

Práctica: Inteligencia Artificial Aplicada a Robots - 2021

Universidad
de Huelva

Daniel Pérez Rodríguez

Índice

1. Introducción.....	3
2. Conexión con la Jetson.....	3
2.1. Putty.....	3
2.2. Visual Studio Code.....	5
3. Instalación de librerías.....	7
3.1. Librería ServoKit.....	7
3.2. Librería de Nvidia para detección de objetos.....	8
4. Retransmisión remota de la cámara.....	9
4.1. Gstreamer.....	10
4.2. VLC.....	12
5. Creación del programa.....	13
5.1. Variables destacadas.....	13
5.2. Funciones.....	15
5.3. Bucle principal.....	17
6. Crear nuestro propio modelo de reconocimiento.....	19
7. Demostración de ejecución.....	21
8. Bibliografía.....	23

1. Introducción

Este documento explica como utilizar una Nvidia Jetson Nano para el reconocimiento de objetos y el seguimiento de los mismo mediante una cámara. Para ello, usaremos redes neuronales previamente entrenadas en reconocer una gran variedad de objetos. Pero que reentrenaremos para crear un modelo que se ajuste a nuestras necesidades. Los objetos a reconocer en nuestro caso serán personas. Para hacer el seguimiento de las mismas, será necesario una cámara que moveremos con la utilización de servo motores conectados a una controladora de servos por I2C modelo PCA9685. Por último, especificar que todo estos pasos se harán en el modo **headless** de la Jetson.

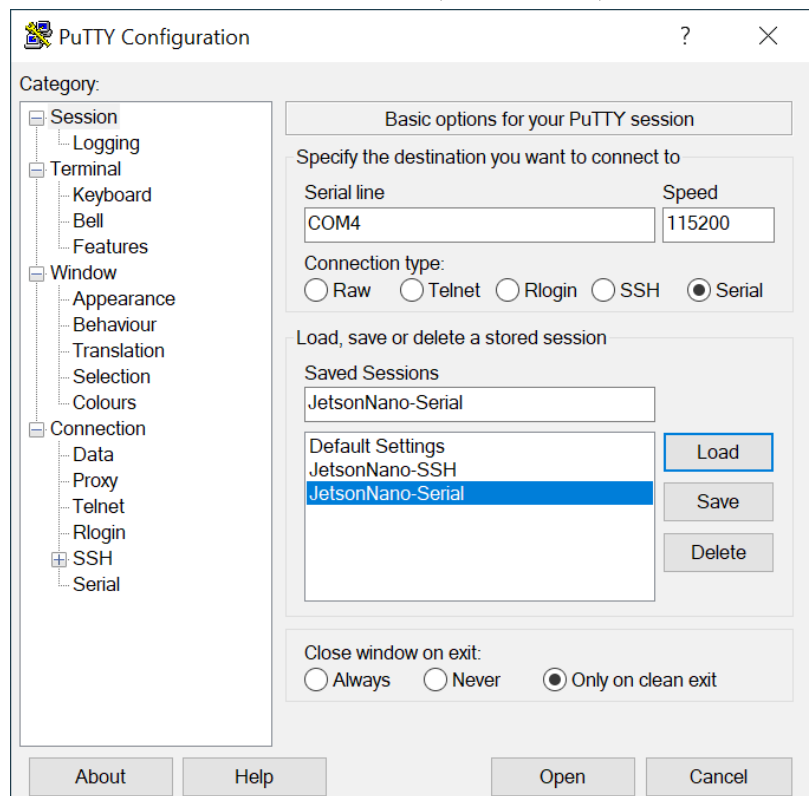
2. Conexión con la Jetson

Una vez conectado el cable micro USB, el cable Ethernet y la alimentación por barril. El primer paso es conectarse a la Jetson de forma remota. Podemos hacerlo de dos formas distintas: Putty o Visual Studio Code(VSC). La primera es la opción más rápida y sencilla, con la que usar la terminal de la Jetson de forma remota. Sin embargo, cuando sea necesario programar, VSC será una mejor opción gracias a la interfaz de usuario que ofrece. Explicaremos como conectarnos de las dos formas posibles.

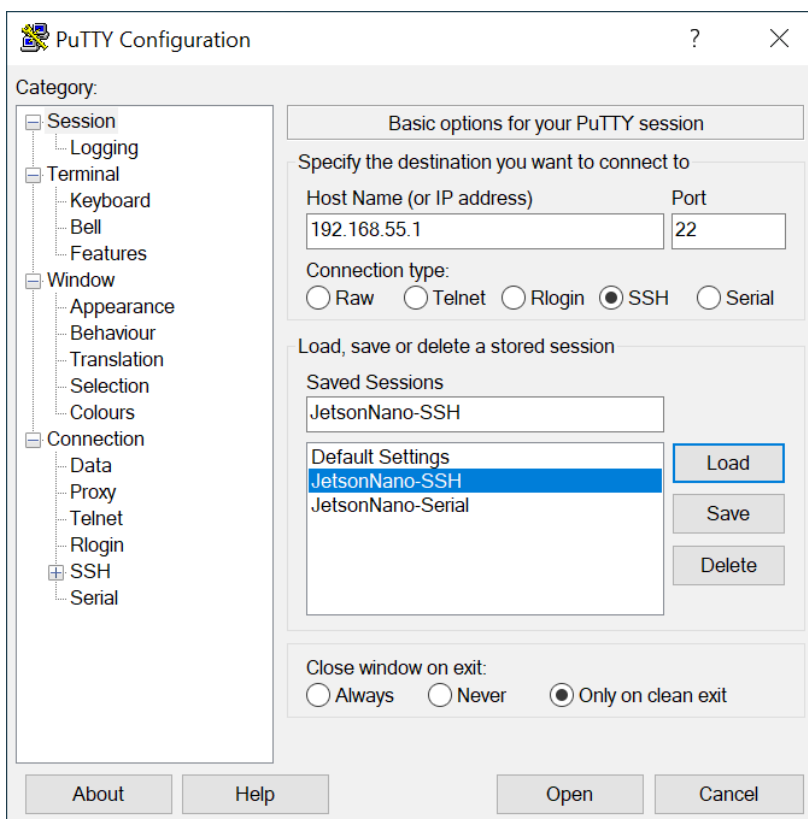
2.1. Putty

Mediante Putty podemos usar dos formas de conectarnos: por micro USB o por Ethernet. Para la primera opción debemos seleccionar el puerto COM en el que se encuentra la Jetson. Para saber en que puerto se encuentra hacemos: Tecla Windows → (Escribimos)

Administrador de dispositivos → Buscamos el apartado de puertos COM (solo aparecerá si está la Jetson conectada). Sabiendo el puerto en el que se encuentra lo siguiente es hacer la conexión en Putty. Seleccionamos la opción de conexión Serial. Especificamos el puerto COM (en nuestro caso el 4), y la velocidad, 115200. Se recomienda pulsar el botón Save para guardar la configuración. Por último pulsamos Intro o el botón Open para establecer la conexión.



Para conectarnos mediante el cable Ethernet, en Putty seleccionaremos la opción de conexión SSH. Nos pedirá la dirección IP de la Jetson (IP por defecto – 192.168.55.1) y el puerto de conexión, siendo el 22. Como con la opción anterior se recomienda guardar la configuración. Por último, pulsamos Intro o el botón Open.



Una vez se establezca la conexión, deberemos introducir usuario y contraseña para poder acceder a la Jetson. Con los datos introducidos ya podemos usar la Jetson de forma remota.

```
COM4 - PuTTY
Ubuntu 18.04.5 LTS Jetson1 ttyGS0
Jetson1 login: alumno1
Password:
Last login: lun jul  5 21:43:36 CEST 2021 on ttyGS0
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 4.9.201-tegra aarch64)

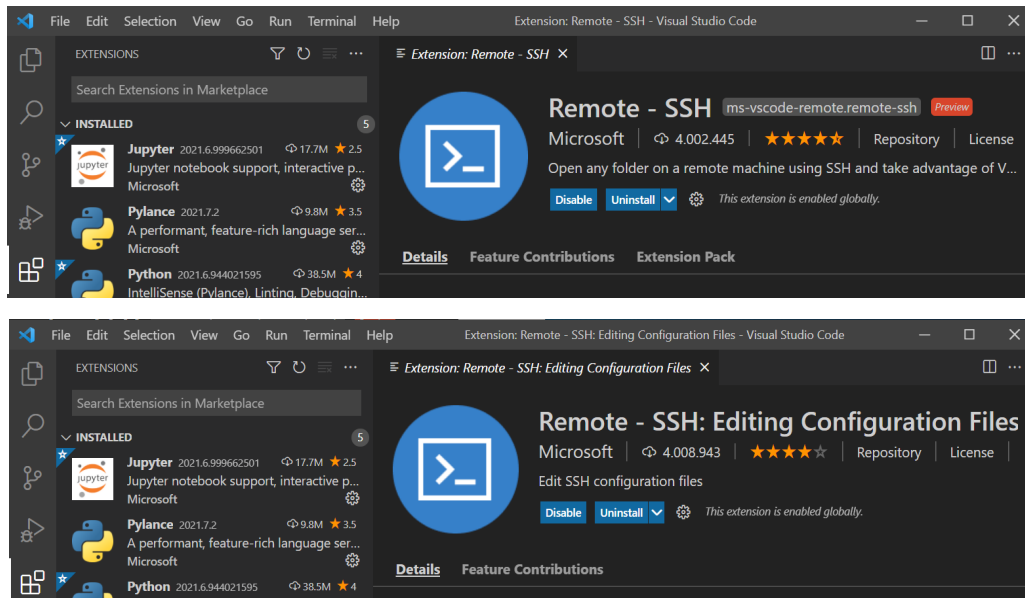
 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage
This system has been minimized by removing packages and content that are
not required on a system that users do not log into.


To restore this content, you can run the 'unminimize' command.

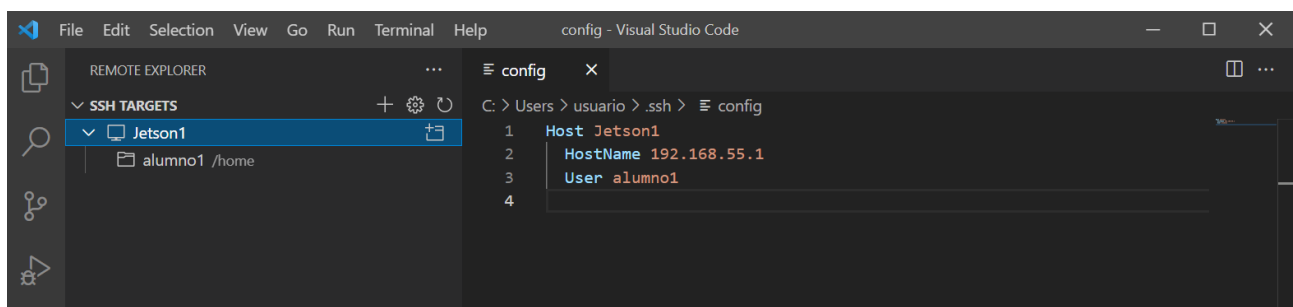
Pueden actualizarse 241 paquetes.
150 actualizaciones son de seguridad.
alumno1@Jetson1:~$
```

2.2. Visual Studio Code

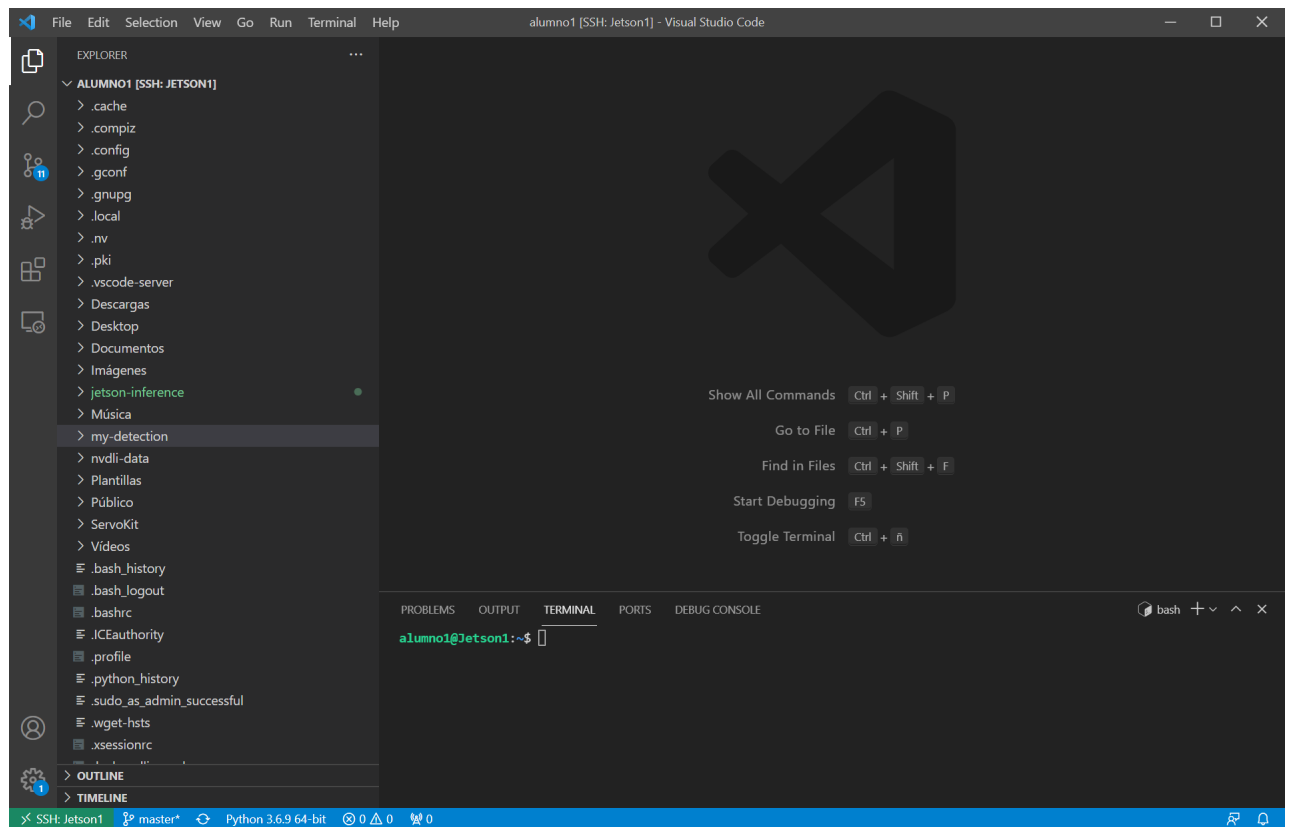
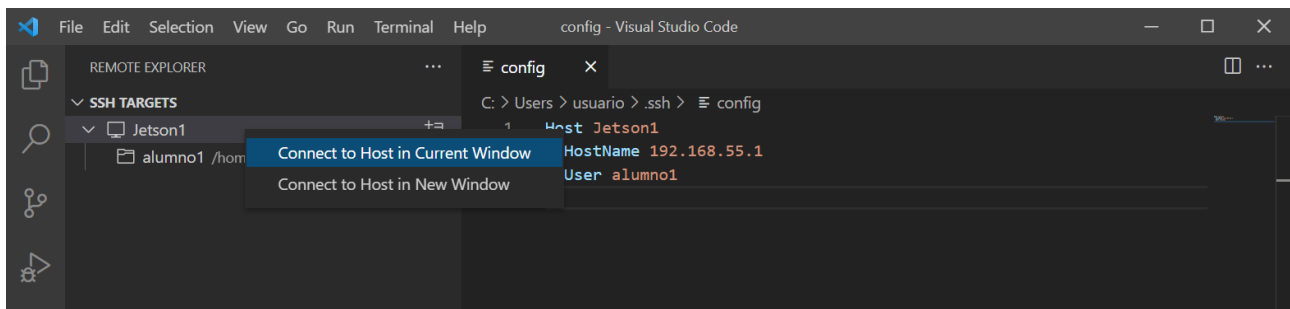
Para poder usar VSC será necesario instalar dos extensiones en el programa. La primera se llama *Remote - SSH*, nos permitirá conectarnos a la Jetson mediante una conexión SSH. Y la segunda es *Remote - SSH: Editing Configuration Files*. Esta nos permitirá poder visualizar las carpetas y su contenido de una forma visual.



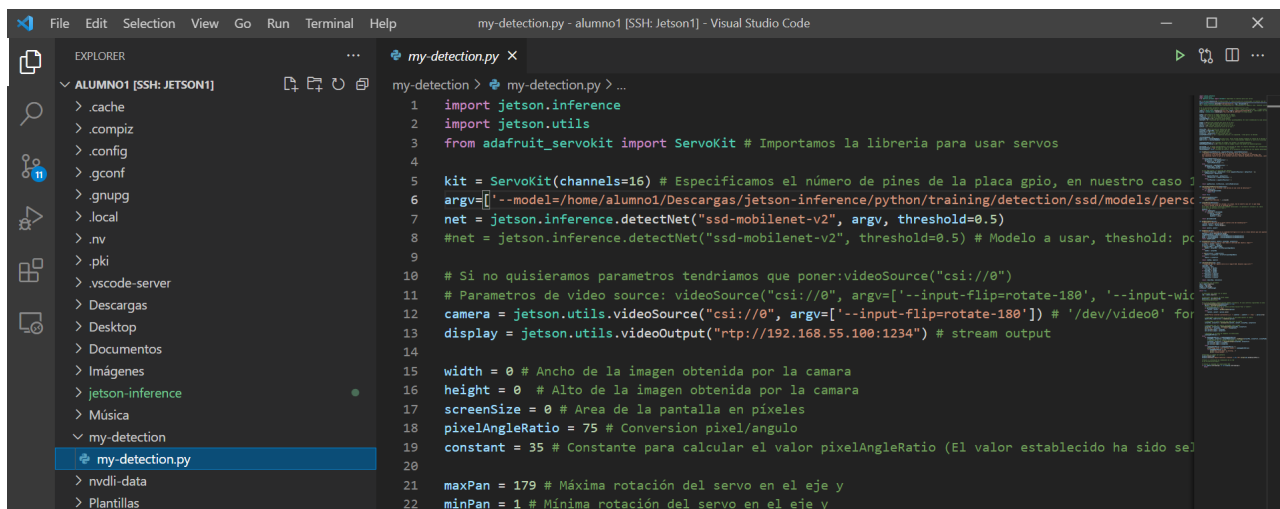
Con los módulos instalados pulsaremos el botón de conexión remota() y en el panel lateral izquierdo pulsaremos en el botón '+' para añadir una conexión. Nos pedirá a que sistema operativo nos vamos a conectar, en nuestro caso Linux. Después nos preguntará donde guardar el archivo de configuración de la conexión, seleccionamos la primera opción y en el archivo de configuración deberemos escribir los siguiente. Primero debemos poner un nombre para la conexión en la entrada Host, seguidamente la dirección IP de la Jetson en la entrada Hostname. Y por último, el nombre de usuario con el que conectarnos a la Jetson. Finalmente podemos guardar el archivo.



Para establecer conexión deberemos pulsar click derecho sobre la entrada con nombre Jetson1 y Connect to Host in Current Window para establecer conexión. Nos pedirá la contraseña del usuario establecido en el archivo de configuración, una vez introducida, podremos usar la Jetson de forma remota.



Desde aquí podremos ver los archivos que contiene la Jetson, usar la terminal, editar archivos y por supuesto programar



3. Instalación de librerías

Será necesario instalar dos librerías. Una para el control de los servo motores de la cámara y otra para realizar el reconocimiento de objetos.

3.1. Librería ServoKit

Esta librería nos permitirá controlar los servo motores. Primero debemos asegurarnos que tenemos instalado pip en el sistema. Tecleamos lo siguiente en la consola:

- `sudo apt-get install python3-pip`

Con pip instalado podremos instalar la librería. Tecleamos lo siguiente:

- `sudo pip3 install adafruit-circuitpython-servokit`

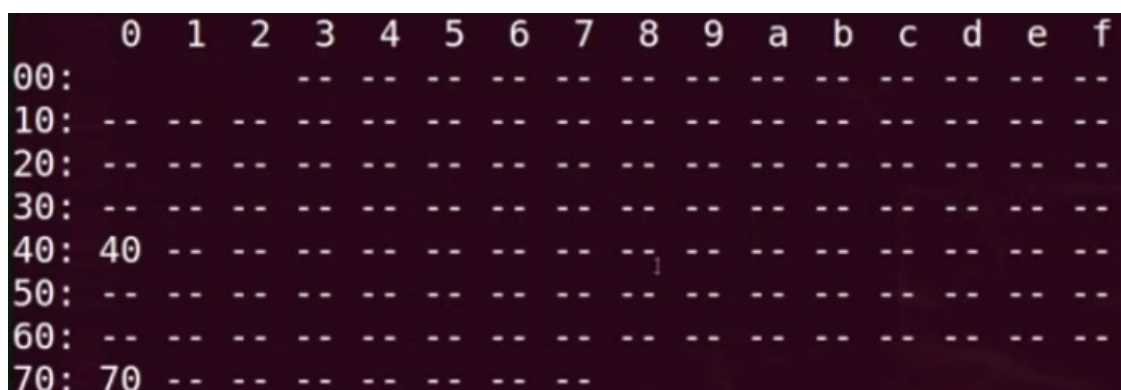
Ahora tenemos que darle permisos al usuario para controlar los servos. Tecleamos lo siguiente:

- `sudo usermod -aG i2c USUARIO`
- `sudo groupadd -f -r gpio`
- `sudo usermod -a -G gpio USUARIO`
- `cp /opt/nvidia/jetson-gpio/etc/99-gpio.rules /etc/udev/rules.d/`
- `sudo udevadm control --reload-rules && sudo udevadm trigger`
- `sudo reboot`

Con esto la librería ya estaría instalada. Para asegurarnos de que detectamos la PCA correctamente usaremos el siguiente comando:

- `sudo i2cdetect -y -r 1`

Teniendo que mostrar lo siguiente:



```
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: 40 -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: 70 -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

Ahora ya podremos agregar la librería en un programa para python.

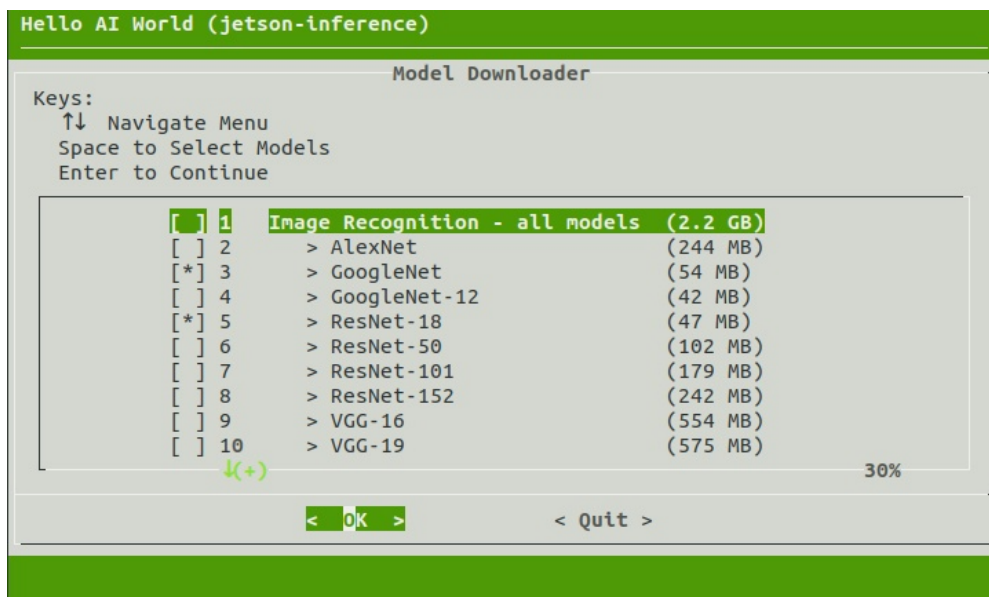
3.2. Librería de Nvidia para detección de objetos

La instalación de esta librería será necesario para el reconocimiento de objetos a través de la cámara. Toda la información sobre la misma se encuentra en este repositorio de github: <https://github.com/dusty-nv/jetson-inference>. En esta memoria solo se de describirán algunas posibilidades de la misma.

Durante el tutorial nos descargaremos el repositorio, esto se puede hacer en cualquier carpeta. La carpeta elegida ha sido la carpeta Downloads. Tecleamos lo siguiente en consola:

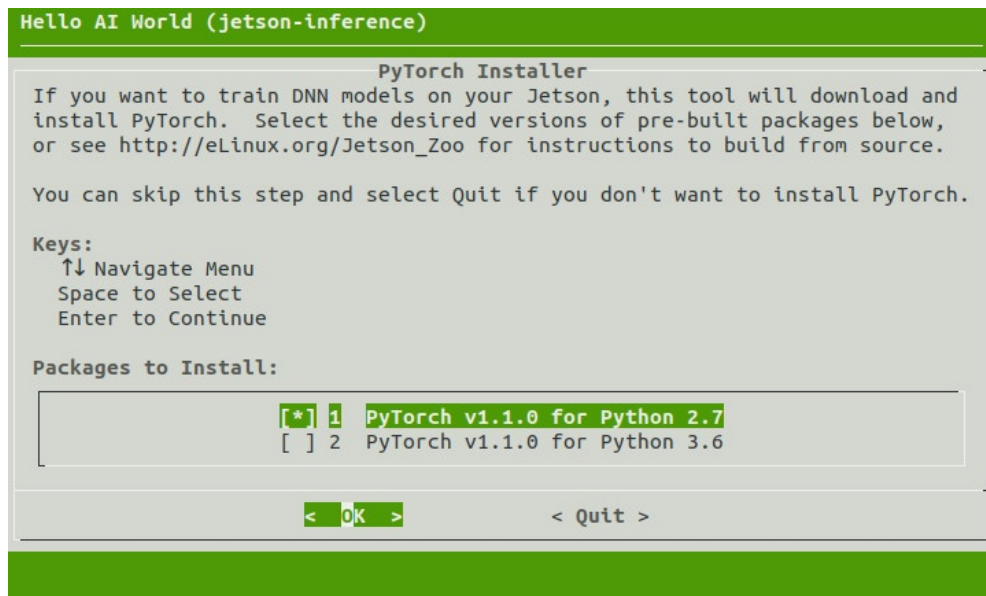
- sudo apt-get install git cmake libpython3-dev python3-numpy
- cd ~/Downloads
- git clone --recursive <https://github.com/dusty-nv/jetson-inference>
- cd jetson-inference
- mkdir build
- cd build
- cmake ../

Este el último comando durará bastante. Pidiendo en algunas partes seleccionar algunas opciones. En la primera nos pedirá que seleccionemos redes previamente entrenadas que queramos utilizar. Cada una es distinta y se ha entrenado para una tarea de clasificación diferente. Nosotros descargaremos las que están seleccionadas por defecto. Podemos consultar la especialización de cada red en esta página: <https://github.com/dusty-nv/jetson-inference/blob/master/docs/detectnet-console-2.md>



Aclarar que se nos permite descargar redes para reconocimiento de imágenes, detección de objetos y segmentación de imágenes. Siendo la segunda categoría mencionada la que nos interesa para esta explicación.

Seguidamente nos pedirá que instalemos una versión de Pytorch. Debemos seleccionar la última versión para Python. En este momento será la versión de Python 3.6



Por último deberemos introducir los siguiente comandos para terminar el proceso de instalación.

- make -j\$(nproc)
- sudo make install
- sudo ldconfig
- sudo apt-get install v4l-utils

Para asegurarnos que todo se ha instalado correctamente teclearemos lo siguiente:

- python3
- import torch
- import torchvision

Si no se ha mostrado ningún error es que la librería se ha instalado correctamente. Pulsamos Crtl+D para salir del bash de python3

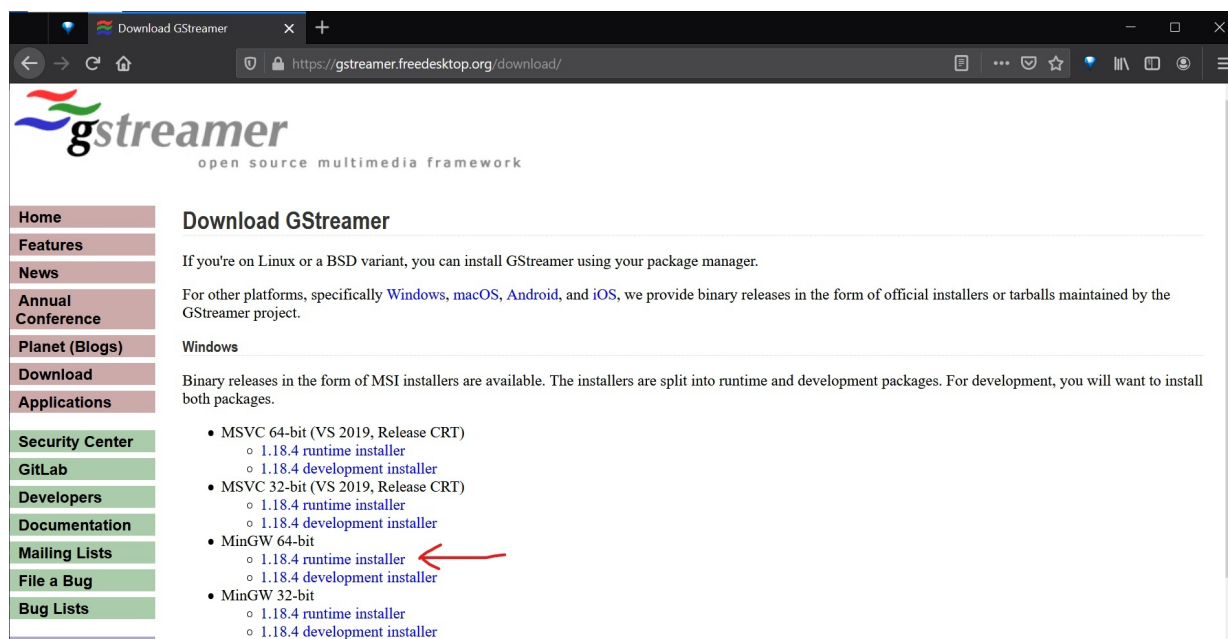
4. Retransmisión remota de la cámara

Este paso será necesario para poder visualizar la vista de la cámara así como los objetos que esta reconociendo. Y una vez vez más tenemos dos alternativas: Gstreamer y VLC.

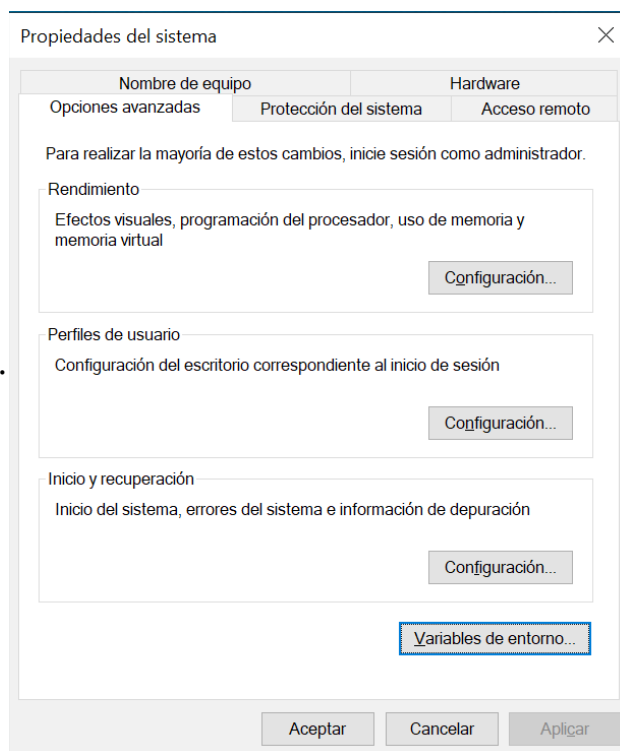
Gstreamer es el recomendado, ya que no presenta ningún tipo de delay a la hora de ver las imágenes, sin embargo, es más complicado de instalar. Por otro lado, VLC es muy sencillo de configurar para habilitar la retransmisión. Como contra partida presenta un mayor retraso en la imagen que visualiza la cámara. Mostramos como configurar ambos.

4.1. Gstreamer

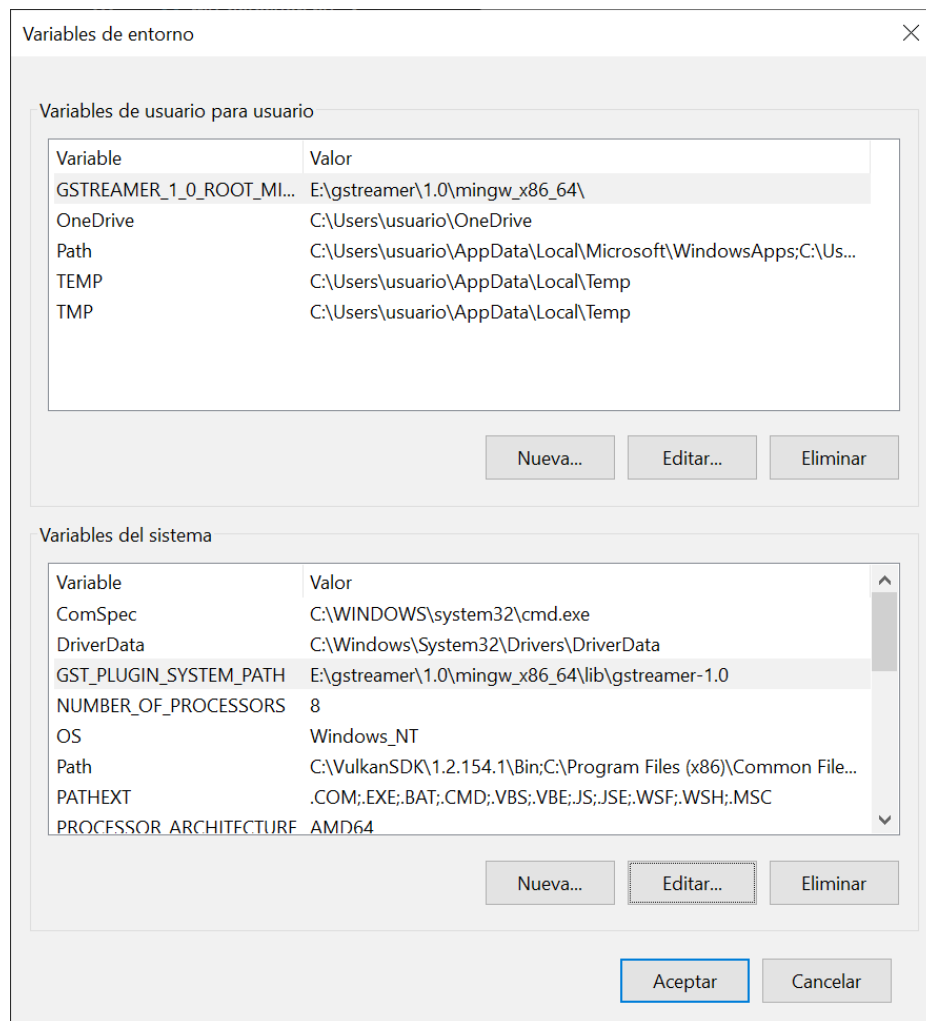
El primer paso será descargar el programa Gstreamer. Para ello, accederemos a la siguiente web: <https://gstreamer.freedesktop.org/download/>. En la misma deberemos descargar la versión MinGW. Seleccionaremos en función de la versión de 32 o 64 bits.



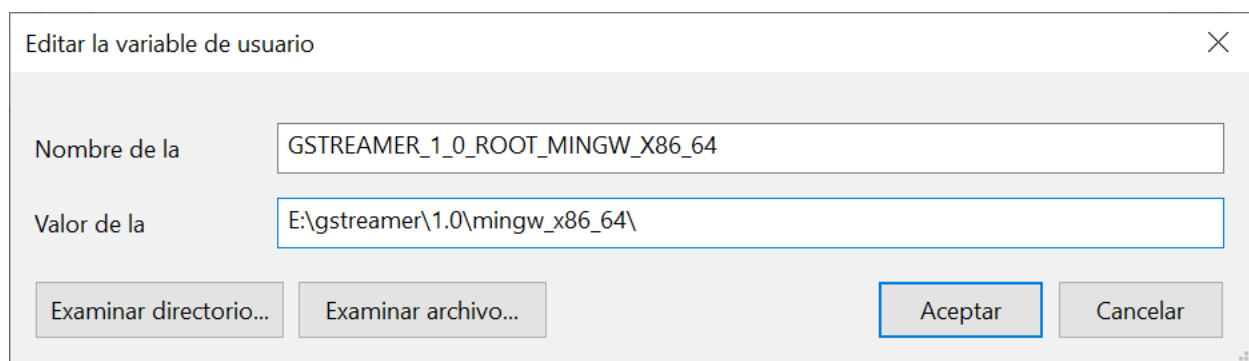
Una vez instalado, deberemos modificar el PATH de Windows. Esto es así porque Gstreamer se usa desde la consola de Windows. Para modificar el PATH deberemos pulsar la tecla de Windows y escribir la palabra 'path'. Pulsamos en la aplicación de *Edición de variables del sistema* y nos aparecerá la siguiente ventana.



Ahora pulsamos en el botón de Variables de entorno. Se nos abrirá un nuevo menú y deberemos añadir dos nuevas variables. Una nueva variable de usuario y otra variable de sistema.

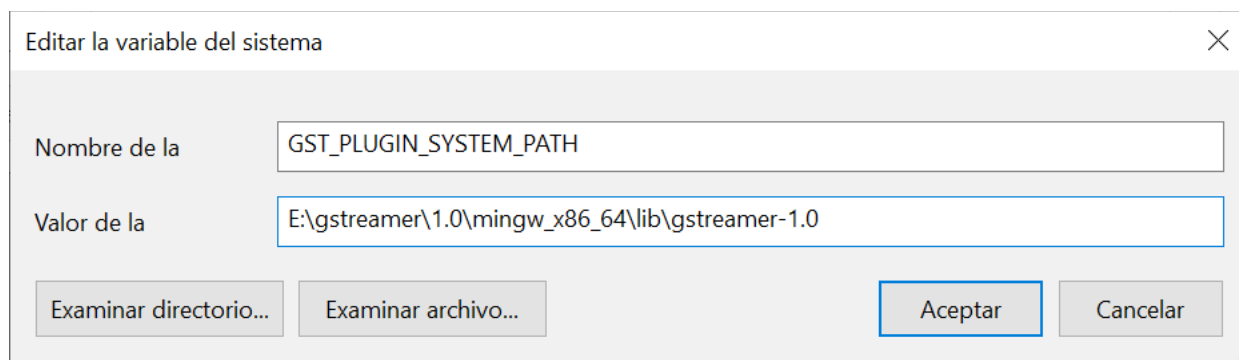


Pulsaremos en el botón de Nueva para añadir cada una de las variables. Para la variable de usuario pondremos lo siguiente:



El apartado ‘Valor de la’ es la ruta de la carpeta mingw_x86_64. Esta ruta dependerá de donde se haya instalado el programa Gstreamer en el primer paso. En nuestro caso, el programa Gstreamer se ha instalado en el disco E. Pulsamos Aceptar para confirmar.

Para la variable del sistema pondremos lo siguiente:



Como con la variable de usuario, la ruta a especificar dependerá de donde se haya instalado el programa. Pulsamos aceptar para terminar de añadir la variable

Con esto el programa ya esta instalado. Para usarlo deberemos abrir una consola de Windows y escribir o pegar el siguiente comando:

```
- gst-launch-1.0 -v udpsrc port=1234 caps = "application/x-rtp, media=(string)video, clock-rate=(int)90000, encoding-name=(string)H264, payload=(int)96" ! rtph264depay !  
decodebin ! videoconvert ! autovideosink
```

Importante: En el parámetro ‘port=’ se deberá especificar un puerto. Este puerto se establecerá cuando queramos que la Jetson se comunice con el PC remoto desde el código del programa que especificaremos más adelante. Por defecto usaremos el puerto 1234.

Cuando queramos dejar de ver la retransmisión podemos pulsar Ctrl+C para dejar de ver la retransmisión.

4.2. VLC

Para VLC también será necesario descargar el programa. Pero siendo ampliamente usado y conocido no explicaremos como descargarlo. Deberemos crear un fichero de texto y escribir lo siguiente:

```
c=IN IP4 192.168.55.100
```

```
m=video 1234 RTP/AVP 96
```

```
a=rtpmap:96 H264/90000
```

En la primera línea deberemos especificar la dirección IP del host. Que en este caso es la IP del PC que estamos usando para usar la Jetson de forma remota. Después, en la segunda

deberemos especificar el puerto a usar. Usaremos el puerto por defecto 1234. Finalmente lo guardamos como un archivo plano de texto con la extensión .sdp

Cuando deseemos visualizar la imagen de la cámara solo deberemos hacer doble click en el archivo creado.

5. Creación del programa

Ahora es el momento de desarrollar el código que realice el seguimiento de los objetos. Las librerías de Nvidia permiten programar en C++ y Python. Siendo el segundo lenguaje la opción utilizada para este código. El primer paso es importar las librerías previamente instaladas:

```
import jetson.inference
import jetson.utils
from adafruit_servokit import ServoKit
```

5.1. Variables destacadas

Tendremos que crear 4 objetos principales que serán indispensables para realización de la práctica. Explicaremos a continuación que hace cada uno de las siguientes objetos:

- **kit** → La variable kit nos permitirá mover los servos motores de la cámara. Este movimiento está en el rango (0-180) grados. Para poder usarlo escribiremos la siguiente sentencia:

```
kit = ServoKit(channels=16)
kit.servo[0].angle = actualPan
```

Siendo channels el número de pines con las que cuenta nuestra PCA.

- **net** → Variable del tipo detectNet encargada de reconocer los objetos presentes en las imágenes capturadas por la cámara. Una vez que creamos la variable, debemos especificar que red queremos usar. Podemos usar una de las redes previamente descargas o un modelo propio. Para usar red previamente entrenada solo tendremos que especificar el nombre de la red y su umbral de reconocimiento, es decir, el valor mínimo de confianza con el que detectar un objeto. Para ver la descripción completa de la clase consultar:

<https://rawgit.com/dusty-nv/jetson-inference/dev/docs/html/python/jetson.inference.html#detectNet>

```
net = jetson.inference.detectNet("ssd-mobilenet-v2", threshold=0.5)
```

En este caso estamos cargando la red “ssd-mobilenet-v2” y estamos poniendo un umbral del 50%. En el caso de queramos usar nuestro propio modelo deberemos entrenarlo previamente (explicación en el punto 6). Una vez entrenado, deberemos cargar el modelo, sus

etiquetas(clases a reconocer) y especificar los parámetros de representación en pantalla. Para ello crearemos una variable que contenga toda la información:

```
argv=[ '--model=/home/alumno1/Descargas/jetson-inference/python/  
training/detection/ssd/models/person/ssd-mobilenet.onnx', '--  
labels=/home/alumno1/Descargas/jetson-inference/python/training/  
detection/ssd/models/person/labels.txt', '--input-blob=input_0', '--  
output-cvg=scores', '--output-bbox=boxes' ]
```

Especificando en el etiqueta '--model=' la ruta absoluta de donde se encuentra el modelo entrenado, así como el la ruta absoluta de las clases que contiene el modelo en la etiqueta '--labels='. Finalmente lo agregamos a la sentencia anterior como parámetro:

```
net = jetson.inference.detectNet("ssd-mobilenet-v2", argv, threshol  
d=0.5)
```

- **camera** → Nos permitirá capturar las imágenes para su posterior reconocimiento. En la creación del objeto deberemos especificar la cámara que vayamos a usar. En nuestro caso estamos usando el puerto CSI, por lo que deberemos especificarlo con 'csi://0'. Siendo 0 el valor que le ha dado linux a nuestro dispositivo de captura. En caso de usar una cámara USB deberemos especificarlo con el nombre '/dev/video0'. Para comprobar las cámaras USBs detectadas por el sistema, deberemos mostrar la ruta '/dev' en una terminal.

```
camera = jetson.utils.videoSource("csi://0", argv=['--input-  
flip=rotate-180'])
```

La variable argv nos permite añadir parámetros adicionales al objeto. En nuestro caso, es necesario rotar la imagen de entrada 180 grados para que se vea de forma correcta. La lista completa de parámetros se pueden consultar en la siguiente página:

<https://github.com/dusty-nv/jetson-inference/blob/master/docs/aux-streaming.md>.

- **display** → Variable encargada de retransmitir la señal de la cámara al PC remoto. Para ello deberemos especificar la dirección IP del PC al que enviarle la señal, así como el puerto. Como se explico en el punto 4, usaremos el puerto por defecto, el 1234. Es recomendable asegurarse previamente de que el PC al que enviaremos la transmisión es visible para la Jetson, para ello se puede usar el comando ping en la terminal de linux. Creamos el objeto:

```
display = jetson.utils.videoOutput("rtp://192.168.55.100:1234")
```

Por último, debemos explicar que significan las palabras pan y tilt, ya que serán nombradas durante la explicación del código. Cuando hablemos de panPosition nos estaremos refiriendo a la posición del servo motor que controla la rotación correspondiente al eje Y. Por otro lado, cuando hablemos de tiltPosition, nos referimos a la posición del servo motor que controla la rotación sobre el eje X.

5.2. Funciones

Ahora pasamos a explicar las funciones utilizadas en el bucle principal del código:

```
def findPeople(anglePanPosition, angleTiltPosition, controlPanDirection):
```

- Descripción: Devuelve la siguiente posición en la que debe estar la cámara en su estado de encontrar personas a las que seguir. Este movimiento es horizontal.

- Parámetros:

- anglePanPosition: Ángulo de pan en el que se encuentra el servo motor.
- angleTiltPosition: Ángulo de tilt en el que se encuentra el servo motor.
- controlPanDirection: Indica la dirección en la que se está moviendo la cámara sobre el eje Y. El valor 1 significa que girará hacia la izquierda, mientras que el valor -1 a la derecha.

- Devuelve:

- (panPosition, tiltPosition, controlPanDirection): Nuevas posiciones de pan y tilt, así como la dirección en la que se debe mover la cámara en el eje Y.

```
def thereIsPeopleOnDetections(detections):
```

- Descripción: Comprueba si se ha reconocido a una persona en los objetos detectados

- Parámetros:

- detectNet[]: Lista de objetos que se han reconocido sobre una imagen.

- Devuelve:

- boolean: Devuelve True en caso de haber reconocido a una persona y False en caso contrario

```
def showClassID(detections):
```

- Descripción: Muestra el valor de identificación de cada objeto reconocido por la terminal.

- Parámetros:

- detectNet[]: Lista de objetos que se han reconocido sobre una imagen.

Mencionar que el código para una persona es igual al valor 1.

```
def selectPerson(detections):
```

- Descripción: Selecciona la persona más cercana a la cámara de entre una lista de detecciones

- Parámetros:

- detectNet[]: Lista de objetos que se han reconocido sobre una imagen.

- Devuelve:
 - detectNet: Objeto detectNet que contiene a la persona más cercana a la cámara.

```
def getBboxTopZone(person):
```

- Descripción: Devuelve la posición de la parte superior central del bounding box de un objeto reconocido

- Parámetros:
 - detectNet: Objeto sobre el cuál comprobar su bounding box.

- Devuelve:
 - (centerX, centerY): Posición de la pantalla en la que se encuentra la posición superior central del bounding box del objeto reconocido.

```
def deadZones(person):
```

- Descripción: Devuelve el error máximo en pan y en tilt permitidos, es decir, la máxima desviación del centro de la cámara respecto al centro del objeto que se está siguiendo. Los objetos más cercanos a la cámara presentarán un error máximo mayor, en comparación con los objetos más lejanos.

- Parámetros:
 - detectNet: Objeto sobre el cuál comprobar el error de pan y tilt máximo.

- Devuelve:
 - (maxEP,maxET): siendo maxEP el valor de máximo de error en pan y maxET el valor de error máximo en tilt.

```
def moveCameraTo(centerX, centerY, actualPan, actualTilt):
```

- Descripción: Mueve la cámara a la posición especificada por parámetro

- Parámetros:
 - centerX: Posición de la pantalla en el eje X al cual mover la cámara.
 - centerY: Posición de la pantalla en el eje Y al cual mover la cámara.
 - actualPan: Valor actual del pan.
 - actualTilt: Valor actual del tilt.

- Devuelve:

- (newPan, newTilt): Nuevos valores de pan y tilt centrados en la posición especificada por parámetro.

```
def checkCameraAngle(pan, tilt):
```

- Descripción: Comprueba que los ángulos pasados por parámetros estén en el rango (0-180) grados.

- Parámetros:

- pan: Valor de pan a comprobar.

- tilt: Valor de tilt a comprobar.

- Devuelve:

- (returnPan, returnTilt): Devuelve el mismo valor que los pasados por parámetros en caso de estar en el rango permitido. En caso contrario se le asignará el máximo o el mínimo valor permitido, dependiendo de si el valor es igual o inferior a 0 o si es igual o superior a 180.

5.3. Bucle principal

Capturamos la primera imagen e inicializamos las variables width, height y screenSize. Que nos dará el tamaño de la imagen que obtenemos de la cámara.

```
img = camera.Capture()
width = img.width
height = img.height
screenSize = width*height
```

Ejecutamos un bucle infinito. En caso de querer detenerlo deberemos pulsar Ctrl+C en la terminal que hemos lanzado el programa.

```
while True:
```

Ahora capturamos una imagen con la webcam.

```
img = camera.Capture()
```

Detectamos los objetos que se encuentran en dicha imagen. En caso de querer mostrar la clase de cada objeto reconocido descomentaremos la segunda sentencia.

```
detections = net.Detect(img)
#showClassID(detections)
```

Ahora debemos comprobar si se ha reconocido alguna persona en la lista de detections

```
if thereIsPeopleOnDetections(detections) is True:
```

En caso de devolver el valor True haremos lo siguiente. Primero seleccionaremos a la persona más cercana a la cámara, y a continuación, especificaremos que zona de su

bounding box debemos seguir. Si el centro o su parte superior central. Para la primera opción lo especificaremos con la palabra “center” y para la segunda con la palabra “top”.

```
person = selectPerson(detections)
if personZoneToFind == "top":
    centerX, centerY = getBboxTopZone(person)
else:
    centerX, centerY = person.Center
```

Por último mostramos por la terminal el centro de la persona y su área.

```
print("Person selected coordinates(x,y): (",centerX,",",centerY,") - Area: ", person.Area)
```

Ahora comprobamos como de grande puede ser el error de pan y tilt para posicionarnos en el centro de la persona.

```
maxErrorPan, maxErrorTilt = deadZones(person)
```

Actualizamos los valores de pan y tilt. Para ello calculamos su nuevo valor y comprobamos que estos sean válidos. Por último, mostramos los nuevos valores en la terminal.

```
actualPan, actualTilt = moveCameraTo(centerX, centerY, actualPan, actualTilt)
actualPan, actualTilt = checkCameraAngle(actualPan, actualTilt)
print("Pan:", actualPan, " - Tilt:", actualTilt)
```

Ahora con los valores de pan y tilt ya calculados, movemos los servo motores.

Correspondiéndose el servo motor 0 con el valor de pan y el servo motor 1 con el valor de tilt.

```
kit.servo[0].angle = actualPan
kit.servo[1].angle = actualTilt
```

Por último, reseteamos la variable numImageWithoutPerson (se explica a continuación).

```
NumImageWithPerson = 0
```

En caso de que en la sentencia:

```
if thereIsPeopleOnDetections(detections) is True:
```

se hubiese devuelto un valor False, deberemos vigilar la zona en busca de una nueva persona a la que seguir.

```
else:
```

Deberemos vigilar la zona en caso de que el número de imágenes capturadas desde que se perdió a la persona de seguimiento (numImageWithoutPerson), sea igual o mayor al número máximo de capturas tras perder a una persona (maxNumImageWithoutPerson). Si esto se cumple, calcularemos la nueva posición de la cámara mediante la función findPeople(). En caso contrario, aumentaremos el valor de la variable numImageWhitoutPerson, mostraremos en la terminal el valor actual de imágenes tras la pérdida del objeto de seguimiento y esperaremos que en la siguiente imagen vuelva a reconocer a una persona. Esto se hace así

porque en ocasiones, la red puede no reconocer adecuadamente el objeto de seguimiento y no detectarlo entre una imagen y otra. Esto dependerá del umbral que hayamos establecido en la creación del objeto net.

```
if numImageWithoutPerson >= maxNumImageWithoutPerson:
    actualPan, actualTilt, actualPanDirection = findPeople(actualPan, actualTilt,
actualPanDirection)
    actualPan, actualTilt = checkCameraAngle(actualPan, actualTilt)
    kit.servo[0].angle = actualPan
    kit.servo[1].angle = actualTilt
else:
    numImageWithoutPerson = numImageWithoutPerson + 1
    print("Image after lose the track object:", numImageWithoutPerson)
    if numImageWithoutPerson >= maxNumImageWithoutPerson:
        print("The track object is missing...")
        print("Finding people...")
```

Finalmente, mostramos la imagen capturada por pantalla y en caso de que haya un problema en la captura de imágenes o en su posterior muestreo en la pantalla, salimos del bucle.

```
display.Render(img)
display.SetStatus("Object Detection | Network {:.0f} FPS".format(net.GetNetworkFPS()))
if not camera.IsStreaming() or not display.IsStreaming():
    break
```

6. Crear nuestro propio modelo de reconocimiento

En el apartado anterior hemos explicado como reconocer y seguir objetos usando una red previamente ya entrenada en reconocer objetos. Ahora vamos a reentrenar la red para que se especialice en reconocer el objeto que nosotros deseemos. Para llevar esto a cabo, vamos a usar imágenes ya catalogadas en la página web: Open Images Dataset V6 (<https://storage.googleapis.com/openimages/web/index.html>). Esta web ofrece un gran catálogo de imágenes preparadas para reentrenar una red, ahorrando mucho trabajo de cara al programador. Cuanta con más de 600 tipos de objetos que podemos utilizar. Pero esto no significa que no podamos crear nuestro propio dataset. Para hacerlo deberemos ejecutar la Jetson en modo display y tras hacer capturas con la cámara, tenemos que clasificar los objetos que se encuentra en cada una de las imágenes. En esta explicación no enseñaremos a crear nuestro propio dataset, pero en caso de ser necesario, Nvidia tiene un tutorial explicando el proceso: https://youtu.be/2XMkPW_sIGg. Ahora procederemos a descargar las imágenes que deseemos para nuestro modelo, en nuestro caso serán imágenes de personas. Pero primero debemos realizar los siguientes pasos:

Primero nos deberemos ir a la carpeta en la que se ha descargado la librería de detección de objetos de Nvidia. Que en nuestro caso es en la carpeta Downloads.

- cd ~/Downloads

Una vez ahí deberemos ejecutar los siguientes comandos:

- cd jetson-inference/python/training/detection/ssd

- wget https://nvidia.box.com/shared/static/djf5w54rjvpqocsiztzaandq1m3avr7c.pth -O models/mobilenet-v1-ssd-mp-0_675.pth

- pip3 install -v -r requirements.txt

Los elementos necesarios para reentrenar la red se encuentran en la carpeta ssd en la que nos encontramos ahora.

Con todo preparado podemos empezar a descargar las imágenes que sean de nuestro interés. Para ello deberemos escribir el siguiente comando:

- python3 open_images_downloader.py --stats-only --class-names "Person" --data=data/person

Explicamos que hace cada etiqueta:

'--stats-only' → Nos permite visualizar el número total de imágenes disponibles con las clases especificadas. En muchas ocasiones habrá disponibles más de 200.000 imágenes disponibles, ocupando más de 250GB. Además de alargar los tiempos de entrenamiento de forma exponencial. Se recomienda no superar las 10.000 imágenes para no alargar de forma considerable los tiempos de entrenamiento.

'--class-names' → Nos permite seleccionar que tipos de imágenes queremos descargar. Para ello deberemos estipular entre comillas dobles las clases deseadas. Por ejemplo: "Apple,Orange,Banana". Los nombres de las clases se pueden encontrar fácilmente en el explorador de Open Images Dataset V6.

'--data=' → Aquí especificamos la ruta donde almacenar el dataset. Siendo recomendable guardarlo siempre en la carpeta data/ seguido del nombre que le queramos dar a nuestro dataset. En nuestro caso lo guardaremos en la carpeta data/person.

Una vez que hayamos decidido las clases y el número total de imágenes a descargar, especificaremos con la etiqueta '--max-images=Número-de-imágenes' el número de imágenes a descargar.

Escribimos el siguiente comando para descargar las imágenes:

- python3 open_images_downloader.py --max-images=5000 --class-names "Person" --data=data/person

Con el dataset ya descargado es el momento de hacer el entrenamiento. Para hacer el mismo deberemos escribir el siguiente comando:

```
- python3 train_ssd.py --data=data/person --model-dir=models/person --batch-size=4  
--workers=2 --epochs=30
```

Explicamos que hace cada etiqueta:

‘--data=’ → Ruta donde se encuentra el dataset con el que entrenar al modelo

‘--model-dir=’ → Ruta donde guardar el modelo entrenado. Se recomienda almacenarlo en una ruta propia dentro de la carpeta models.

‘--batch-size=’ → Cantidad de memoria RAM que se usará durante el proceso. El valor por defecto es 4. En caso de que el proceso de entrenamiento sea detenido, se recomienda reducir este valor o usar memoria SWAP.

‘--workers=’ → Número de hilos para la carga de datos (0 = desactiva el multihilo). El valor por defecto es 2.

‘--epochs=’ → Número de épocas o veces que la red recorre el dataset. Hasta 100 épocas es algo aconsejable, pero a partir de ahí, el tiempo de entrenamiento incrementará de forma considerable. El valor por defecto son 30 épocas.

Si deseáramos parar el entrenamiento lo podemos hacer pulsando las teclas Ctrl+C. En caso de que quisiéramos reanudar el proceso de entrenamiento lo podríamos hacer de la siguiente forma. El script almacena un punto de restauración cada vez que completa una época. Para reanudar el entrenamiento deberemos añadir la etiqueta ‘--resume=’ seguido del punto de guardado. Ejemplo:

```
- python3 train_ssd.py --data=data/person --model-dir=models/person --batch-size=4  
--workers=2 --epochs=30 --resume=models/person/mb1-ssd-Epoch-3-Loss-5.245234623.pth
```

El siguiente paso es convertir nuestro modelo de Pytorch a ONNX. Para que podemos cargarlo con TensorTR. Para hacer esto escribiremos el siguiente comando:

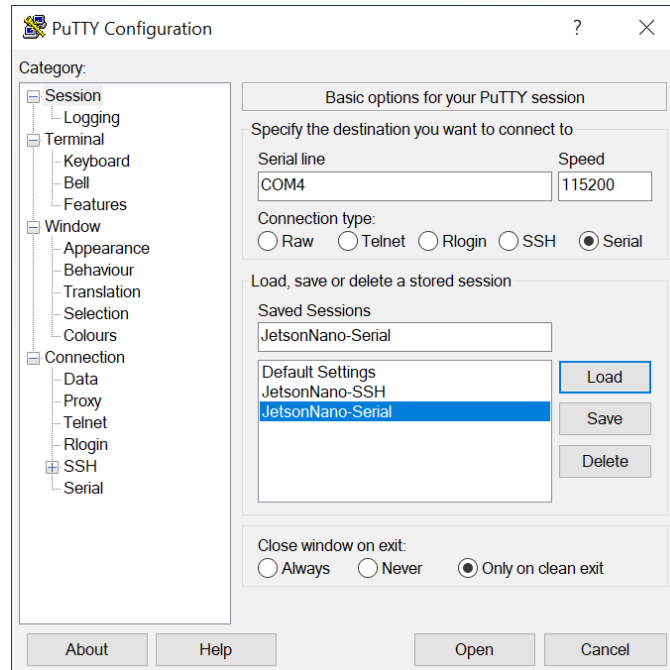
```
- python3 onnx_export.py --model-dir=models/person
```

Esto guardará un modelo llamado ‘ssd-mobilenet.onnx’ en la carpeta models/person a partir del modelo entrenado previamente en la misma. Ahora podremos utilizar desde nuestro programa el modelo creado.

7. Demostración de ejecución

Este apartado es una explicación sobre como ejecutar el programa previamente creado y comentado en la memoria, en caso de que se quiera hacer una prueba de demostración.

Primero nos conectaremos a la Jetson mediante Putty y el puerto serial. Especificaremos el puerto COM y la velocidad (para más información leer el punto 2).



Seguidamente introducimos nuestro usuario y contraseña. A continuación, ejecutamos el programa que se encuentra en la ruta `~/my-detection/my-detection.py` con el comando:

```
- python3 ~/my-detection/my-detection.py
```

Una vez hecho esto, deberemos habilitar la retransmisión de la cámara desde el PC remoto. Para ello podemos usar Gstreamer o VLC. Para usar Gstreamer abriremos la consola de Windows y escribiremos el siguiente comando:

```
- gst-launch-1.0 -v udpsrc port=1234 caps = "application/x-rtp, media=(string)video, clock-
rate=(int)90000, encoding-name=(string)H264, payload=(int)96" ! rtph264depay !
decodebin ! videoconvert ! Autovideosink
```

Tras pulsar intro podremos ver la retransmisión de la cámara.

En el caso de usar VLC crearemos un archivo plano de texto con la extensión `'.sdp'` con el siguiente contenido:

```
c=IN IP4 192.168.55.100
m=video 1234 RTP/AVP 96
a=rtpmap:96 H264/90000
```

Una vez guardado solo tenemos que abrir el archivo para ver la imagen de la cámara.

8. Bibliografía

- Librería de Nvidia Jetson-inference: <https://github.com/dusty-nv/jetson-inference>
- Playlist de tutoriales de Nvidia: <https://youtube.com/playlist?list=PL5B692fm6--uQRRDTPsJDp4o0xbzkoyf8>
- Playlist de tutoriales de Inteligencia artificial en la Jetson Nano por Paul McWhorter: <https://youtube.com/playlist?list=PLGs0VKk2DiYxP-ElZ7-QXIERFFPkOuP4>
- Repositorio de Drivers para Jetson-GPIO: <https://github.com/NVIDIA/jetson-gpio>