

TRABAJO FIN DE MÁSTER

ESTO ES UNA PRUEBA
Desarrollo de un entorno de
trabajo aplicado al desarrollo de
firmware para robots móviles

Realizado por:

Rubén Garrido Alonso

Máster Universitario en Ingeniería Electrónica

Dirigido por:

Vicent Girbés Juan
Valero Laparra Pérez-Muelas

Realizado en el departamento de
Ingeniería Electrónica

Julio, 2022

Agradecimientos

Quiero agradecer a X por...

También quiero agradecer a Y por...

Resumen

Incluya aquí un resumen de los aspectos generales de su trabajo, en español.

Palabras clave: Palabra clave 1, palabra clave 2, ..., palabra clave N

Abstract

This section should contain an English version of the Spanish abstract.

Keywords: Keyword 1, keyword 2, ..., keyword N

Índice general

Índice de figuras

Índice de extractos de código

1. Introducción

1.1. Introducción

Cada vez es mayor el peso que la tecnología esta teniendo en el día a día de la sociedad. La interacción con dispositivos tecnológicos está convirtiéndose poco a poco en algo cotidiano. Son cada vez mas las tareas que pueden ser suplidas por dispositivos electrónicos que ayudan a que tareas del día a día, incluso labores más peligrosas puedan desarrollarse de una forma segura y eficiente.

Uno de los sectores en auge en esta última década es el sector de los sistemas embebidos y la capacidad de procesamiento de estos mismos, es cada vez mayor la capacidad de dotar de inteligencia a dispositivos cada vez con dimensiones mas reducidas pero con una capacidad de rendimiento que crece de forma exponencial.

El proyecto que se describirá a lo largo de la memoria se encarga de dotar de inteligencia artificial un robot con capacidad de teleoperación y reconocimiento del entorno en el cual se esté operando este mismo. Para poder conseguir esto, se ha tenido que poner a punto un robot que sea capaz de moverse y además sea capaz de procesar y reconocer los objetos de su entorno procesando esta información en tiempo real.

Para conseguir esto se ha dotado al robot de visión artificial, la cual recogerá la información necesaria de una cámara incorporada en el robot y que se complementará con otros sensores que ayudaran al dispositivo móvil a poder moverse de una forma autónoma evitando colisiones.

1.2. Motivación del proyecto

La motivación del proyecto surge con el auge del desarrollo de robots móviles junto con el actual desarrollo de dispositivos dotados de inteligencia artificial o "capacidad de aprendizaje" en estas máquinas.

Con el objetivo de hacer mas accesible el proyecto a toda la comunidad se ha tratado de utilizar hardware abierto, así como intentar utilizar plataformas de software open source.

tras recopilar información sobre el funcionamiento de diferentes tipos de robots móviles, vehículos auto-guiados y diversos dispositivos de conducción autónoma, se ha tratado de comprender y estudiar su funcionamiento interno, desde el desarrollo de hardware de cada uno de los robots hasta comprender la estructura de software que cada un de los diferentes dispositivos puede contener. De este modo, una vez estudiadas las diferentes alternativas que se encuentran actualmente accesibles en el

mercado, se ha tratado de proporcionar una solución accesible, que cuente con cierta escalabilidad y que sirva como plataforma de desarrollo para demás desarrolladores.

Para poder desarrollar ciertas soluciones para diferentes tipos de robots, siempre se han de tener en cuenta las diferentes versiones de todas y cada una de las dependencias que el programa a desarrollar puede tener. También es cierto, que hoy en día existen diversas tecnologías que permiten encapsular todas aquellas dependencias en un entorno estable para que todos los desarrolladores de un mismo proyecto cuenten con las mismas herramientas y programas, actualizadas a la misma versión, de esta forma se eliminan problemas de retrocompatibilidad entre diferentes desarrolladores y los prototipos a desarrollar.

1.3. Objetivo del proyecto

El objetivo del proyecto es dotar a los usuarios desarrolladores de un entorno encapsulado con todas las herramientas necesarias para poder llevar a cabo el desarrollo programas y nuevas funcionalidades en el marco del desarrollo de robots móviles.

Para ello, la plataforma de desarrollo debe de ser estable y permitir un esquema multiusuario, donde cada uno de los desarrolladores pueda llevar a cabo sus labores con total comodidad y sin conflictos con las versiones desarrolladas por otros usuarios.

Dado que el objetivo principal es implantar un entorno estable y para usar por diferentes usuarios, debe de ser de fácil acceso y fácil instalación para todos los usuarios.

1.4. Marco teórico

El robot utilizado para el proyecto alberga diferentes bloques de funcionamiento o diferentes secciones que trabajaban de forma paralela para poder conformar el proyecto. Es por ello que se debe tener claro desde el inicio del proyecto la forma en la que se va a establecer la comunicación entre los diferentes nodos del proyecto o las diferentes partes que albergan diferente hardware.

Dado el dispositivo que se quiere desarrollar es un robot, se ha optado por utilizar un sistema operativo de software abierto llamado ROS (Robot Operative System).

Dicho sistema operativo se ejecuta sobre una CPU bajo un sistema Linux basado en una distribución Debian.

El proyecto a desarrollar se divide en dos grandes bloques:

- Alto nivel : Lo forman los dispositivos que albergarán las lógicas de navegación así como la parte mas abstraída del Hardware. Prácticamente toda la inteligencia del robot estará contenida en este bloque. Para poder desarrollar el alto

nivel se han utilizado dos Raspberrys, las cuales ejecutarán la lógica de funcionamiento del robot así como el “cerebro” del robot ejecutando el framework ROS.

En el alto nivel también se encuentran diferentes sensores cuya información necesita ser procesada por una CPU y que son utilizados para poder ejercer una navegación más fina así como poder almacenar un mapa de los diferentes entornos por los que ha estado navegando.

- Bajo nivel : El bajo nivel contiene la lógica necesaria para poder realizar la gestión entre los sensores y actuadores y la lógica que cada uno de estos necesita para su correcto funcionamiento. El bajo nivel esta ligado al hardware y alberga la parte de programación que gestiona de una forma mas directa la mayoría de sensores y de actuadores.

El bajo nivel lo conforma entre otros, el microcontrolador de Arduino utilizado para el proyecto.

Para poder realizar la detección de objetos y personas, se empleará un modelo de inteligencia artificial. Se realizará una comparativa entre los diferentes tipos de modelos y tipologías a la hora de realizar una detección de objetos.

El procesado de toda la información que se lleve a cabo para poder procesar la imagen adquirida por la cámara así como la información que servirá como input al modelo de inteligencia artificial, se procesará en el mismo robot. A esta técnica se le conoce como “edge computing”.

2. Entorno Software

Tal y como se menciona a lo largo de la memoria, el desarrollador debe de tener una estructura y un entorno de trabajo que garantice la correcta funcionalidad de todas las herramientas facilitadas para el desarrollo.

Son diferentes las herramientas que se prepararán a lo largo de la memoria, todas las herramientas, funcionan bajo licencias de software libre, y son completamente gratuitas de utilizar. Dado que el entorno está orientado a el desarrollo de firmware para robot móviles, el entorno se encapsulará para que todo se ejecute sobre el sistema operativo Linux. El motivo por el que se ha elegido Linux como base de desarrollo es por ser el sistema operativo principal para el desarrollo de software, ya que está construido bajo licencias de software libre.

Para el desarrollo de

2.1. Linux

2.2. ROS

2.3. Entorno desarrollo Robot

2.3.1. Puesta a punto Coral TPU

Con el objetivo de poder llevar a cabo la detección de objetos a tiempo real, se ha hecho uso de un acelerador gráfico USB, en concreto un dispositivo TPU (Tensor Procesor Unit) el cual se encargará de poder llevar a cabo el cálculo y procesamiento relacionado con la detección de objetos y la aplicación y procesamiento del modelo matemático.

Para poder llevar a cabo el procesamiento de las imágenes y el cálculo estructural de la red neuronal, se utilizan dispositivos a los que comúnmente se les denomina acelerador gráfico. Estos dispositivos se encargan de soportar el peso computacional correspondiente a los cálculos asociados a los modelos matemáticos. En este caso, el gran parte del peso del procesamiento recae sobre el modelo matemático y su procesamiento en CPU, es por ello, que para obtener unos mejores tiempos de respuesta y poder llevar a cabo un procesamiento en tiempo real y con una latencia baja se utilice una TPU.

Tal y como se puede observar en la figura XX, cada una de los diferentes tipos de procesadores destaca en un ámbito diferente y cada uno tiene sus aplicaciones particulares. En lo que atañe al proyecto, ya que la carga de trabajo con mayor peso

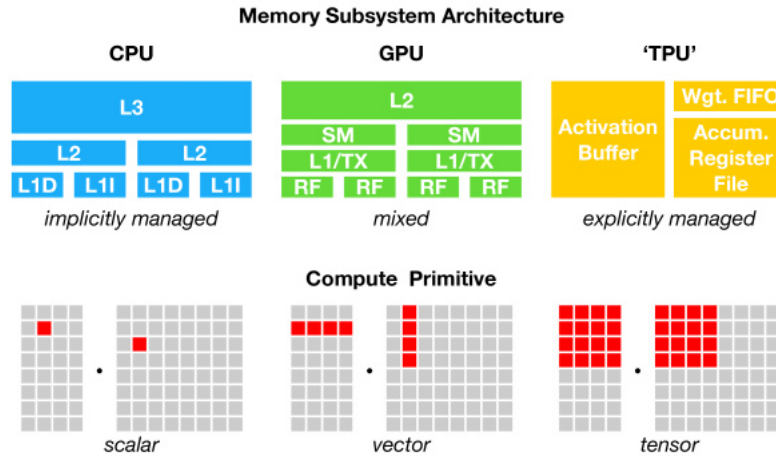


Figura 2.1: Comparativa CPU, GPU y TPU.

recae sobre el modelo de inteligencia artificial, la mejor opción para poder ejecutar todos estos cálculos de la manera más eficiente es utilizar una TPU.

Las TPU son capaces de procesar la información de modelos matemáticos, como por ejemplo las redes neuronales, de 15 a 30 veces mas rápido que una GPU convencional, además de ser mas eficientes energéticamente debido a su bajo consumo. El inconveniente de estos dispositivos es la baja versatilidad a la hora de incluirlos en el proyecto. Para la aplicación deseada encaja perfectamente debido al tipo de procesamiento que realizará, el bajo coste económico y la fácil instalación del dispositivo en el sistema operativo gracias a su conexión USB 3.0.

Para poder poner en funcionamiento la TPU se deben de cumplir una serie de requisitos software para que el sistema operativo sea capaz de trabajar con el hardware. Es necesario instalar una serie de drivers para que se pueda establecer un entendimiento hardware-software y de esa forma poder realizar la mayor parte del procesamiento en este hardware.

Para poner a punto la Coral TPU y poder comprobar su correcto funcionamiento, el primero de los pasos a seguir es instalar un gestor de paquetes para poder resolver todas las dependencias que se vayan necesitando a lo largo de la instalación y puesta a punto. En el caso de los proyectos, se ha instalado conda para poder gestionar entornos e instalar los diferentes paquetes para python que se van necesitando a lo largo de la puesta en marcha.

Se ha de tener en cuenta la distribución y arquitectura que presentará el proyecto a la hora de poder instalar un paquete determinado ya que dependiendo de cada arquitectura, distribución y kernel de cada uno de los sistemas operativos y hardware que se esté utilizando se deberá de adaptar a un software y otro.

Para el caso del proyecto, se está ejecutando todo el código sobre un linux, con una distribución basada en ubuntu server al que se le ha instalado un entorno gráfico. A la hora de instalar el gestor de paquetes Conda hay que tener en cuenta diversos factores diferenciales.



Figura 2.2: Google Coral TPU USB Accelerator.

Primero debemos de conocer la arquitectura y la version del sistema operativo sobre el cual estamos ejecutando, para ello se puede abrir una ventana de comandos o terminal y ejecutar el siguiente comando: `uname -m`.

Tal y como se puede observar en la figura XX, el comando `uname -r` nos muestra en la terminal el tipo de arquitectura sobre el que estamos trabajando. Como era de esperar, Raspberry trabaja con una CPU ARM, por lo cual el resultado de ejecutar el comando es `.arch64`.

Para poder instalar conda sobre procesadores ARM habrá que recurrir a una versión de conda adaptada para estas arquitecturas, ya que los gestores mas utilizados para poder trabajar con conda, bien sea `.Anaconda.` o bien "Miniconda" únicamente están disponibles para poder instalarlos sobre procesadores AMD, por lo cual hay que buscar una alternativa o solución a este pequeño inconveniente.

Tras investigar las diferentes opciones que se encuentran como software abierto por la red, se ha escogido `.Archiconda` como gestor de entornos y paquetes de conda. Archiconda es un software libre y abierto para poder instalar en dispositivos con arquitectura ARM como suelen ser la mayoría de placas de desarrollo para sistemas



Figura 2.3: Gestor de paquetes y entornos Conda.

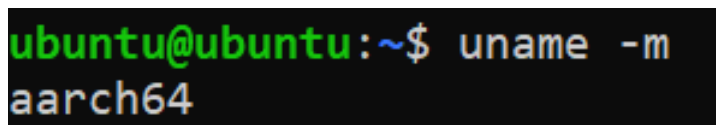


Figura 2.4: Ejecución de comando "uname -r".

embebidos.

Para poder instalar archiconda es necesario descargar un repositorio para posteriormente ejecutar los diferentes comandos necesarios. Con el objetivo de simplificar toda la instalación se ha cogido de un repositorio un bash script para linux que se puede o bien descargar o bien generar desde la propia consola.

Ya que el objetivo es poder dejar toda la información necesaria en un mismo repositorio, se ha decidido generar un fichero bash script propio para poder almacenarlo en un repositorio propio.

Para poder crear el fichero se debe de realizar el siguiente procedimiento:

En primer lugar se debe de generar un archivo en linux con extension .sh, a este fichero se le dará el nombre que el usuario desee, en este caso "installArchiconda.sh".

Una vez creado este fichero, se rellenará con el código o acciones que se deseen ejecutar, en este caso el extracto de código que se desea introducir se puede muestra a continuación.

Una vez el archivo queda generado, se guarda para su posterior ejecución. Para poder dar permiso de ejecución en un sistema linux se utilizará el comando "sudo chmod +x installArchiconda.sh ". De este modo estamos añadiendo permiso de ejecución para el fichero y se podrá ejecutar desde la terminal utilizando "./installArchiconda.sh ". En las siguientes figuras se puede observar el proceso de creación del fichero así como su ejecución.

```
#!/bin/bash
```

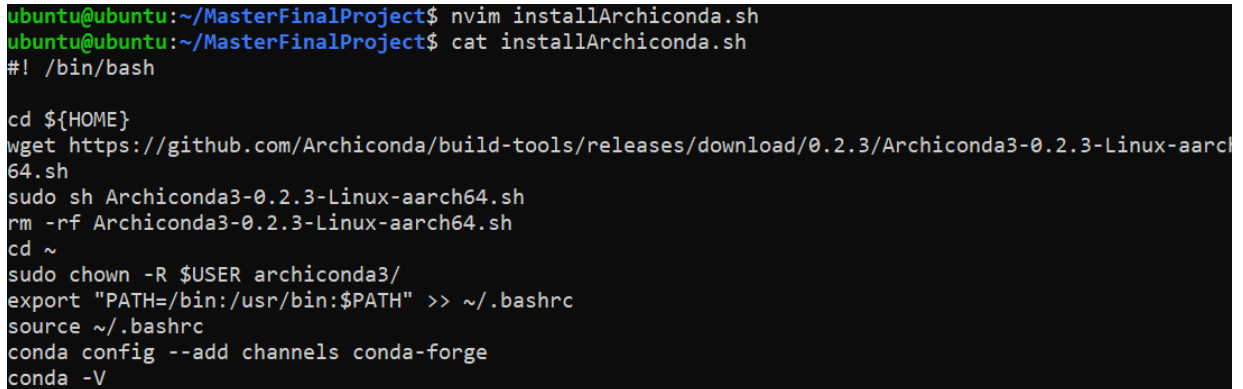
```
cd ${HOME}
```

```
wget https://github.com/Archiconda/build-tools/releases/  
download/0.2.3/Archiconda3-0.2.3-Linux-aarch64.sh
```

```
sudo sh Archiconda3-0.2.3-Linux-aarch64.sh
```

```
rm -rf Archiconda3-0.2.3-Linux-aarch64.sh
cd ~
sudo chown -R $USER archiconda3/
export "PATH=/bin:/usr/bin:$PATH" >> ~/.bashrc
source ~/.bashrc
conda config --add channels conda-forge
conda -V
```

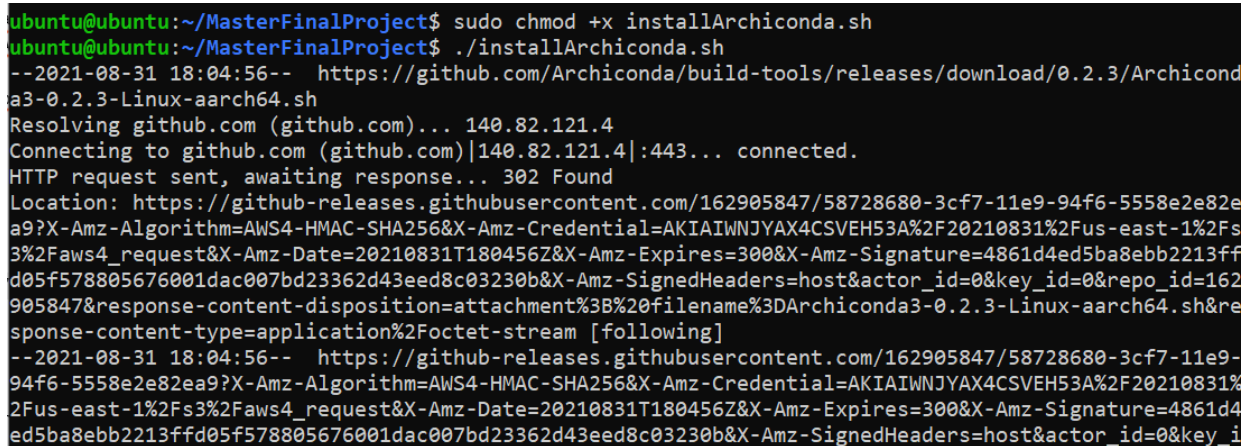
Extracto de código 2.1: Código bash



```
ubuntu@ubuntu:~/MasterFinalProject$ nvim installArchiconda.sh
ubuntu@ubuntu:~/MasterFinalProject$ cat installArchiconda.sh
#!/bin/bash

cd ${HOME}
wget https://github.com/Archiconda/build-tools/releases/download/0.2.3/Archiconda3-0.2.3-Linux-aarch64.sh
sudo sh Archiconda3-0.2.3-Linux-aarch64.sh
rm -rf Archiconda3-0.2.3-Linux-aarch64.sh
cd ~
sudo chown -R $USER archiconda3/
export "PATH=/bin:/usr/bin:$PATH" >> ~/.bashrc
source ~/.bashrc
conda config --add channels conda-forge
conda -V
```

Figura 2.5: Creación archivo instalador archiconda.



```
ubuntu@ubuntu:~/MasterFinalProject$ sudo chmod +x installArchiconda.sh
ubuntu@ubuntu:~/MasterFinalProject$ ./installArchiconda.sh
--2021-08-31 18:04:56-- https://github.com/Archiconda/build-tools/releases/download/0.2.3/Archiconda3-0.2.3-Linux-aarch64.sh
Resolving github.com (github.com)... 140.82.121.4
Connecting to github.com (github.com)|140.82.121.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://github-releases.githubusercontent.com/162905847/58728680-3cf7-11e9-94f6-5558e2e82ea9?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20210831%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20210831T180456Z&X-Amz-Expires=300&X-Amz-Signature=4861d4ed5ba8ebb2213ffd05f578805676001dac007bd23362d43eed8c03230b&X-Amz-SignedHeaders=host&actor_id=0&key_id=162905847&response-content-disposition=attachment%3B%20filename%3DArchiconda3-0.2.3-Linux-aarch64.sh&response-content-type=application%2Foctet-stream [following]
--2021-08-31 18:04:56-- https://github-releases.githubusercontent.com/162905847/58728680-3cf7-11e9-94f6-5558e2e82ea9?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20210831%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20210831T180456Z&X-Amz-Expires=300&X-Amz-Signature=4861d4ed5ba8ebb2213ffd05f578805676001dac007bd23362d43eed8c03230b&X-Amz-SignedHeaders=host&actor_id=0&key_id=162905847
```

Figura 2.6: Ejecución archivo instalador archiconda.

Una vez se ha instalado archiconda, se puede comprobar que se ha instalado correctamente ejecutando el comando `conda --version` o bien `conda -V`. Si este comando devuelve la versión de conda significará que se ha instalado correctamente y estará todo listo para crear un entorno e instalar los diferentes paquetes necesarios.

Con el gestor de paquetes conda se puede proceder a la instalación y preparación de todo el entorno de desarrollo. El primer de los pasos a seguir para poder

poner a punto el proyecto, es crear un entorno aislado donde poder compilar y donde poder alojar todos los paquetes necesarios para que a la hora de ejecutar se puedan encontrar todas las librerías en el entorno adecuado. Para poder hacer esto el primero de los comando a ejecutar es `conda create --name tflite1-env` ”.

```
ubuntu@ubuntu:~/MasterFinalProject$ conda create --name tflite1-env
Solving environment: done

## Package Plan ##

  environment location: /home/ubuntu/archiconda3/envs/tflite1-env

Proceed ([y]/n)?
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate tflite1-env
#
# To deactivate an active environment, use
#
#     $ conda deactivate
```

Figura 2.7: Ejecución archivo instalador archiconda.

Una vez creado el entorno conda donde poder ejecutar los comandos, el siguiente paso será poder ejecutar `conda activate tflite1-env` ”. De esta forma se activa el entorno conda donde se quiere trabajar y estará todo preparado para poder empezar a instalar dependencias.

```
ubuntu@ubuntu:~/MasterFinalProject$ conda activate tflite1-env
(tflite1-env) ubuntu@ubuntu:~/MasterFinalProject$
(tflite1-env) ubuntu@ubuntu:~/MasterFinalProject$
(tflite1-env) ubuntu@ubuntu:~/MasterFinalProject$
(tflite1-env) ubuntu@ubuntu:~/MasterFinalProject$
(tflite1-env) ubuntu@ubuntu:~/MasterFinalProject$
```

Figura 2.8: Ejecución archivo instalador archiconda.

El siguiente paso es instalar las dependencias, para poder instalar las dependencias desde un mismo scripts se han seguido los mismos pasos que se han seguido para la instalación de archiconda. Los requerimientos para poder ejecutar la detección de objetos vienen en el siguiente archivo ejecutable:

```
#!/bin/bash
```

```
# Get packages required for OpenCV
```



```

sudo apt-get -y install libjpeg-dev libtiff5-dev libjasper-
    dev libpng12-dev
sudo apt-get -y install libavcodec-dev libavformat-dev
    libswscale-dev libv4l-dev
sudo apt-get -y install libxvidcore-dev libx264-dev
sudo apt-get -y install qt4-dev-tools libatlas-base-dev

# Need to get an older version of OpenCV because version 4
    has errors
pip3 install opencv-python
# pip install opencv-python (If pip3 drops you an error)

# Get packages required for TensorFlow
# Using the tflite_runtime packages available at https://
    www.tensorflow.org/lite/guide/python
# Will change to just 'pip3 install tensorflow' once newer
    versions of TF are added to piwheels

#pip3 install tensorflow
# Modded for installin tflite on a RPI4 running ubuntu mate
    20.04
version=$(python -c 'import sys;_print(".".join(map(str, _
    sys.version_info[:2])))')

if [ $version == "3.7" ]; then
pip3 install --extra-index-url https://google-coral.github.
    io/py-repo/ tflite_runtime
fi

if [ $version == "3.5" ]; then
pip3 install --extra-index-url https://google-coral.github.
    io/py-repo/ tflite_runtime
fi

```

Extracto de código 2.2: Código bash

Una vez instalado el fichero de requerimientos, la raspberry estará lista para poder ejecutar una red neuronal precompilada. El siguiente de los pasos consiste en poder establecer la Coral TPU como dispositivo que será encargado de poder procesar toda esta información y poder realizar la detección de objetos.

En este punto la Raspberry es capaz de compilar y poder ejecutar la detección de objetos. Esto se puede conseguir tanto desde una entrada de vídeo como puede ser por ejemplo una webcam, o bien se puede conseguir procesar una imagen o vídeo y aplicar los coeficientes de la red neurona para poder clasificar los diferentes inputs que puedan llegar a través del vídeo.

2.3.2. Comunicación entre dispositivos

Con el objetivo de poder establecer la comunicación entre todos los dispositivos que formarán parte del robot final, se ha decidido establecer una comunicación serie a través de una UART.

2.3.3. Scripts de python

2.3.4. Imagen Linux

2.4. Entorno de desarrollo software PC

En este capítulo se expone la configuración del entorno de programación en Linux, como se estructura el proyecto y las herramientas que el entorno de Linux/-Debian ofrece.

La puesta a punto del robot a nivel de software se ha desarrollado en un entorno GNU-Linux utilizando distribuciones Debian-Ubuntu para poder instalar todas las herramientas necesarias para poder desarrollar el software que se va a implantar en el robot.

2.4.1. Sistema operativo Ubuntu

Con el objetivo de tener todas las herramientas posibles a la hora de desarrollar el proyecto y poder utilizar todas las funcionalidades que un sistema operativo puede ofrecer, se ha desarrollado el proyecto prácticamente en su totalidad utilizando Linux como base del sistema operativo y base de desarrollo tanto en PC como en la CPU que utilizará el robot.



Figura 2.9: Logotipo distribución Ubuntu.

Ubuntu es una de las distribuciones mas famosas del

2.4.2. ROS

ROS es un framework de código abierto que proporciona una serie de librerías para la gestión de desarrollo software aplicado a robots.

Entre las librerías que ROS ofrece se encuentran librerías orientadas a la abstracción de hardware, gestión de dispositivos de bajo nivel, comunicación entre sistemas... Todas estas funcionalidades que este framework proporciona hacen que la gestión de la arquitectura software del robot sea mucho mas intuitiva y manejable y por lo tanto hacer el proyecto mas escalable y abierto a nuevas funcionalidades.

Otra de la principal ventaja que hace que ROS sea uno de los frameworks mas utilizados para el desarrollo de aplicaciones con robots es la alta capacidad de reutilizar código que ofrece, esto se debe a las diferentes capas de abstracción que se utiliza en el desarrollo que hacen el desarrollo de "nodos" "paquetes" de ROS sean muy poco o nada dependientes del hardware a utilizar.



Figura 2.10: logo ROS framework.

ROS está diseñado para ser lo más ligero posible, esto hace que sea mas sencillo poder ejecutar este framework con recursos limitados, en este proyecto, ROS estará corriendo sobre una Raspberry Pi, por tanto es otra de las cualidades a tener en cuenta a la hora de escoger herramientas de software aplicadas a este robot.

Hay dos lenguajes soportados para desarrollar paquetes de ROS, se puede desarrollar tanto en C++ como en Python. Esto lo hace aún mas versátil si cabe y más accesible a mas desarrolladores abriendo el campo del desarrollo colaborativo y la reutilización de código.

2.4.3. Instalación de paquetes y herramientas necesarias en PC.

El primer paso para utilizar todas las herramientas de desarrollo necesarias es poner a punto el software que se va a utilizar. Durante todo el proyecto se han desempeñado tareas de programación y elaboración de código, para poder gestionar y llevar un seguimiento del control de versiones del código del proyecto se ha creado un repositorio donde se ha ido subiendo cada un de los diferentes cambios que se van realizando con el paso del tiempo.

La herramienta escogida para realizar el control de versiones ha sido GIT. Una de las ventajas significativas de poder utilizar un software de control de versiones

como puede ser GIT, es la portabilidad que le ofrece al código. Al utilizar repositorios como contenedor de todo el código del proyecto, este se puede clonar y ejecutar desde diferentes dispositivo sin ningún problema.

Por tanto, ha resultado ser especialmente útil para poder portar el código entre sistemas operativos así como para poder transportarlo del PC a la Raspberry Pi. De esa forma, todo el proyecto queda contenido en un mismo repositorio, a disposición de todos los usuarios que quieran desarrollar sobre él.



Figura 2.11: Logo control de versiones GIT.

Para trabajar con el control de versiones se debe de seguir una determinada metodología con el objetivo de llevar una desarrollo del proyecto de forma correcta. Generalmente a la hora de desarrollar un proyecto en GIT, se suele hacer de forma colaborativa, donde todo un equipo de desarrollo se verá involucrado en un determinado código. En este caso, solo uno o dos usuarios serán aquellos que hagan modificaciones en el código.

La metodología mas adecuada que se ha escogido para llevar a cabo el proyecto ha sido "Git Flow". Git Flow es una metodología de trabajo donde el repositorio que aloja el proyecto se estructura en diversas ramas o espacios donde se diferencian diversas partes del código y donde se estructura el código de forma que según la versión o funcionalidad se puede escoger de una forma sencilla. Generalmente la estructura de las diferentes ramas utilizadas es la que se muestra en la FIGURAX.

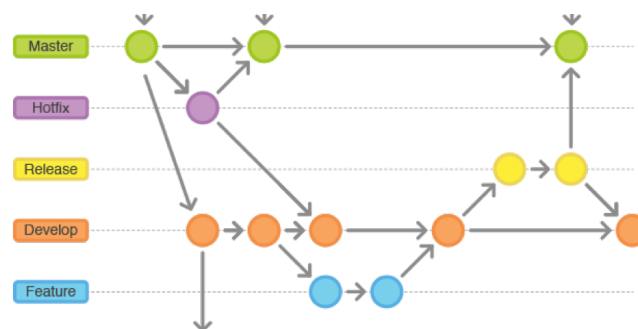


Figura 2.12: Metodología de trabajo "Git Flow".

Cada una de las ramas alberga el código con las siguientes funcionalidades:

- Master : Alberga el código que esta actualmente en producción, en este caso alberga el código que se ha utilizado en su versión completamente funcional mas reciente.
- Release : Es el último código funcional que se ha enviado para evaluar su calidad y poder enviarlo a producción, también contiene una versión definitiva del código o proyecto a utilizar pero no necesariamente debe de ser el código utilizado en producción.
- Develop : Alberga el código que está en actual desarrollo, no necesariamente debe de funcionar a la perfección, aunque es recomendable que todo el código que se pueda descargar de la rama de develop al menos debe de compilar y albergar funcionalidades básicas desarrolladas, aunque en estas se puedan encontrar diferentes bugs. Es la rama donde se encuentra el código antes de mandarlo a la rama de release.
- Feature : En la rama de feature se desarrollan las nuevas funcionalidades que se van añadir al código.

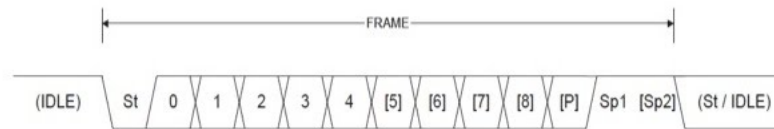
En el caso de este proyecto se ha desglosado en tres ramas, ya que el proyecto no se ha llevado a producción masiva, ha sido suficiente con tener tres ramas diferentes para estructurar de forma correcta el proyecto. En este caso se ha utilizado las ramas "develop" donde se ha ido guardando el código funcional en desarrollo, la rama "feature/" donde se iba desarrollando cada una de las funcionalidades antes de combinarlas con la rama "develop" por último la rama master, donde se ha ido metiendo el código que era completamente funcional y definitivo.

Otro ámbito importante a la hora de desarrollar el proyecto es poder realizar una correcta depuración de cada una de las partes del proyecto y de cada uno de las diferentes funcionalidades que se van desarrollando a lo largo del proyecto en diferentes ámbitos. Para poder cumplir con este objetivo se han instalado diferentes herramientas que han permitido poder obtener información interesante que verifique el correcto funcionamiento del robot.

En este caso, gran parte de la comunicación establecida entre las diferentes partes del robot, como por ejemplo la comunicación entre el microcontrolador y la Raspberry Pi se han realizado a través de un protocolo UART. La comunicación UART se establece a través de un protocolo serie que se encarga de establecer una comunicación asíncrona donde se envían y reciben datos de forma secuencial a una velocidad determinada, comúnmente llamada "baudrate".

Para poder llevar a cabo una correcta depuración se ha utilizado un software llamado hercules, el cual permite poder establecer una comunicación entre el ordenador desde el cual se esté trabajando y el dispositivo con el cual se desee comunicar. De este modo, se pueden obtener información para poder conocer en que punto de cada función está en cada momento el código ejecutado y poder mostrar en pantalla la información necesaria para su correcta depuración.

El ordenador donde se esté ejecutando el software Hercules debe de poder interpretar la información a través de un USB, por tanto es necesario convertir esa



St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.
IDLE	No transfers on the communication line (RxD or TxD). An IDLE line must be high.

Figura 2.13: Resumen protocolo UART.

información proveniente de un protocolo UART a un protocolo serie que pueda entender el ordenador.

En este caso se ha utilizado un adaptador capaz de convertir esta información para que pueda establecerse una comunicación en ambos canales y sentidos. El adaptador utilizado se puede encontrar en la siguiente figura.

Otra herramienta que resulta muy útil a la hora de poder identificar problemas y poder depurar proyectos son los analizadores lógicos. Estos dispositivos permiten poder hacer un análisis a nivel de pines de cada uno de los canales o flujos que se quieran analizar en un proyecto. En ciertas circunstancias, cuando los dispositivos que se están programando no cuentan con herramientas propias de depuración con las que poder monitorizar "logs." simplemente se quiere cerciorar que el flujo de comunicación funciona de forma correcta. Cualquier canal de comunicación o cualquier pista eléctrica que pueda conducir cualquier tipo de señal puede ser analizada con el analizador lógico, es aquí donde reside la versatilidad de este dispositivo.

Hay distintos ámbitos en los que un analizador lógico puede ser útil en este proyecto, y en este caso se ha utilizado de diferentes maneras. EL dispositivo utilizado para el proyecto es un analizador lógico genérico que es fácil de encontrar en cualquier tienda de electrónica y con un precio bastante accesible a la mayor parte de los usuarios. El dispositivo se muestra en la figura siguiente.

Para hacer uso del analizador lógico se necesita un software específico para interpretar la información adquirida por cada uno de los canales del analizador lógico. En este caso se ha utilizado un software libre y abierto llamado sigrok-pulseview, al cual se le deben de instalar los drivers compatibles con el dispositivo que se desea utilizar para poder completar la instalación y poder tener el dispositivo en pleno funcionamiento. Se puede encontrar un ejemplo de funcionamiento en la imagen siguiente.

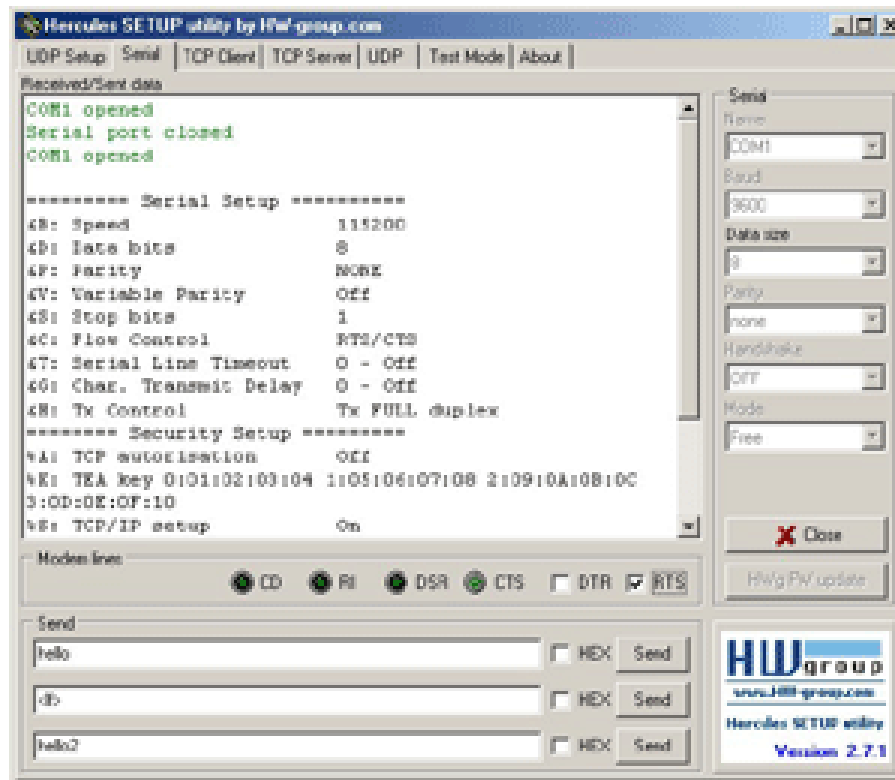


Figura 2.14: Comunicación serie en Hercules.

2.4.4. Docker Containers

Con el propósito de poder replicar el funcionamiento del robot en todos los escenarios posibles de desarrollo, se ha tratado de utilizar un contenedor Docker. Los contenedores Docker, son utilizados para encapsular y poder replicar y escalar un entorno de desarrollo determinado, donde todos y cada uno de los usuarios que utilicen dicho entorno, puedan tener las mismas herramientas disponibles para dicho desarrollo.

El modelo de robot escogido es el "rosbot description". Este modelo simula un robot de 2 ejes y cuatro ruedas implementado con una sensórica muy similar a la que se va a utilizar en el proyecto.

2.4.5. Bash Scripting

Con el objetivo de comprender como se realiza la detección de objetos, se han de dominar previamente diversos conceptos claves sobre los modelos de inteligencia artificial que se pueden encontrar actualmente en la red y disponen de la capacidad de poder embeberlos en un sistema que sea capaz de procesar esta información en el hardware que lleve el robot utilizado para el proyecto.

El modelo mas utilizado para la detección y segmentación de objetos relacionados con la robótica suelen ser las redes neuronales. Las redes neuronales favorecen



Figura 2.15: Conversor USB-TTL.



Figura 2.16: Analizador lógico utilizado en el proyecto.

el ajuste de una determinada



Figura 2.17: Analizador lógico utilizado en el proyecto.

3. Puesta en Marcha Laboratorio

3.1. Herramientas

3.2. Conclusión

En este capítulo concluimos que...

4. Conclusiones

4.1. Conclusión general

En este capítulo explicaremos...

4.2. Propuesta de futuro

5. Conclusiones
