces, hay que

convertir los archivos xml.

Este procedimiento se divide en dos pasos principales:

- Convertir XML a CSV
- Convertir CSV a RECORD

7.2.2.1 Convertir de XML a CSV

CSV es formato simple de datos en línea que separa los valores mediante un carácter, el programa que se utilizará será xml_to_csv.py. Se encargará de crear dos archivos en la carpeta data, test_labels.csv y train_labels.csv. Para empezar la conversió ejecutamos:

roscd my_tf_course_pkg

python scripts/xml_to_csv.py

7.2.2.2 Convertir de CSV a RECORD

Pero hay que dar un paso más. Se necesitan combinar todas las imágenes y los datos csv en un solo archivo al que sea fácil acceder. Estos archivos se denominan archivos tfrecord. Para generarlos, debe ejecutar otro script de Python:

- generate_tfrecord_n.py
- extract_training_lables_csv.py

En extract_training_lables_csv.py, extrae las etiquetas que va a utilizar. En ete caso solo hemos utilizado una etiqueta, que es mira_robot, de los archivos csv generados anteriormente.

Este script también genera automáticamente el número de clases etiquetadas en los archivos csv. Estos están escritos en **. Nombres de archivos. Para llevar a cabo está acción se ejecutan las siguientes sentencias:

rm -rf data/train.tfrecord data/test.tfrecord

python scripts/generate_tfrecord_n.py --image_path_input=images/train --csv_input=data/train_labels.csv --output_path=data/train.tfrecord --output_path_names=data/train_new_names.names

python scripts/generate_tfrecord_n.py --image_path_input=images/test --csv_input=data/test_labels.csv --output_path=data/test.tfrecord --output_path_names=data/test_new_names.names

ls data | grep .tfrecord

cat data/train_new_names.names

cat data/test_new_names.names

cp data/train_new_names.names data/new_names.names

cat data/new_names.names

Aquí generamos los archivos tfrecord y verificamos también que los nombres de clase sean los mismos para el entrenamiento y las pruebas. Esto se debe a que necesitamos que en test_new_names.names pueda encontrar todas las clases definidas. En este caso solo debería aparecer la etiqueta de Mira.

7.2.2.3 Verificar los archivos RECORD

Este paso no es necesario, pero verifica que los pasos anteriores están bien realizados. Debe saber cómo leer lo que hay dentro de un archivo .tfrecord, para poder verificar los archivos de otras personas y asegurarse de que todo salió bien. De esta tarea se encarga elcódigo visualize_dataset.py convierte el archivo de registro a un string. Luego puede guardar esa salida en un archivo de registro. como este:

python scripts/visualize_dataset.py --classes=./data/new_names.names --dataset=./data/train.tfrecord

python scripts/visualize_dataset.py --classes=./data/new_names.names --dataset=./data/test.tfrecord

Debería haberse generado una imagen de salida en ./output.jpg, con el cuadro especificado en la fase de etiquetado. Se generarán tantas imágenes diferentes como veces se ejecuten estas líneas. Como se puede comprobar el tamaño y la forma de la imagen cambia y aparece el robot recuadrado identificándolo y ubicándolo.

Figura 29: Robot Mira verificado 1

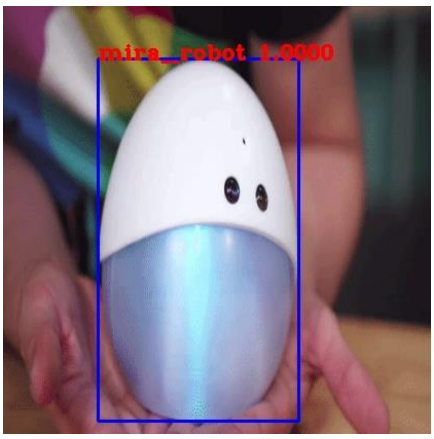

Figura 30: Robot Mira verificado 2

7.2.3 Entrenamiento

Este problema se resolverá con YOLO [24], un algoritmo que utiliza redes neuronales convolucionales para la detección de objetos. La ventaja de utilizar este tipo de redes neuronales es que además de predecir las etiquetas, también detecta la ubicación del objeto.

El código que se encarga de entrenar el modelo es train.py. Todas las variables definidas tienen una función; existen 13 que son las siguientes [25]:

- dataset: ruta al conjunto de datos.
- val_dataset: ruta al conjunto de datos de validación.
- tiny: variable booleana que elige entre yolov3 o yolov3-tiny, se diferencian en fps en un video.
- weights: ruta al archivo donde se encuentra los pesos de entrenamiento.
- classes: ruta a las clases.
- mode: lista de modos posibles.
- transfer: lista de tipos de transferencia.
- size: tamaño de la imagen, 416.

- epochs: número de periodos, 2.
- batch_size: tamaño del lote, 8.
- learning_rate_ frecuencia de aprendizaje, 1e-3.
- num_classes: número de clases, 80.
- weight_num_classes: útil en el aprendizaje de transferencia con diferentes números de clases.

Sólo hay definida una función en el programa, main, donde el programa se divide en varias partes. En la primera parte se configura la GPU, seguidamente se elige entre YoloV3Tiny o YoloV3, la diferencia entre ellas es q la primera tiene menos capas, 2 a la salida, y predice peor los objetos más pequeños.

Luego, se cargan los datos del tipo .tfrecord, especificando la ruta, las clases y el tamaño de la imagen. Esta ación se hace con las imágenes de prueba y de validación. Se configura el modelo de aprendizajede transferencia, si la opción es "darknet, no_output" se coge el modelo preentrenado según el modelo de Yolo que se haya escogido. El código se optimiza con una función de Keras y se depura.

Se analizan los dos tipos de datos que se necesitan, los datos de entrenamiento y los datos de validación. Por último, se utiliza el TensorBroad para comparar los resultados finales.

Para ejecutar el programa se escribirán los siguientes comandos:

```
roscd my_tf_course_pkg
num_new_classes=$(wc -l < data/new_names.names)
echo $num_new_classes

num_coco_classes=$(wc -l < coco.names)
echo $num_coco_classes
rm -rf logs

#Ejecutmos el programa de entrenamiento dándole las entradas necesarias y configuradas anteriormente
python scripts/train.py \
--dataset ./data/train.tfrecord \
--val_dataset ./data/test.tfrecord \
--classes ./data/new_names.names \
--num_classes $num_new_classes \
--mode fit --transfer darknet \
--batch_size 16 \
--epochs 20 \
--weights ./checkpoints/yolov3.tf \
--weights_num_classes $num_coco_classes
```

Aparecen las siguientes líneas en la ventana de comando:

```
INFO:tensorflow:global step 182: loss = 1.4935 (4.103 sec/step)
INFO:tensorflow:global step 183: loss = 1.7676 (3.408 sec/step)
INFO:tensorflow:global step 184: loss = 1.8432 (3.273 sec/step)
INFO:tensorflow:global step 185: loss = 2.3316 (3.165 sec/step)
INFO:tensorflow:global step 186: loss = 1.8591 (3.387 sec/step)
INFO:tensorflow:global step 187: loss = 1.6603 (3.416 sec/step)
INFO:tensorflow:global step 188: loss = 1.9711 (3.407 sec/step)
INFO:tensorflow:global step 189: loss = 1.9095 (3.231 sec/step)
INFO:tensorflow:global step 190: loss = 1.7817 (3.286 sec/step)
INFO:tensorflow:global step 191: loss = 1.6739 (3.168 sec/step)
INFO:tensorflow:global step 192: loss = 1.5430 (3.245 sec/step)
INFO:tensorflow:global step 193: loss = 1.8834 (3.266 sec/step)
INFO:tensorflow:global step 194: loss = 2.4093 (3.162 sec/step)
```

Figura 31: Líneas después de ejecutar train.py

Para visualizar cómo avanzan los datos utilizaremos Tensorboard, que es una herramienta integrada de Tensorflow que permite monitorear la capacitación a través de los archivos de registro publicados por el script de capacitación. Se ejecuta en otra ventana de comando:

roscd my_tf_course_pkg

source ~/.py3venv/bin/activate

#Para visualizer la función de pérdida

tensorboard --logdir=logs/train --host 0.0.0.0

Después de aproximadamente 2 horas se obtiene la siguiente gráfica. TotalLoss (función de pérdida) se ha estabilizado alrededor del valor de 1.00. Se considera que es un modelo que ya aprendió y no mejorará. Se tendrá que detener el proceso de aprendizaje cuando TotalLoss se estabiliza y mantiene su valor.

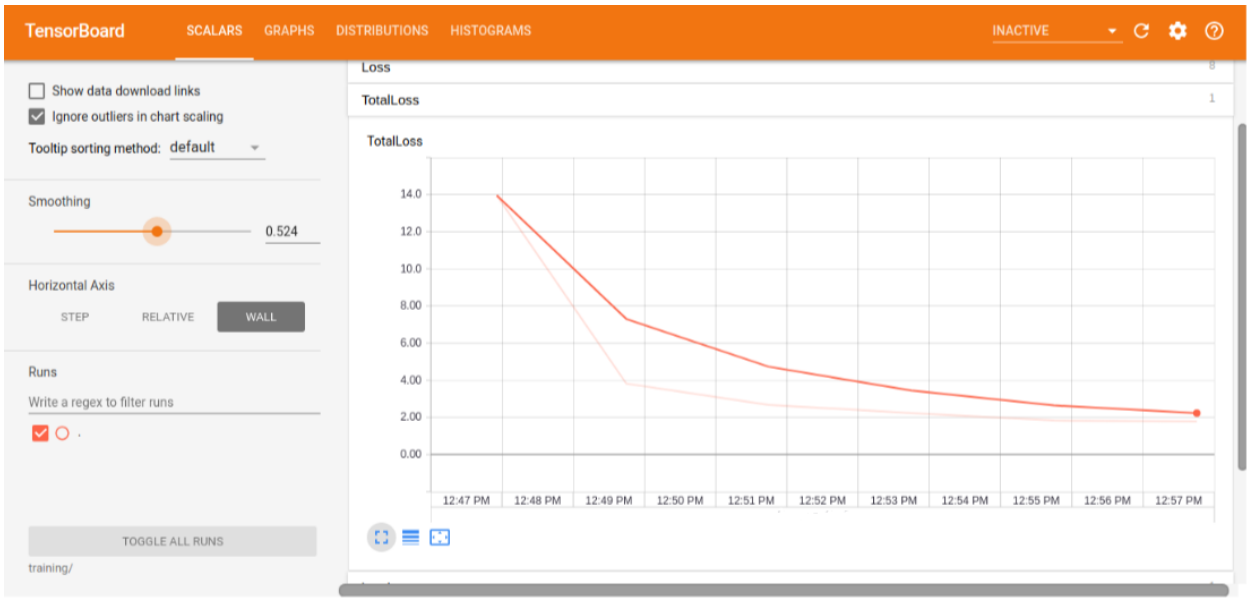

Figura 32: Gráfica estabilidad del modelo

8 RESULTADOS Y CONCLUSIÓN

Para conseguir que el robot sea capaz de identificar a otros robots Mira como él y algunos objetos, hay que finalizar con el siguiente código, search_for_mira_robot.py, para ejecutarlo se siguen las siguientes sentencias:

python scripts/search_for_mira_robot.py \

--classes ./checkpoint_working_twolabels/checkpoints/new_names.names \

--num_classes $num_new_classes \

--weights ./checkpoint_working_twolabels/checkpoints/yolov3_train_20.tf \

El robot de enmedio será el que se encargará de las fotos y el reconocimiento de los objetos que tiene alrededor, detectando cada uno de los objetos y clasificándolos según su clase.

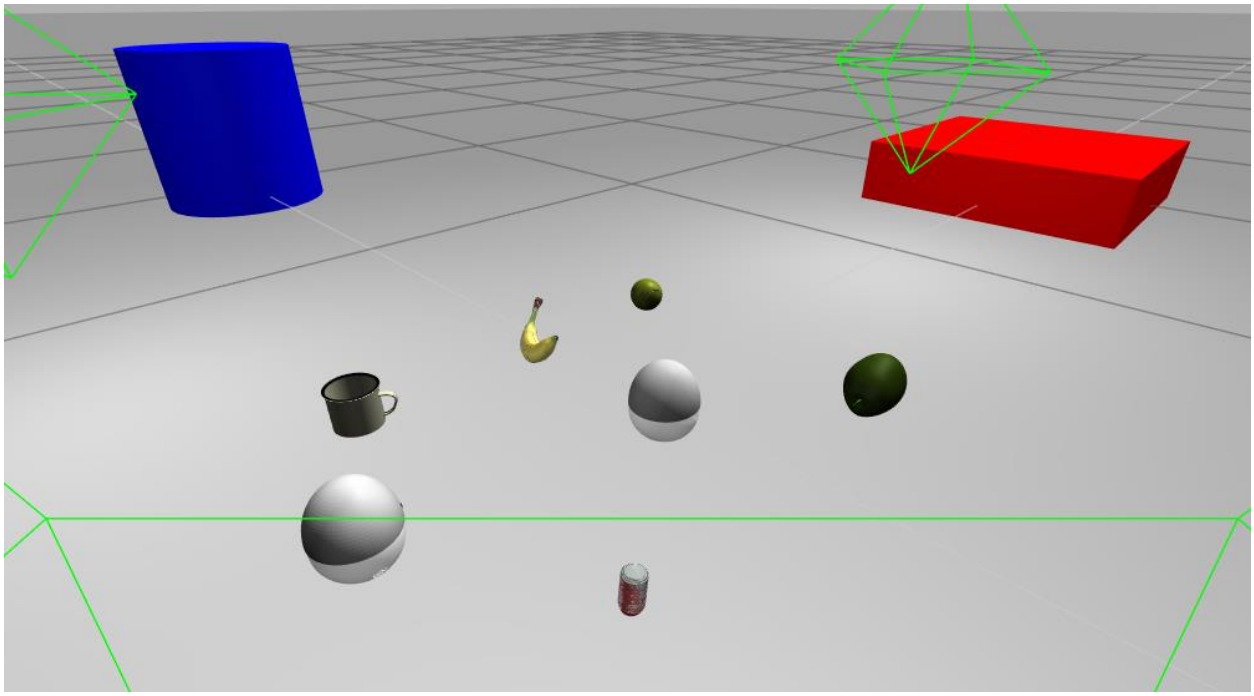

Figura 33: Conjunto de robot y objetos a reconocer

El entrenamiento dura aproximadamente 15 horas, donde las imágenes reconocidas serán “object” que será cualquier objeto y “mira_robot” si localiza algún robot de este tipo. Para visualizar la cámara de robot:

rosrun rqt_image_view rqt_image_view

La gráfica siguiente representa la complejidad y el tiempo necesario para entrenar a un modelo en el que debe identificar varios objetos. La función de pérdida se estabiliza sobre el valor de 1.50 después de varias horas pero durante todo el tiempo su valor fluctúa.

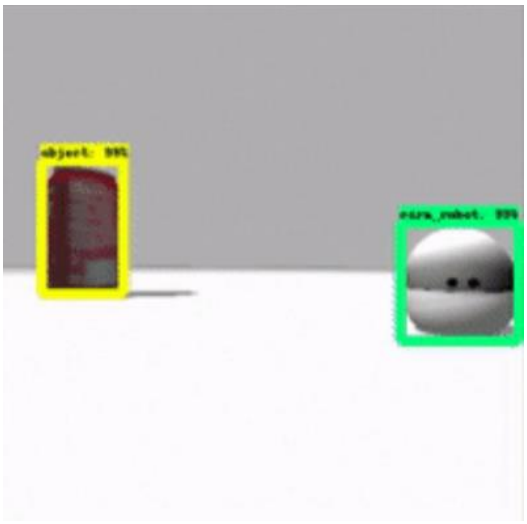

Figura 34: Identificación objetos 1

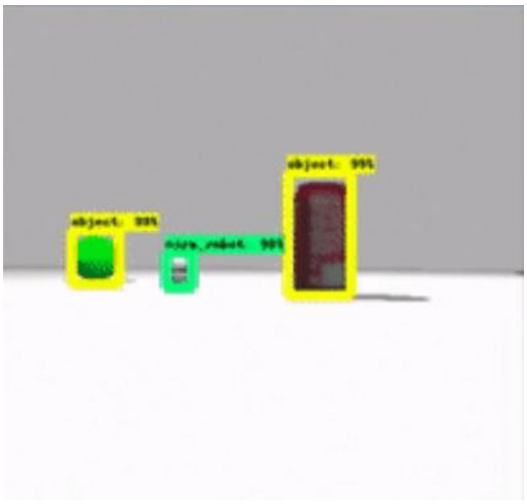

Figura 35: Identificación objetos 2

Figura 36: Gráfica de estabilidad del modelo complejo

Finalizando este trabajo y dejando propuestas para el futuro, una opción es crear un recolector de basura, GarbageRocollector robot, en el que sea capaz de identificar los objetos que se encuentra mediante la camara y decidir si es basura o no.

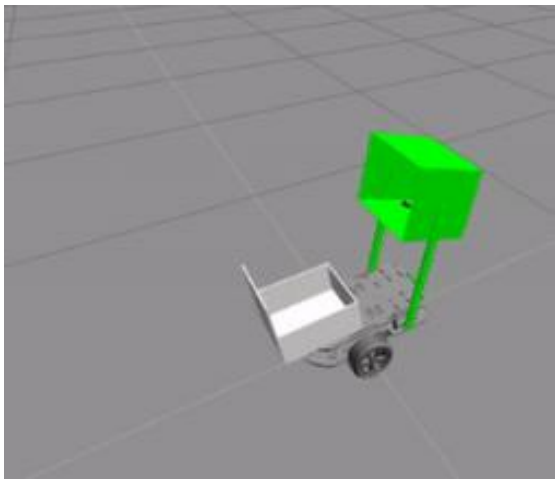

Figura 37: GarbageCollector Robot

REFERENCIAS

[1] «Historia del “Deep Learning” Etapas,» 8 enero 2019. [En línea]. Disponible: https://www.datahack.es/historia-deep-learning-etapas/.

[2] «Historia del “Deep Learning” Pioneros,» 10 enero 2019. [En línea]. Disponible: https://www.datahack.es/historia-deep-learning-pioneros/.

[3] «Historia del “Deep Learning” Hitos,» 14 enero 2019. [En línea]. Disponible: https://www.datahack.es/historia-del-deep-learning-3/.

[4] A. G. Serrano, «Inteligencia Artificial: Fundamentos, práctica y aplicaciones,» 2012. [En línea]. Disponible: www.rclibros.es/pdf/InteligenciaArtificial.pdf.

[5] V. Román, «Introucción al “Machine Learning”: Una guía desde cero,» 6 febrero 219. [En línea]. Disponible: https://medium.com/datos-y-ciencia/introduccion-al-machine-learning-una-guía-desde-cero-b696a2ead359.

[6] G. G. Gutiérrez y A. R. Gutiérrez, «Introducción Aprendizaje de Máquinas,» [En línea]. Disponible: c3.itm.edu.co/~gerardo.gutierrez/machinelearning/Introducción%20Aprendizaje%20de%20Máquina.pdf.

[7] E. O. Peralta, «Qué es el “Deep Learning”,» [En línea]. Disponible: https://www.genwords.com/blog/deep-learning.

[8] R. Arrabales, «“Deep Learning”: qué es y por qué va a ser una tecnología clave en el futuro de la inteligencia artificial,» 28 octubre 2016. [En línea]. Disponible: https://www.xataka.com/robotica-e-ia/deep-learning-que-es-y-por-que-va-a-ser-una-tecnologia-clave-en-el-futuro-de-la-inteligencia-artificial.

[9] «Introducción al “Deep Learning”,» 13 junio 2017. [En línea]. Disponible: https://iaarhub.github.io/.

[10] P. Larrañaga, I. Inza y A. Moujahid, «Redes neuronales,» [En línea]. Disponible: http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t8neuronales.pdf.

[11] D. Calvo, «Red Neuronal Convolucional CNN,» 20 julio 2017. [En línea]. Disponible: https://www.diegocalvo.es/red-neuronal-convolucional/#:~:text=Arquitectura,como%20una%20red%20perceptr%C3%B3n%20multicapa..

[12] «¿Qué es y cómo funciona ““Deep Learning””?,» 7 mayo 2014. [En línea]. Disponible: https://rubenlopezg.wordpress.com/2014/05/07/que-es-y-como-funciona-deep-learning/.

[13] «Wikipedia: Propagación hacia atrás,» [En línea]. Disponible: https://es.wikipedia.org/wiki/Propagación_hacia_atrás.

[14] «Introducción a las redes neuronales aplicadas,» [En línea]. Disponible: http://halweb.uc3m.es/esp/Personal/personas/jmmarin/esp/DM/tema3dm.pdf.

[15] «Top 10 Best Deep Learning Frameworks in 2019,» 3 junio 2019. [En línea]. Disponible:

https://towardsdatascience.com/top-10-best-deep-learning-frameworks-in-2019-5ccb90ea6de.

[16] «Top 10 Machine Learning Frameworks,» 9 agosto 2019. [En línea]. Disponible: https://hackernoon.com/top-10-machine-learning-frameworks-for-2019-h6120305j.

[17] «Top 8 Deep Learning Frameworks,» [En línea]. Disponible: https://marutitech.com/top-8-deep-learning-frameworks/.

[18] «ROS,» [En línea]. Disponible: https://www.ros.org/.

[19] V. Sereda, *Machine Learning For Robots with ROS,* Irlanda, 2017.

[20] «Gazebo,» [En línea]. Disponible: http://gazebosim.org/.

[21] «Keras,» [En línea]. Disponible: https://keras.io/.

[22] «RVIZ,» [En línea]. Disponible: http://wiki.ros.org/rviz.

[23] «Robot Ignite Academy,» [En línea]. Disponible: https://www.robotigniteacademy.com/.

[24] «Yolo v3 Object Detection in Tensorflow,» [En línea]. Disponible: https://www.kaggle.com/aruchomu/yolo-v3-object-detection-in-tensorflow.

[25] «Github,» [En línea]. Disponible: https://github.com/zzh8829/yolov3-tf2.

ANEXO A: ESPECIFICACIONES DE LA MÁQUINA UTILIZADA

Este trabajo se ha realizado con un ordenador personal, un portátil ASUS F541U de las siguientes características:

- Procesador Intel Core i5-7200U (2 Núcleos, 3M Cache, 2.5GHz hasta 3.1GHz)
- Memoria RAM 8GB DDR4 2133MHz
- Disco duro 1TB 5400rpm SATA
- Pantalla 15.6" FHD (1366x768/16:9)
- Controlador gráfico NVIDIA GeForce 920M 2GB DDR3 VRAM

De este ordenador portátil se ha aprovechado la tarjeta gráfica GeForce, que es compatible para configurar TensorFlow GPU. Esta configuración ha permitido reducir los tiempos de entrenamiento en comparación a los entrenamientos realizados con CPU's. Los tiempos de entrenamiento se podrían reducir con un ordenador con mejores características en el controlador gráfico.

ANEXO B: CÓDIGOS

En este anexo se exponen todos los códigos citados en la memoria para llevar a cabo el objetivo de este trabajo.

- separate_dataset_train_test.py: Código en Python que separa los datos para entrenar y para testear.
- xml_to_csv.py: Código en Python que convierte el archivo de imagen .xml en .csv.
- generate_tfrecord_n.py: Código en Python que transforma los archivos .csv en .tfrecord.
- extract_training_lables_csv.py: Código en Python que extrae las etiquetas que se van a usar para el entrenamiento.
- visualize_dataset.py: Código en Python que verifica que los archivos .tfrecord son correctos y con la etiqueta correspondiente.
- train.py: Código en Python que se encarga de entrenar el modelo.
- search_for_mira_robot.py: Código en Python en el que el robot Mira visualiza e identifica otros robots como él y los diferencia de otros objetos.

separate_dataset_train_test.py

```
import os
import random
import copy
from shutil import copyfile
from shutil import rmtree
def generate_list_files_dataset_clean(dataset_dir_path = "./new_dataset"):

    images_list, annotation_list = check_annotations_vs_images(dataset_dir_path,
assert_active=False)

    for i in images_list[:]:
      image_name_xml = os.path.splitext(i)[0]+".xml"
      if image_name_xml not in annotation_list:
            images_list.remove(i)

    for j in annotation_list[:]:
      annotation_name_jpg = os.path.splitext(j)[0]+".jpg"
      annotation_name_png = os.path.splitext(j)[0]+".png"
```

```
            if annotation_name_jpg not in images_list and annotation_name_png not in images_list:
                    annotation_list.remove(j)

        annotation_length = len(annotation_list)
        images_length = len(images_list)
        print("Annotations="+str(annotation_length))
        print("Images="+str(images_length))

        if annotation_length != images_length:
            assert False,"Lengths are note the same...Something is wrong"
        else:
            print("Lengths ok and CLEAN")

        return annotation_list, images_list

def divide_dataset(image_list, ratio=0.8):

        train_image_list = []
        test_image_list = []
        len_image_list = len(image_list)
        len_train_image_list = int(len_image_list * ratio)
        len_test_image_list = len_image_list - len_train_image_list
        assert (len_train_image_list + len_test_image_list) == len_image_list, "Total Length wrong"
        for i in range(len_test_image_list):
                test_image = random.choice(image_list)
                image_list.remove(test_image)
                test_image_list.append(test_image)

        train_image_list = copy.deepcopy(image_list)

        len_train_image_list = int(len(train_image_list))
        len_test_image_list = int(len(test_image_list))
```

```
        print("len_train_image_list="+str(len_train_image_list))
        print("len_test_image_list="+str(len_test_image_list))

        assert (len_train_image_list + len_test_image_list) == len_image_list, "Total Length wrong
AFTER Generation"

        return train_image_list, test_image_list

def dataset_copy_files_to_folders(train_image_list, test_image_list, dataset_dir_path =
"./new_dataset", train_test_dataset_dir_path = "./new_dataset"):

        abs_dataset_dir_path = os.path.abspath(dataset_dir_path)
        train_images_folder = os.path.abspath(os.path.join(train_test_dataset_dir_path, "train"))
        validation_images_folder = os.path.abspath(os.path.join(train_test_dataset_dir_path, "test"))

        copy_files_to_folder(abs_dataset_dir_path, train_images_folder, train_image_list)
        copy_files_to_folder(abs_dataset_dir_path, validation_images_folder, test_image_list)

        print("Checking the train folder files")
        check_list_files(train_image_list, train_images_folder)

        print("Checking the test folder files")
        check_list_files(test_image_list, validation_images_folder)

def copy_files_to_folder(origin_folder, destination_folder, image_list):

        if os.path.exists(destination_folder):
                rmtree(destination_folder)

        os.makedirs(destination_folder)
        print("Created folder=" + str(destination_folder))
```

```
        for image_file_name in image_list:
                image_file_name_xml = os.path.splitext(image_file_name)[0]+".xml"
                abs_path_file = os.path.abspath(os.path.join(origin_folder,image_file_name))
                abs_path_file_xml = os.path.abspath(os.path.join(origin_folder,image_file_name_xml))

                abs_path_file_destination =
os.path.abspath(os.path.join(destination_folder,image_file_name))
                abs_path_file_destination_xml =
os.path.abspath(os.path.join(destination_folder,image_file_name_xml))

                copyfile(abs_path_file, abs_path_file_destination)
                copyfile(abs_path_file_xml, abs_path_file_destination_xml)

def check_annotations_vs_images(folder_path, assert_active=False):
        annotation_list= []
        images_list = []
        for file_name in os.listdir(folder_path):
           if file_name.endswith(".xml"):
                   annotation_list.append(file_name)
           if file_name.endswith(".jpg") or file_name.endswith(".png"):
                   images_list.append(file_name)

        annotation_length = len(annotation_list)
        images_length = len(images_list)
        print("Annotations="+str(annotation_length))
        print("Images="+str(images_length))

        if annotation_length != images_length:
                msg = "Lengths are note the same...Something is wrong"
                if assert_active:
                        assert False,msg
```

```
                else:
                        print(msg)
        else:
                print("Lengths ok and CLEAN")

        return images_list, annotation_list


def check_list_files(list_files, folder_path):

        images_list, annotation_list = check_annotations_vs_images(folder_path)

        if len(list_files) != len(images_list):
                assert False, "The images in the folder dont coincide"
        else:
                print("Number of files n folder and in list are the same, so everything OK")

annotation_list, images_list = generate_list_files_dataset_clean()
train_image_list, test_image_list = divide_dataset(images_list, ratio=0.8)
dataset_copy_files_to_folders(train_image_list, test_image_list, dataset_dir_path = "./new_dataset",
train_test_dataset_dir_path = "./new_dataset")
```

xml_to_csv.py

```
import os
import glob
import pandas as pd
import xml.etree.ElementTree as ET

def xml_to_csv(path):
   xml_list = []
   for xml_file in glob.glob(path + '/*.xml'):
```

```
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
            value = (root.find('filename').text,
                     int(root.find('size')[0].text),
                     int(root.find('size')[1].text),
                     member[0].text,
                     int(member[4][0].text),
                     int(member[4][1].text),
                     int(member[4][2].text),
                     int(member[4][3].text)
                     )
            xml_list.append(value)
    column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    return xml_df


def main():
    for directory in ['train', 'test']:
        image_path = os.path.join(os.getcwd(), 'images/{}'.format(directory))
        xml_df = xml_to_csv(image_path)
        xml_df.to_csv('data/{}_labels.csv'.format(directory), index=None, encoding='utf-8')
        print('Successfully converted xml to csv.')


main()
```

generate_tfrecord_n.py

```
from __future__ import division
from __future__ import print_function
from __future__ import absolute_import

import os
```

```
import io
import pandas as pd
import numpy as np
import tqdm
import tensorflow as tf

from PIL import Image
from collections import namedtuple, OrderedDict
from extract_training_lables_csv import extract_training_labels_csv, class_text_to_int

from absl import app, flags, logging
from absl.flags import FLAGS
import hashlib

FLAGS = flags.FLAGS
flags.DEFINE_string('image_path_input', '', 'Path to the Images refered to in the CSV')
flags.DEFINE_string('csv_input', '', 'Path to the CSV input')
flags.DEFINE_string('output_path', '', 'Path to output TFRecord')
flags.DEFINE_string('output_path_names', '', 'Path to output New Class Names file')


def split(df, group):
    data = namedtuple('data', ['filename', 'object'])
    gb = df.groupby(group)
    return [data(filename, gb.get_group(x)) for filename, x in zip(gb.groups.keys(), gb.groups)]


def create_tf_example(group, path, unique_label_array):
    img_path = os.path.join(path, '{}'.format(group.filename))
    image = open(img_path, 'rb').read()
    key = hashlib.sha256(image).hexdigest()
```

```
filename = group.filename.encode('utf8')
image_format = b'jpg'
xmins = []
xmaxs = []
ymins = []
ymaxs = []
classes_text = []
classes = []
truncated = []
views = []
difficult_obj = []



truncated_hardcoded = 0
difficult_hardcoded = 0
view_hardcoded = "Unspecified"

for index, row in group.object.iterrows():
    width = int(row['width'])
    height = int(row['height'])
    xmins.append(float(row['xmin'] / width))
    xmaxs.append(float(row['xmax'] / width))
    ymins.append(float(row['ymin'] / height))
    ymaxs.append(float(row['ymax'] / height))
    classes_text.append(row['class'].encode('utf8'))
    classes.append(class_text_to_int(row['class'], unique_label_array))

    difficult = bool(int(difficult_hardcoded))
    difficult_obj.append(int(difficult))

    truncated.append(int(truncated_hardcoded))
```

```
        views.append(view_hardcoded.encode('utf8'))


    tf_example = tf.train.Example(features=tf.train.Features(feature={
        'image/height': tf.train.Feature(int64_list=tf.train.Int64List(value=[height])),
        'image/width': tf.train.Feature(int64_list=tf.train.Int64List(value=[width])),
        'image/filename': tf.train.Feature(bytes_list=tf.train.BytesList(value=[filename])),
        'image/source_id': tf.train.Feature(bytes_list=tf.train.BytesList(value=[filename])),
        'image/key/sha256': tf.train.Feature(bytes_list=tf.train.BytesList(value=[key.encode('utf8')])),
        'image/encoded': tf.train.Feature(bytes_list=tf.train.BytesList(value=[image])),
        'image/format': tf.train.Feature(bytes_list=tf.train.BytesList(value=['jpeg'.encode('utf8')])),
        'image/object/bbox/xmin': tf.train.Feature(float_list=tf.train.FloatList(value=xmins)),
        'image/object/bbox/xmax': tf.train.Feature(float_list=tf.train.FloatList(value=xmaxs)),
        'image/object/bbox/ymin': tf.train.Feature(float_list=tf.train.FloatList(value=ymins)),
        'image/object/bbox/ymax': tf.train.Feature(float_list=tf.train.FloatList(value=ymaxs)),
        'image/object/class/text': tf.train.Feature(bytes_list=tf.train.BytesList(value=classes_text)),
        'image/object/class/label': tf.train.Feature(int64_list=tf.train.Int64List(value=classes)),
        'image/object/difficult': tf.train.Feature(int64_list=tf.train.Int64List(value=difficult_obj)),
        'image/object/truncated': tf.train.Feature(int64_list=tf.train.Int64List(value=truncated)),
        'image/object/view': tf.train.Feature(bytes_list=tf.train.BytesList(value=views)),

    }))


    return tf_example

def create_class_names_file(unique_class_list, names_file_path="./new_class_names.names"):
    print("Generating New Names Class list file==>"+str(names_file_path))
    with open(names_file_path, 'w') as f:
        for item in unique_class_list:
            f.write("%s\n" % item)
```

```
            print("New Class to file==>"+str(item))
    print("Generating New Names Class list file...DONE")



def main(_argv):
    writer = tf.io.TFRecordWriter(FLAGS.output_path)

    path = os.path.join(os.getcwd(), FLAGS.image_path_input)
    examples = pd.read_csv(FLAGS.csv_input)
    unique_label_array = extract_training_labels_csv(examples)

    create_class_names_file(unique_class_list=unique_label_array,
                            names_file_path=FLAGS.output_path_names)
    grouped = split(examples, 'filename')
    for group in tqdm.tqdm(grouped):
        tf_example = create_tf_example(group, path, unique_label_array)
        writer.write(tf_example.SerializeToString())

    writer.close()
    output_path = os.path.join(os.getcwd(), FLAGS.output_path)
    print('Successfully created the TFRecords: {}'.format(output_path))



if __name__ == '__main__':
    app.run(main)
```

extract_training_lables_csv.py

```
import pandas as pd
import numpy as np

def extract_training_labels_csv(csv_object):
    unique_label_set = set([])
```

```
    d = csv_object.loc[: , "class"]
    for key,value in d.iteritems():
        unique_label_set.add(value)

    unique_label_list = list(unique_label_set)
    unique_label_list.sort()
    return unique_label_list

def class_text_to_int(row_label, unique_label_array):
    class_integer = int(unique_label_array.index(row_label) + 1)
    return class_integer

if __name__ == "__main__":
    print("Opening CSV...")
    csv_input_for_labels = "scripts/dummy1.csv"
    examples = pd.read_csv(csv_input_for_labels)
    print("Opened CSV...")
    unique_label_array = extract_training_labels_csv(examples)
    label_contents = ""
    for lable in unique_label_array:
        print("Generating Index for lable=="+str(lable))
        index = class_text_to_int(lable, unique_label_array)
        print("Label=="+str(lable)+", index ="+str(index))
        label_contents += "item {\n    id : "+str(index)+"\n    name : "+str(lable)+"\n}\n"
```

visualize_dataset.py

```
import time
from absl import app, flags, logging
from absl.flags import FLAGS
import numpy as np
import tensorflow as tf
```

```
from models import (
    YoloV3, YoloV3Tiny
)
from dataset import load_tfrecord_dataset, transform_images
from utils import draw_outputs

flags.DEFINE_string('classes', './data/coco.names', 'path to classes file')
flags.DEFINE_integer('size', 416, 'resize images to')
flags.DEFINE_string(
    'dataset', './data/voc2012_train.tfrecord', 'path to dataset')
flags.DEFINE_string('output', './output.jpg', 'path to output image')


def main(_argv):
    class_names = [c.strip() for c in open(FLAGS.classes).readlines()]
    logging.info('classes loaded')

    dataset = load_tfrecord_dataset(FLAGS.dataset, FLAGS.classes, FLAGS.size)
    dataset = dataset.shuffle(512)

    for image, labels in dataset.take(1):
        boxes = []
        scores = []
        classes = []
        for x1, y1, x2, y2, label in labels:
            if x1 == 0 and x2 == 0:
                continue

            boxes.append((x1, y1, x2, y2))
            scores.append(1)
            classes.append(label)
        nums = [len(boxes)]
```

```
        boxes = [boxes]
        scores = [scores]
        classes = [classes]

        logging.info('labels:')
        for i in range(nums[0]):
            logging.info('\t{}, {}, {}'.format(class_names[int(classes[0][i])],
                                               np.array(scores[0][i]),
                                               np.array(boxes[0][i])))

        img = cv2.cvtColor(image.numpy(), cv2.COLOR_RGB2BGR)
        img = draw_outputs(img, (boxes, scores, classes, nums), class_names)
        cv2.imwrite(FLAGS.output, img)
        logging.info('output saved to: {}'.format(FLAGS.output))


if __name__ == '__main__':
    app.run(main)
```

train.py

```
from absl import app, flags, logging
from absl.flags import FLAGS

import tensorflow as tf
import numpy as np
from tensorflow.keras.callbacks import (
    ReduceLROnPlateau,
    EarlyStopping,
    ModelCheckpoint,
    TensorBoard
)
```

```
from models import (
    YoloV3, YoloV3Tiny, YoloLoss,
    yolo_anchors, yolo_anchor_masks,
    yolo_tiny_anchors, yolo_tiny_anchor_masks
)
from utils import freeze_all
import dataset as dataset

flags.DEFINE_string('dataset', '', 'path to dataset')
flags.DEFINE_string('val_dataset', '', 'path to validation dataset')
flags.DEFINE_boolean('tiny', False, 'yolov3 or yolov3-tiny')
flags.DEFINE_string('weights', './checkpoints/yolov3.tf',
                    'path to weights file')
flags.DEFINE_string('classes', './data/coco.names', 'path to classes file')
flags.DEFINE_enum('mode', 'fit', ['fit', 'eager_fit', 'eager_tf'],
                  'fit: model.fit, '
                  'eager_fit: model.fit(run_eagerly=True), '
                  'eager_tf: custom GradientTape')
flags.DEFINE_enum('transfer', 'none',
                  ['none', 'darknet', 'no_output', 'frozen', 'fine_tune'],
                  'none: Training from scratch, '
                  'darknet: Transfer darknet, '
                  'no_output: Transfer all but output, '
                  'frozen: Transfer and freeze all, '
                  'fine_tune: Transfer all and freeze darknet only')
flags.DEFINE_integer('size', 416, 'image size')
flags.DEFINE_integer('epochs', 2, 'number of epochs')
flags.DEFINE_integer('batch_size', 8, 'batch size')
flags.DEFINE_float('learning_rate', 1e-3, 'learning rate')
flags.DEFINE_integer('num_classes', 80, 'number of classes in the model')
flags.DEFINE_integer('weights_num_classes', None, 'specify num class for `weights` file if different, '
                     'useful in transfer learning with different number of classes')
```

```
def main(_argv):
    physical_devices = tf.config.experimental.list_physical_devices('GPU')
    if len(physical_devices) > 0:
        tf.config.experimental.set_memory_growth(physical_devices[0], True)

    if FLAGS.tiny:
        model = YoloV3Tiny(FLAGS.size, training=True,
                           classes=FLAGS.num_classes)
        anchors = yolo_tiny_anchors
        anchor_masks = yolo_tiny_anchor_masks
    else:
        model = YoloV3(FLAGS.size, training=True, classes=FLAGS.num_classes)
        anchors = yolo_anchors
        anchor_masks = yolo_anchor_masks

    if FLAGS.dataset:
        train_dataset = dataset.load_tfrecord_dataset(
            FLAGS.dataset, FLAGS.classes, FLAGS.size)
    else:
        assert False, "You need to load a Training dataset"
    train_dataset = train_dataset.shuffle(buffer_size=512)
    train_dataset = train_dataset.batch(FLAGS.batch_size)
    train_dataset = train_dataset.map(lambda x, y: (
        dataset.transform_images(x, FLAGS.size),
        dataset.transform_targets(y, anchors, anchor_masks, FLAGS.size)))
    train_dataset = train_dataset.prefetch(
        buffer_size=tf.data.experimental.AUTOTUNE)

    if FLAGS.val_dataset:
```

```
    val_dataset = dataset.load_tfrecord_dataset(
        FLAGS.val_dataset, FLAGS.classes, FLAGS.size)
else:
    assert False, "You need to load a Validation dataset"

val_dataset = val_dataset.batch(FLAGS.batch_size)
val_dataset = val_dataset.map(lambda x, y: (
    dataset.transform_images(x, FLAGS.size),
    dataset.transform_targets(y, anchors, anchor_masks, FLAGS.size)))

if FLAGS.transfer == 'none':
    pass
elif FLAGS.transfer in ['darknet', 'no_output']:
    if FLAGS.tiny:
        model_pretrained = YoloV3Tiny(
            FLAGS.size, training=True, classes=FLAGS.weights_num_classes or FLAGS.num_classes)
    else:
        model_pretrained = YoloV3(
            FLAGS.size, training=True, classes=FLAGS.weights_num_classes or FLAGS.num_classes)
    model_pretrained.load_weights(FLAGS.weights)

    if FLAGS.transfer == 'darknet':
        model.get_layer('yolo_darknet').set_weights(
            model_pretrained.get_layer('yolo_darknet').get_weights())
        freeze_all(model.get_layer('yolo_darknet'))

    elif FLAGS.transfer == 'no_output':
        for l in model.layers:
            if not l.name.startswith('yolo_output'):
                l.set_weights(model_pretrained.get_layer(
                    l.name).get_weights())
                freeze_all(l)
```

```
else:
    model.load_weights(FLAGS.weights)
    if FLAGS.transfer == 'fine_tune'
        darknet = model.get_layer('yolo_darknet')
        freeze_all(darknet)
    elif FLAGS.transfer == 'frozen':
        freeze_all(model)

optimizer = tf.keras.optimizers.Adam(lr=FLAGS.learning_rate)
loss = [YoloLoss(anchors[mask], classes=FLAGS.num_classes)
        for mask in anchor_masks]

if FLAGS.mode == 'eager_tf':
    avg_loss = tf.keras.metrics.Mean('loss', dtype=tf.float32)
    avg_val_loss = tf.keras.metrics.Mean('val_loss', dtype=tf.float32)

    for epoch in range(1, FLAGS.epochs + 1):
        for batch, (images, labels) in enumerate(train_dataset):
            with tf.GradientTape() as tape:
                outputs = model(images, training=True)
                regularization_loss = tf.reduce_sum(model.losses)
                pred_loss = []
                for output, label, loss_fn in zip(outputs, labels, loss):
                    pred_loss.append(loss_fn(label, output))
                total_loss = tf.reduce_sum(pred_loss) + regularization_loss

            grads = tape.gradient(total_loss, model.trainable_variables)
            optimizer.apply_gradients(
                zip(grads, model.trainable_variables))
```

```
            logging.info("{}_train_{}, {}, {}".format(
                epoch, batch, total_loss.numpy(),
                list(map(lambda x: np.sum(x.numpy()), pred_loss))))
            avg_loss.update_state(total_loss)

        for batch, (images, labels) in enumerate(val_dataset):
            outputs = model(images)
            regularization_loss = tf.reduce_sum(model.losses)
            pred_loss = []
            for output, label, loss_fn in zip(outputs, labels, loss):
                pred_loss.append(loss_fn(label, output))
            total_loss = tf.reduce_sum(pred_loss) + regularization_loss

            logging.info("{}_val_{}, {}, {}".format(
                epoch, batch, total_loss.numpy(),
                list(map(lambda x: np.sum(x.numpy()), pred_loss))))
            avg_val_loss.update_state(total_loss)

        logging.info("{}, train: {}, val: {}".format(
            epoch,
            avg_loss.result().numpy(),
            avg_val_loss.result().numpy()))

        avg_loss.reset_states()
        avg_val_loss.reset_states()
        model.save_weights(
            'checkpoints/yolov3_train_{}.tf'.format(epoch))
    else:
        model.compile(optimizer=optimizer, loss=loss,
                      run_eagerly=(FLAGS.mode == 'eager_fit'))

        callbacks = [
```

```
        ReduceLROnPlateau(verbose=1),
        EarlyStopping(patience=3, verbose=1),
        ModelCheckpoint('checkpoints/yolov3_train_{epoch}.tf',
                        verbose=1, save_weights_only=True),
        TensorBoard(log_dir='logs')
    ]

    history = model.fit(train_dataset,
                        epochs=FLAGS.epochs,
                        callbacks=callbacks,
                        validation_data=val_dataset)


if __name__ == '__main__':
  try:
    app.run(main)
  except SystemExit:
    pass
```

search_for_mira_robot.py

```
import rospy
from sensor_msgs.msg import Image
from std_msgs.msg import String

import time
from absl import app, flags, logging
from absl.flags import FLAGS
import numpy as np
import tensorflow as tf
from models import (
  YoloV3, YoloV3Tiny
)
```

```
from dataset import transform_images, load_tfrecord_dataset
from utils import draw_outputs



flags.DEFINE_string('classes', './data/new_names.names', 'path to classes file')
flags.DEFINE_string('weights', './checkpoints/yolov3_train_20.tf',
                    'path to weights file')
flags.DEFINE_boolean('tiny', False, 'yolov3 or yolov3-tiny')
flags.DEFINE_integer('size', 416, 'resize images to')
flags.DEFINE_string('output', './data/output.jpg', 'path to output image')
flags.DEFINE_integer('num_classes', 1, 'number of classes in the model')

class TensorFlowImageRecognition(object):

    def __init__(self):
        physical_devices = tf.config.experimental.list_physical_devices('GPU')
        if len(physical_devices) > 0:
            tf.config.experimental.set_memory_growth(physical_devices[0], True)



        if FLAGS.tiny:
            self.yolo = YoloV3Tiny(classes=FLAGS.num_classes)
        else:
            self.yolo = YoloV3(classes=FLAGS.num_classes)

        self.yolo.load_weights(FLAGS.weights).expect_partial()
        logging.info('weights loaded')

        self.class_names = [c.strip() for c in open(FLAGS.classes).readlines()]
        logging.info('classes loaded')

    def detect_objects_from_image(self, img_raw, save_detection=False):
```

```
        img = tf.expand_dims(img_raw, 0)
        img = transform_images(img, FLAGS.size)

        t1 = time.time()
        boxes, scores, classes, nums = self.yolo(img)
        t2 = time.time()
        logging.info('time: {}'.format(t2 - t1))

        logging.info('detections:')
        objects_detected_list = []
        for i in range(nums[0]):
            logging.info('\t{}, {}, {}'.format(self.class_names[int(classes[0][i])],
                                               np.array(scores[0][i]),
                                               np.array(boxes[0][i])))

            objects_detected_list.append(self.class_names[int(classes[0][i])])


        rospy.logdebug("Result-Detection="+str(objects_detected_list))
        img = cv2.cvtColor(img_raw, cv2.COLOR_BGR2RGB)
        img_detection = draw_outputs(img, (boxes, scores, classes, nums), self.class_names)
        if save_detection:
            cv2.imwrite(FLAGS.output, img)
            logging.info('output saved to: {}'.format(FLAGS.output))

        return img_detection, objects_detected_list


class RosTensorFlow(object):
    def __init__(self, save_detections=False, image_rostopic=True):
```

```
        self._save_detections = save_detections
        self._process_this_frame = True
        self._image_rostopic = image_rostopic
        self._cv_bridge = CvBridge()
        self.tensorflow_object = TensorFlowIageRecognition()

        self._camera_image_topic = '/mira/mira/camera1/image_raw'
        self.check_image_topic_ready()
        self._sub = rospy.Subscriber(self._camera_image_topic, Image, self.callback, queue_size=1)
        self._result_pub = rospy.Publisher('result', String, queue_size=1)

        self.image_detection = rospy.Publisher('/image_detection', Image, queue_size=10)




    def check_image_topic_ready(self):
        camera_image_data = None
        while camera_image_data is None and not rospy.is_shutdown():
            try:
                camera_image_data = rospy.wait_for_message(self._camera_image_topic, Image,
timeout=1.0)
                rospy.loginfo("Current "+str(self._camera_image_topic)+" READY=>")
            except Exception as ex:
                print(ex)
                rospy.logerr("Current "+str(self._camera_image_topic)+" not ready yet, retrying...")
        rospy.loginfo("Camera Sensor READY")

    def publish_results_objecets_list(self, objects_detected_list):
        result_msg = String()
        for object_name in objects_detected_list:
            result_msg.data = object_name
```

```
        self._result_pub.publish(result_msg)


    def callback(self, image_msg):
        if (self._process_this_frame):
            rospy.logdebug("Image processing....")
            image_np = self._cv_bridge.imgmsg_to_cv2(image_msg, "bgr8")

            img_detection, objects_detected_list =
self.tensorflow_object.detect_objects_from_image(img_raw=image_np,
                                                 save_detection=self._save_detections)

            self.publish_results_objecets_list(objects_detected_list)

            if self._image_rostopic:
                self.image_detection.publish(self._cv_bridge.cv2_to_imgmsg(img_detection, "bgr8"))
            else:
                cv2.imshow("Image window", img_detection)
                cv2.waitKey(1)
            rospy.logdebug("Image processing....DONE")
        else:
            pass
        self._process_this_frame = not self._process_this_frame


    def main(self):
        rospy.spin()


def main_action(_argv):
    rospy.init_node('search_mira_robot_node', log_level=rospy.DEBUG)
    tensor = RosTensorFlow()
    tensor.main()


if __name__ == '__main__':
```

```
try:
    app.run(main_action)
except SystemExit:
    pass
```