GRADO EN INGENIERÍA DE ROBÓTICA SOFTWARE

Escuela Técnica Superior de Ingeniería de Telecomunicación

Curso académico 2021-2022

Trabajo fin de grado

Escribe el título del trabajo aquí
con la segunda línea aquí

Tutor: Julio Vega Pérez
Autor: Álvaro Mariscal Ávila

Documento de **Álvaro Mariscal Ávila**.

Agradecimientos

Unas bonitas palabras...

Quizás un segundo párrafo esté bien. No te olvides de nadie.

Un tercero tampoco viene mal para contar alguna anécdota...

¿Alguien más? Aunque sean *actores* secundarios.

Un quinto párrafo como colofón.

A alguien especial;
si no, tampoco pasa nada

Madrid, xx de xxxxxx de 20xx
Tu nombre

Resumen

Escribe aquí el resumen del trabajo. Un primer párrafo para dar contexto sobre la temática que rodea al trabajo.

Un segundo párrafo concretando el contexto del problema abordado.

En el tercer párrafo, comenta cómo has resuelto la problemática descrita en el anterior párrafo.

Por último, en este cuarto párrafo, describe cómo han ido los experimentos.

Acrónimos

AERO *Autonomous Exploration Rover*

AI *Artificial Intelligence*

ANN *Artificial Neural Network*

API *Application Programming Interface*

HRI *Human-Robot Interaction*

AGV *Automated guided vehicle*

AMR *Autonomous mobile robot*

Índice general

Índice de figuras

Listado de códigos

Listado de ecuaciones

Índice de cuadros

Capítulo 1
Introducción

Quizás algún fragmento de libro inspirador...

Autor, *Título*

En la actualidad los vehículos autónomos están en auge, cada vez tenemos más ejemplos de tareas típicamente realizadas por humanos que ya, es posible realizar sin un humano al mando. La conducción autónoma tiene el potencial para cambiar la forma en que nos movemos, aportando seguridad y comfort, si bien es cierto que aún no vemos vehículos circulando sin conductor, mucha de la tecnología necesaria para hacerlo ya está presente en los vehículos actuales. La conducción autónoma va mucho más allá de, únicamente los vehículos que transitan las ciudades, es aplicable a muchos otros ámbitos, por ejemplo entornos industriales, de inspección o incluso de exploración donde el vehículo se enfrenta a situaciones impredecibles y ante las que debe saber reaccionar correctamente.

1.1. Coches autónomos

Cuando se nos viene a la cabeza el concepto de vehículo autónomo, se suele relacionar con coches autónomos, es decir, los vehículos que circulan a diario por las ciudades; coches, autobuses, furgonetas pero sin una persona al volante. Todavía no está presente en las ciudades pero en un corto plazo de tiempo lo estará, el principal inconveniente actual es la regulación, a diferencia de, por ejemplo, el entorno de la aviación, donde desde hace décadas el control de la aeronave es autómatico a excepción de tareas como el despegue, el aterrizaje o situaciones de emergencia.

Actualmente coches de última generación como *Tesla*, ya incorporan un grado de autonomía elevado en determinadas situaciones, pero siempre con un conductor al

volante que debe permanecer atento para poder reaccionar.

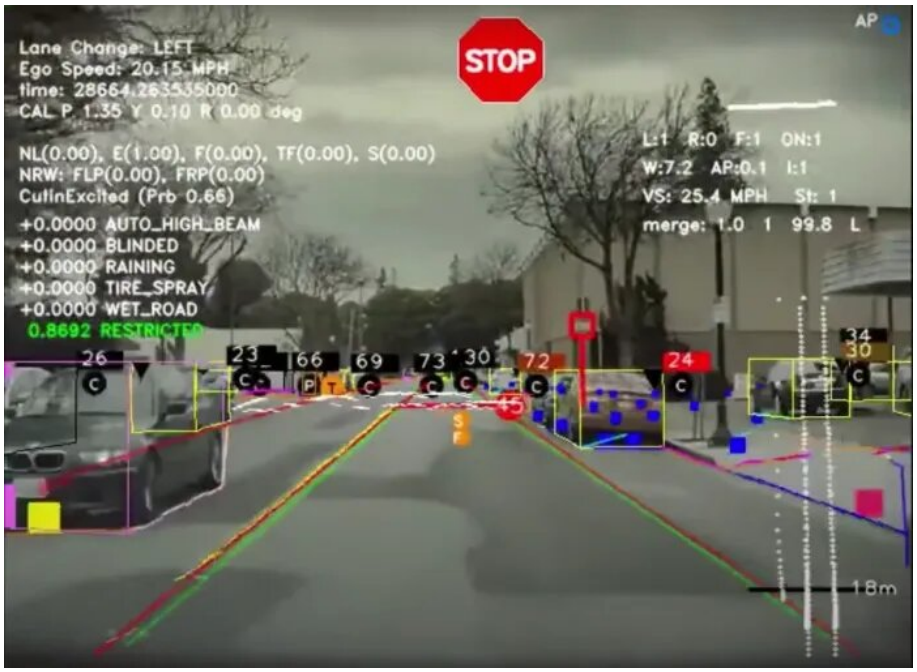

Figura 1.1: *Tesla AutoPilot.*

Atendiendo al estándar *SAE J3016* los niveles de autonomía se pueden dividir en cinco:

1. Sin automatización: avisos y asistencia puntualmente
2. Asistencia a la conducción: centrado de carril **o** control de crucero
3. Automatización parcial: centrado de carril **y** control de crucero
4. Automatización condicionada: conducción automática en atascos
5. Automatización elevada: conducción automática en algunas situaciones
6. Automatización completa: conducción automática en cualquier situación

[SAE, 2018]

1.2. AMRs

Los *AMRs* son robots móviles autónomos, capaces de navegar por entornos dinámicos, conviviendo con humanos a su alrededor y sabiendo sobreponerse a

situaciones para las que no habían sido programados explícitamente. Son los sucesores de los *AGVs*, vehículos guiados automatizados, este tipo de vehículos requieren una cierta infraestructura dependiendo del tipo de guiado, ya sea filoguiados, a través de pintura o a través de cualquier otra técnica que haga que ese vehículo sólo pueda funcionar cuando se sabe la infraestructura previa que estará presente en el entorno de trabajo.

Además, presentan muchas dificultades para relacionarse con obstáculos o humanos, donde ante un cambio pequeño del entorno, el robot se detendrá por seguridad. A diferencia de estos, los *AMRs*, son capaces de realizar multitud de tareas en entornos donde la infraestructura necesaria es casi nula, quizá tengan una serie de requisitos en cuanto a conectividad, pero con esa excepción, son robots que pueden ser diseñados para navegar por cualquier tipo de ambiente.

Un ejemplo muy representativo de *AMRs*, es *Kiva Systems*, empresa comprada por *Amazon* para automatizar sus almacenes en tareas de logística a nivel interno, maximizando la productividad y el almacenamiento, tanto en profundidad como en altura y minimizando el coste en personal.

Figura 1.2: *AMRs Kiva Systems* en almacenes de *Amazon*.

1.3. Visión como sensor principal

Gracias a un sensor como la cámara podemos obtener una información muy completa del entorno que rodea al robot y con ello poder actuar en consecuencia. La cámara se presenta como el sensor más interesante que puede equipar un robot, cuentan con un tamaño y peso muy reducido, como es el caso de la 1.3, además de un coste muy

reducido. Sin embargo, presenta algunas dificultades cuando nos disponemos a tratar la imagen recibida; en cuanto a potencia del dispositivo, para procesar una imagen, y dependiendo de su resolución, es necesario un mínimo de requerimientos técnicos a nivel de *hardware*, para poder conseguir un procesamiento con un nivel adecuado de *fps*, por otra parte, la imagen recibida deberá haber sido captada en un entorno con buenas condiciones lumínicas. Para ello existen diferentes tipos de sensores *EO/IR*, como por ejemplo, cámaras de visión noctura, donde el espectro utilizado es *FIR*, con ello se consigue resaltar en la imagen lo que realmente es necesario.

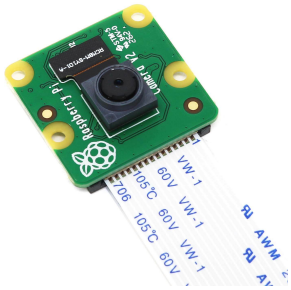

Figura 1.3: *PiCamera* usada en la placa *Raspberry Pi*.

Un ejemplo es *BMW's FIR-based Autoliv Night Vision System* [Raiciu, 2009], trabaja con imagenes de *320x240* y cuenta con un rango de *300 metros*.

Figura 1.4: *BMW's FIR-based Autoliv Night Vision System*.

Cuando se realiza el procesamiento de una imagen, es interesante conocer la distancia, por ejemplo, a la que se encuentra un objeto en concreto, para ello existen

soluciones para poder conocer dicha información con una **única** cámara tradicional, es el caso de *cite Open Vision System for Low-Cost Robotics Education* o [Dal, 2021], sin embargo, estos algoritmos se basan en la suposición de que todos los objetos se encuentras situados en un plano imaginario situado en el suelo. Para eliminar esta restricción, y poder conocer distancias de objetos que no se encuentran situados en el suelo, surgen las *cámaras RGBD*, en las que cada píxel de la imagen proporciona, además del color RGB, una tercera componente llamada *depth*. Una de las primeras *cámaras RGBD* comerciales es la *Kinect* desarrollada por *Microsoft* para su consola *Xbox 360*.

Figura 1.5: Imagen de profundidad *Kinect*.

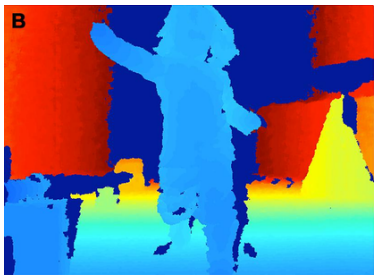

Figura 1.6: *Kinect* desarrollada por *Microsoft*.

La visión artificial ofrece multitud de posibilidades, no solo en robótica, tiene una gran de aplicaciones en campos tan dispares como la medicina, la realidad aumentada, el procesamiento de señales o la agricultura.

1.4. Arquitectura

La decisión de qué arquitectura utilizar en un robot es determinante. Es necesario evaluar multitud de factores; consumo de energía, potencia requerida, tamaño y peso, sistema operativo a utilizar, posibilidad de recibir respuesta en *real time*, precio etc...

Existen multitud de arquitecturas utilizadas en proyectos robóticos *x86*, *x86_64*, *ARMv6*, *ARMv7* o *AArch64*. Pero, actualmente hay dos arquitecturas predominantes *x86_64* y *AArch64*, ambas de *64 bits*, ya que los 32 bits han quedado desfasados para la gran mayoría de aplicaciones, además, los nuevos sistemas operativos comienzan a no soportarlos. [J.Pomeyrol, 2019]

Estas dos arquitecturas tienen grandes diferencias en cuanto al diseño de los procesadores y las instrucciones que utilizan:

- Reduced Instruction Set Computer (*RISC*)
- Complex Instruction Set Computer (*CISC*)

El conjunto de instrucciones utilizado por *AArch64* es *CISC*, este conjunto se compone de gran cantidad de instrucciones y muchas de ellas complejas para realizar tareas que, el conjunto *RISC*, puede realizar con varias instrucciones. Este último conjunto es utilizado por la arquitectura *x86_64*.

Ejemplos representativos de *AArch64*:

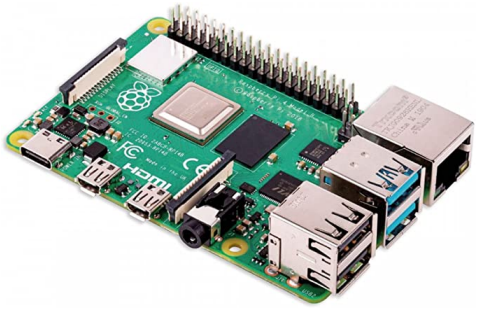

Figura 1.7: *Raspberry Pi 4*.

Figura 1.8: *Jetson Nano*.

Ejemplos representativos de *x86_64*:

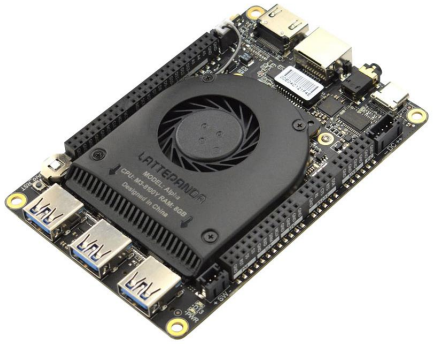

Figura 1.9: *LattePanda Alpha 864s*.

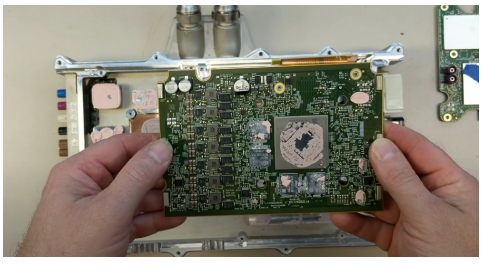

Figura 1.10: *AMD Zen* como CPU y *Navi 23* como GPU, usado en Tesla Model S [Ros, 2021].

1.5. Deep Learning

El *Deep Learning* se basa en redes neuronales que parten del *Machine Learning*, que a su vez surge de la *Inteligencia Artificial* (*IA*). Sus inicios se remontan al año 1979 cuando *Kunihiko Fukushima* desarrolló una red neuronal de entre 5 y 6 capas llamada *neocognitrón*[Fukushima, 1979], con el objetivo de reconocer carácteres japoneses.

Este tipo de redes neuronales tiene multitud de aplicaciones, pero todas comparten grandes cantidades de datos, conocidos como *datasets*, en cualquier formato; vídeo, imagen, sonido. Algunas de ellas son: clasificación de objetos, procesamiento natural del lenguaje, *Big Data*, anaĺisis médico, conversión de imágenes en blanco y negro a color etc...

1.5.1. YOLO

Uno de los algoritos más populares, capaz de detectar y clasificar objetos provinientes de una imagen es *YOLO* (*You Only Look Once*). Sus principales ventajas son una gran precisión y la posibilidad, con el hardware adecuado, de ejecutar en tiempo real. Este algoritmo hace honor a su nombre y, por tanto, solo realiza una *propagación hacia delante* en cada ejecución.

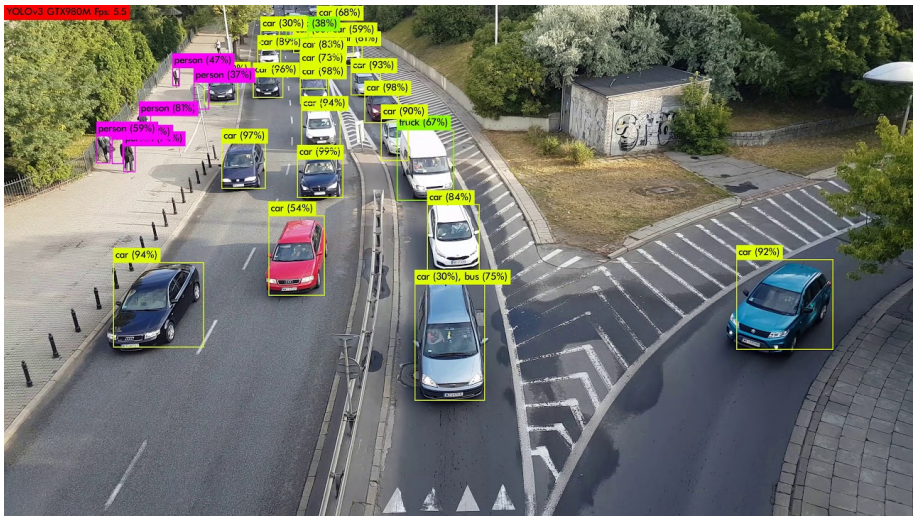

Figura 1.11: Objetos detectados por *YOLO* en una carretera.

Se basa en el uso de redes nueronales convolucionales, *Convolutional neural network*.

Se diferencia de una red neuronal tradicional, en que la operación de multiplición de matrices, se sustituye por una operación matemática llamada convolución, consistente en mezclar dos fuentes de información para producir una tercera, en este caso dos funciones.

YOLO hace uso, esencialmente, de tres técnicas para conseguir reconcer objetos:

- División de la imagen en celdas: de esta forma se pueden detectar multitud de objetos en una imagen
- Creación de *bounding boxes*: cajas 1.11 dibujando el contorno del objeto detectado y fijando la probabilidad de que el objeto detectado sea correcto
- Intersección sobre la únion: consiste en seleccionar el *bounding box* con mayor probabilidad cuando hay varios superpuestos

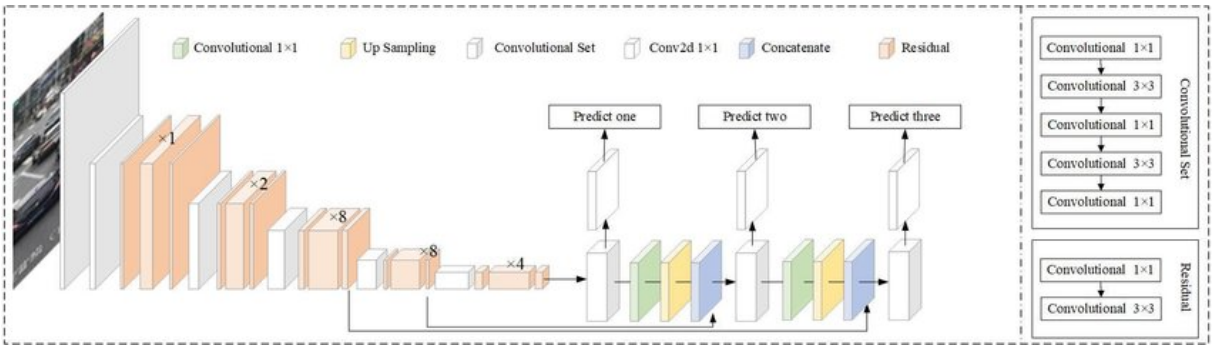

Figura 1.12: Arquitectura *YOLOv3* con 53 capas convolucionales.

Uno de los principales problemas de *YOLO* es la baja de probabilidad de detectar objetos de tamaño reducido, debido principalmente a la baja resolución de la red (está permitido incrementarla pero disminuye su rendimiento) que hace muy difícil su detección.

Además, existen versiones reducidas de este algoritmo como, Tiny-YOLO y Fast YOLO capaces de ser ejecutados en equipos de bajo coste y reducido tamaño.

En los siguientes capítulos trataremos los objetivos a cumplir ...
Solucionar números páginas

Capítulo 2
Objetivos

Quizás algún fragmento de libro inspirador...

Autor, *Título*

Escribe aquí un párrafo explicando brevemente lo que vas a contar en este capítulo. En este capítulo lo ideal es explicar cuáles han sido los objetivos que te has fijado conseguir con tu trabajo, qué requisitos ha de respetar el resultado final, y cómo lo has llevado a cabo; esto es, cuál ha sido tu plan de trabajo.

2.1. Descripción del problema

Cuenta aquí el objetivo u objetivos generales y, a continuación, concrétalos mediante objetivos específicos.

2.2. Requisitos

Describe los requisitos que ha de cumplir tu trabajo.

2.3. Metodología

Qué paradigma de desarrollo software has seguido para alcanzar tus objetivos.

2.4. Plan de trabajo

Qué agenda has seguido. Si has ido manteniendo reuniones semanales, cumplimentando objetivos parciales, si has ido afinando poco a poco un producto final completo, etc.

Capítulo 3
Plataforma de desarrollo

Quizás algún fragmento de libro inspirador...

Autor, *Título*

Escribe aquí un párrafo explicando brevemente lo que vas a contar en este capítulo. En este capítulo, explica qué has usado a nivel hardware y software para poder desarrollar tu trabajo: librerías, sistemas operativos, plataformas, entornos de desarrollo, etc.

Capítulo 4
Diseño

Quizás algún fragmento de libro inspirador...

Autor, *Título*

Escribe aquí un párrafo explicando brevemente lo que vas a contar en este capítulo. En este capítulo (y quizás alguno más) es donde, por fin, describes detalladamente qué has hecho y qué experimentos has llevado a cabo para validar tus desarrollos.

4.1. Snippets

Puede resultar interesante, para clarificar la descripción, mostrar fragmentos de código (o *snippets*) ilustrativos. En el Código 4.1 vemos un ejemplo escrito en `C++`.

```
void Memory::hypothesizeParallelograms () {
  for(it1 = this->controller->segmentMemory.begin(); it1++) {
    squareFound = false; it2 = it1; it2++;
    while ((it2 != this->controller->segmentMemory.end()) && (!squareFound))
        {
      if (geometry::haveACommonVertex((*it1),(*it2),&square)) {
        dist1 = geometry::distanceBetweenPoints3D ((*it1).start, (*it1).end);
        dist2 = geometry::distanceBetweenPoints3D ((*it2).start, (*it2).end);
      }
    // [...]
```

Código 4.1: Función para buscar elementos 3D en la imagen

En el Código 4.2 vemos un ejemplo escrito en `Python`.

4.2. Verbatim

Para mencionar identificadores usados en el código —como nombres de funciones o variables— en el texto, usa el entorno literal o verbatim

```
def mostrarValores():
    print (w1.get(), w2.get())

master = Tk()
w1 = Scale(master, from_=0, to=42)
w1.pack()
w2 = Scale(master, from_=0, to=200, orient=HORIZONTAL)
w2.pack()
Button(master, text='Show', command=mostrarValores).pack()

mainloop()
```

Código 4.2: Cómo usar un Slider

`hypothesizeParallelograms()`. También se puede usar este entorno para varias líneas, como se ve a continuación:

```
void Memory::hypothesizeParallelograms () {
  // add your code here
}
```

4.3. Ecuaciones

Si necesitas insertar alguna ecuación, puedes hacerlo. Al igual que las figuras, no te olvides de referenciarlas. A continuación se exponen algunas ecuaciones de ejemplo: Ecuación 4.1 y Ecuación 4.2.

$$H = 1 - \frac{\sum_{i=0}^{N} \frac{(\frac{d_{js}+d_{je}}{2})}{N}}{M} \tag{4.1}$$

Ecuación 4.1: Ejemplo de ecuación con fracciones

$$v(entrada) = \begin{cases} 0 & \text{if} \quad \epsilon_t < 0{,}1 \\ K_p \cdot (T_t - T) & \text{if} \quad 0{,}1 \leq \epsilon_t < M_t \\ K_p \cdot M_t & \text{if} \quad M_t < \epsilon_t \end{cases} \tag{4.2}$$

Ecuación 4.2: Ejemplo de ecuación con array y letras y símbolos especiales

4.4. Tablas o cuadros

Si necesitas insertar una tabla, hazlo dígnamente usando las propias tablas de LaTeX, no usando pantallazos e insertándolas como figuras... En el Cuadro 4.1 vemos

un ejemplo.

Parámetros	**Valores**
Tipo de sensor	Sony IMX219PQ[7] CMOS 8-Mpx
Tamaño del sensor	3.674 x 2.760 mm (1/4"format)
Número de pixels	3280 x 2464 (active pixels)
Tamaño de pixel	1.12 x 1.12 um
Lente	f=3.04 mm, f/2.0
Ángulo de visión	62.2 x 48.8 degrees
Lente SLR equivalente	29 mm

Cuadro 4.1: Parámetros intrínsecos de la cámara

Capítulo 5
Conclusiones

Quizás algún fragmento de libro inspirador...

Autor, *Título*

Escribe aquí un párrafo explicando brevemente lo que vas a contar en este capítulo, que básicamente será una recapitulación de los problemas que has abordado, las soluciones que has prouesto, así como los experimentos llevados a cabo para validarlos. Y con esto, cierras la memoria.

5.1. Conclusiones

Enumera los objetivos y cómo los has cumplido.

Enumera también los requisitos implícitos en la consecución de esos objetivos, y cómo se han satisfecho.

No olvides dedicar un par de párrafos para hacer un balance global de qué has conseguido, y por qué es un avance respecto a lo que tenías inicialmente. Haz mención expresa de alguna limitación o peculiaridad de tu sistema y por qué es así. Y también, qué has aprendido desarrollando este trabajo.

Por último, añade otro par de párrafos de líneas futuras; esto es, cómo se puede continuar tu trabajo para abarcar una solución más amplia, o qué otras ramas de la investigación podrían seguirse partiendo de este trabajo, o cómo se podría mejorar para conseguir una aplicación real de este desarrollo (si es que no se ha llegado a conseguir).

5.2. Corrector ortográfico

Una vez tengas todo, no olvides pasar el corrector ortográfico de LaTeXa todos tus ficheros *.tex*. En `Windows`, el propio editor `TeXworks` incluye el corrector. En `Linux`, usa `aspell` ejecutando el siguiente comando en tu terminal:

```
aspell --lang=es --mode=tex check capitulo1.tex
```

Bibliografía

[Dal, 2021] Dal, A. (2021). Distance(webcam) estimation with single-camera opencv-python.

[Fukushima, 1979] Fukushima, K. (1979). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *NHK Broadcasting Science Research Laboratories*.

[J.Pomeyrol, 2019] J.Pomeyrol (2019). Canonical concreta el soporte de 32-bit para ubuntu 20.04 lts. Muy Linux.

[Raiciu, 2009] Raiciu, T. (2009). How night vision works. *autoevolution.com*.

[Ros, 2021] Ros, I. (2021). Tesla monta cpus zen de amd y gpus rdna 2 en sus coches.

[SAE, 2018] SAE, I. (2018). Sae j3016 levels of driving automation. *SAE J3016*.

[Vega, 2018a] Vega, J. (2018a). *Educational framework using robots with vision for constructivist teaching Robotics to pre-university students*. Doctoral thesis on computer science and artificial intelligence, University of Alicante.

[Vega, 2018b] Vega, J. (2018b). JdeRobot-Kids framework for teaching robotics and vision algorithms. In *II jornada de investigación doctoral*. University of Alicante.

[Vega, 2019] Vega, J. (2019). El profesor Julio Vega, finalista del concurso 'Ciencia en Acción 2019'. URJC, on-line newspaper interview.

[Vega and Cañas, 2019] Vega, J. and Cañas, J. (2019). PyBoKids: An innovative python-based educational framework using real and simulated Arduino robots. *Electronics*, 8:899–915.

[Vega et al., 2012] Vega, J., Perdices, E., and Cañas, J. (2012). *Attentive visual memory for robot localization*, pages 408–438. IGI Global, USA. Text not available. This book is protected by copyright.