**TECNOLÓGICO NACIONAL DE MÉXICO**

**INSTITUTO CAMPUS IZTAPALAPA**

**INGENIERÍA MECATRÓNICA**

**Programación Avanzada**

**M7M**

**Interfaz del proyecto**

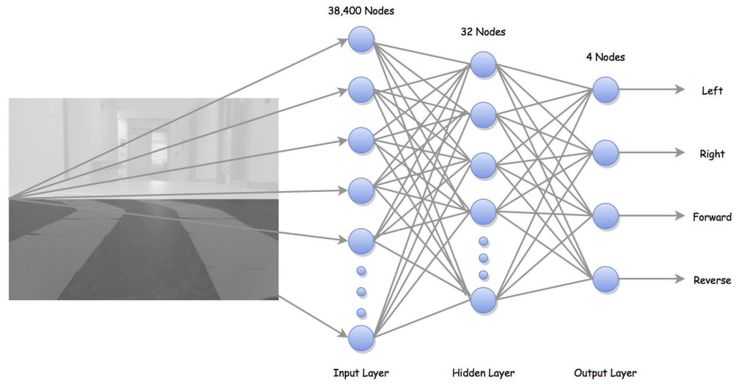
**MOLINA RODRIGUEZ JAEL**

**Profesor:**

**Soria frías Sigfrido**

Interfaz del proyecto

Coche RC autónomo

El cerebro del coche autónomo (la inteligencia artificial IA) se basa en parte en redes neuronales que procesan las imágenes y datos de sensores disponibles (bien sean datos de [radar](https://blog.nxp.com/automotive/radar-camera-and-lidar-for-autonomous-cars), datos obtenidos mediante láser ([LIDAR](https://www.nytimes.com/2017/05/25/automobiles/wheels/lidar-self-driving-cars.html)), imágenes, otros sensores instalados en el vehículo…). Con esos datos de entrada y el entrenamiento correspondiente la red neuronal puede obtener una salida en cada momento y situación (por ejemplo, acelerar, frenar, girar… según su posición y el entorno en el que se encuentre el vehículo). Se usara una librería para llevar acabo el procesamiento de imágenes por medio de la cámara utilizando OpenCV o las librerías para así generar las redes neuronales como Tensorflow.

Descargando las librería que se ocuparían para este proyecto la raspberry tomara las imágenes al ordenador por wifi así procesando estas por medio de las redes neuronales y así obtener diferentes salidas, esta podría ser la forma en la que pretendemos controlar el vehículo, ya que es muy interesante también poder controlar de forma manual desde el ordenador el coche teledirigido, como si de un rover en suelo marciano se tratara, controlándolo desde lugares remotos.

En este caso, es la propia Raspberry Pi la que implementa la red neuronal mediante el uso de [OpenCV](http://opencv.org/) (procesado de imágenes y visión por computador) y [Tensorflow](https://www.tensorflow.org/) (red neuronal con [Machine Learning](https://en.wikipedia.org/wiki/Machine_learning) para detectar señales, semáforos, etcétera). Así, directamente podríamos instalar estas librerías (todas ellas opensource*,* código abierto que todo el mundo puede utilizar, modificar, mejorar y compartir) para que sea el propio coche el que conduzca, sin necesidad de ordenador externo (aunque lo seguiremos utilizando para ver qué está viendo y haciendo la red neuronal y dar los comandos de arrancar el vehículo y apagarlo).

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamenteSeguido de esto se configurara la cámara con la raspberry para así poder obtener las imágenes que se tienen que procesar se Accedera a la Raspberry Pi mediante SSH (ver post [Coche RC autónomo (III)](https://jorgecasas.github.io/2017/09/12/autonomous-rc-car-iii)). Tras [conectar la cámara a la Raspberry Pi](https://www.raspberrypi.org/documentation/usage/camera/), tendremos que activar el módulo de la cámara en el menú correspondiente de la aplicación de configuración.

Para grabar vídeos de forma manual y ver que todo funciona, podemos ejecutar el siguiente comando

Interfaz de usuario gráfica, Texto, Correo electrónico

Descripción generada automáticamente

Con la librería OpenCV se podrá usar la cámara y Python la visión por computador para así poder comentar los primeros scripts utilizando la librería picamera y openCV para procesar, ya que así conseguimos enviar las imágenes desde el cliente al servidor, pero para lograrlo necesitaremos netcat y así obtener objetos de tipo imagen para que puedan ser procesadas.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamenteSe va a crear los dos primerosscripts python*,* los cuales nos servirán de base para ir programando más adelante todo lo necesario para que el vehículo autónomo puedaver y procesar lo que ve Para ejecutar los scripts basta con ejecutar los comandos.

Para copiar los ficheros tanto a la Raspberry Pi como a nuestro ordenador, puede ser interesante crear un repositor, tanto en la Raspberry Pi como en nuestro ordenador, de manera que podamos compartir nuestro código y actualizarlo fácilmente en los diferentes dispositivos. Tendremos que instalar git. Y gracias a esto clonaremos el código para poder crear ramas para nuestro desarrollo, seguido de esto gracias a git podremos actualizar nuestro código y descargarlo mediante la raspberry.

En el servidor incluiremos el siguiente script stream\_server\_computer.py. Su función es la siguiente:

* Importamos las librerías (como OpenCV cv2 para procesar las imágenes recibidas, socketserver para poder crear un servidor de sockets…)
* Indicamos variables de configuración (como el puerto que va a utilizar el servidor, en nuestro caso el 8000)
* Creamos un hilo de ejecución (thread), que creará un servidor de socket, esperando hasta recibir la primera petición de un cliente (de la Raspberry Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

  Descripción generada automáticamentePi). Creamos un hilo porque en el futuro necesitaremos crear varios sockets en paralelo (uno para recibir imágenes, otro para recibir datos de distancia del sensor ultrasónico, etcétera).
* En cuanto recibamos la primera conexión, iremos leyendo el buffer de bits que nos vaya enviando el cliente (la Raspberry Pi). Este cliente irá enviando imagen a imagen (frames), que serán obtenidos como objetos de imagen de OpenCV.
* Al obtener un objeto de imagen OpenCV, lo procesaremos decodificando los datos mediante cv2.imdecode() (por ejemplo, lo podemos convertir a escala de grises), y lo mostraremos en una ventana mediante cv2.imshow()
* Y así hasta que pulsemos sobre la ventana la tecla q, que finalizará la ejecución.

Es importante tener en cuenta que tendremos que **ejecutar este script servidor antes que el script cliente**. Para ello, como hemos comentado, basta con ejecutar desde el entorno virtualizado python stream\_server\_computer.py.

En el cliente incluiremos el siguiente script stream\_client\_raspberry.py. Su cometido es el siguiente:

* Texto

  Descripción generada automáticamenteImportar librerías (como socket para crear un cliente de sockets, picamera para poder obtener imágenes de la cámara de la Raspberry Pi, etc…)
* Indicar parámetros de configuración (IP del servidor, puerto del servidor, dimensiones de la imagen a capturar, frames por segundo, duración máxima de envío…)
* Abrimos un socket a la dirección IP y puerto del servidor.
* Inicializamos la cámara de la Raspberry Pi, dándole 2 segundos para inicializarse
* Enviamos imágenes en forma de stream JPEG al servidor hasta que pase el tiempo máximo recording\_time o se cierre la comunicación (pulsando Ctrl + C, etcétera)

El código del cliente es el siguiente Lo ejecutaremos mediante python stream\_client\_raspberry.py.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamenteEl coche necesitara sensores que obtendrán datos que podrán ser utilizados para garantizar la seguridad e integridad de los vehículos y todo lo que les rodea. Así, en este post voy a detallar cómo utilizar el [sensor de ultrasonidos HC-SR04](http://amzn.to/2yC24UO) para detectar obstáculos delante del vehículo. El mismo script y circuito que vamos a utilizar puede servir como sensor de aparcamiento, ya que nos permitirá detectar a qué distancia nos encontramos de un obstáculo o pared. También puede ser utilizado para medir cuán lleno está un pozo o un bidón de almacenamiento de líquidos.

En nuestro diseño definiremos una distancia límite. Si el vehículo se encuentra más cerca de un obstáculo que dicha distancia, se detendrá para no chocar. En las pruebas que hemos realizado vamos a encender un led cuando estemos a menos distancia y mostraremos un mensaje de alerta. De esta forma, y con un pequeño circuito, aprenderemos también a utilizar las entradas y salidas **GPIO** de la Raspberry Pi. En primer lugar, accedemos a la Raspberry Pi y accedemos al entorno virtual de Python, instalando con pip la librería requerida para poder gestionar las entradas y salidas GPIO. Siguiendo el script, tenemos que crear un circuito teniendo en cuenta la distribución de los pines GPIO de la Raspberry Conectaremos pin VCC+ al pin VCC y el pin GROUND al pin GROUND del sensor HC-SR04, para alimentarlo.

Conectaremos el pin de salida GPIO 18 de la Raspberry Pi directamente al pin TRIGGER del sensor HC-SR04. Conectaremos el pin de entrada GPIO 24 de la Raspberry Pi al pin ECHO del sensor HC-SR04, pero no lo haremos directamente sino mediante una resistencia de 1K. Adicionalmente, conectaremos un led que encenderemos cuando el obstáculo se encuentre más cerca de la distancia límite:

Conectaremos el pin de salida GPIO 12 de la Raspberry Pi directamente al pin positivo del led, conectaremos una resistencia de 1K del otro pin del led al pin GROUND de la Raspberry Pi y sólo queda probarlo ejecutando el siguiente comando. El led deberá encenderse y apagarse según acerquemos objetos al sensor. Este código lo adaptaremos en los próximos pasos dentro de nuestro script global, encapsulándolo en un hilo independiente, de manera que podamos utilizar la información obtenida a la hora de controlar el vehículo.

Al configurar el sensor ultrasónico enviaremos la información desde nuestro cliente ([Raspberry Pi](http://amzn.to/2yjVkh6)) a nuestro servidor, para luego poder procesar esos datos de distancia a objetos junto con las imágenes de las videocámaras La idea es, por tanto, actualizar nuestros scripts Python 3 server.py (a ejecutar en nuestro ordenador durante el desarrollo, y luego exclusivamente desde la Raspberry Pi 3 cuando esté montada en el vehículo teledirigido) y client.py (a ejecutar en la Raspberry Pi 3 siempre). A tener en cuenta:

* Vamos a crear en el cliente un thread (hilo de ejecución) que lea datos del sensor ultrasónico con la distancia a objetos enfrente de dicho sensor y la envíe al servidor por un puerto definido (el 8001 en nuestro caso), y otro thread que obtenga imágenes de la videocámara y las envíe mediante streaming a otro puerto definido (el 8000)
* En el servidor crearemos igualmente dos threads, uno para gestionar las imágenes recibidas de la cámara y otro para obtener los datos de distancia obtenidos del sensor ultrasónico.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamenteEn el servidor, el thread principal será el que procese imágenes de la videocámara, ya que desde él podremos leer los datos recibidos del sensor ultrasónico y podremos mostrar mensajes en la propia imagen que visualicemos. Desde este thread, como veremos en futuros posts, iremos llamando a diferentes funciones para controlar el vehículo (redes neuronales que procesen las imágenes, y lógica de si el coche debe girar, frenar, acelerar, etcétera)

A la vez que recibimos dichos datos (imágenes y distancia a obstáculos) vamos a ir preparando esta lógica: Vamos a mostrar en la imagen si tenemos obstáculo delante o no, escribiendo un texto.

* El código del cliente (Raspberry Pi 3) es el siguiente (client.py).

Texto

Descripción generada automáticamenteEl código del servidor es el siguiente (server.py)

En ambos scripts he ido incluendo variables para facilitar la configuración usada, como puede ser la activación / desactivación de logs, dimensiones de imágenes, etcétera, por lo que es recomendable revisar dicha configuración cuando estéis haciendo pruebas. Ahora sólo queda probarlo, arrancando primero el servidor y luego el cliente (accediendo previamente al entorno virtualizado python mediante workon cv):

* En el servidor: python server.py
* En el cliente: python client.py

El resultado será similar al de la imagen que acompaña este post. En próximos capítulos empezaremos a detectar señales de STOP mediante redes neuronales.

con todas las librerías necesarias para la visión por computador usando OpenCV, sabemos utilizar un sensor (el de ultrasonidos) para detectar si hay un obstáculo delante nuestro y a qué distancia, y sabemos enviar las imágenes y estos datos desde la Raspberry Pi a nuestro ordenador.

Los siguientes pasos son:

* Aprender a procesar las imágenes para detectar señales de STOP y semáforos
* Aprender a calibrar las imágenes recibidas para saber a qué distancia aproximada se encuentran esas señales de STOP y los semáforos
* Aprender a crear una red neuronal que procese las imágenes para saber cuando tendremos que acelerar, frenar, girar a la izquierda o a la derecha
* Aprender a controlar servos con la Raspberry Pi para que el vehículo acelere, frene, gire a izquierda o derecha cuando corresponda.

En este post vamos a aprender cómo detectar señales de STOP

Para ello, vamos a utilizar clasificadores en cascada que ofrece la propia librería OpenCV. Eso sí, necesitamos ficheros descriptores de los elementos que tenemos que detectar para que los clasificadores puedan detectar si en las imágenes aparecen o no esos objetos y la posición en la que aparecen. En nuestro caso, señales de STOP y semáforos, pero necesitaríamos descriptores para cada tipo de objeto adicional (cualquier otro tipo de señal, balizamiento, peatones, bicicletas, maceteros y bolardos…). Estos ficheros descriptores los podemos crear nosotros mismos, utilizando un montón de imágenes positivas (donde aparezca el objeto a detectar en multitud de posiciones, condiciones ambientales, etcétera) y un montón de imágenes negativas (donde no aparezca el objeto, pudiendo aparecer otros objetos similares diferentes). En muchos blogs indican que se necesitan como poco unas 60 imágenes positivas para la señal de STOP, y unas 600 para las imágenes negativas, pero cuantas más tengamos de cada una de ellas mejor.

Las imágenes positivas (las de la señal de STOP que queremos detectar) tienen que ser recortadas para que aparezca únicamente el objeto. Las imágenes negativas pueden ser de cualquier cosa, pero si vamos a crear un sistema de conducción lo suyo sería usar fotografías de situaciones comunes que nos vayan a surgir: Fotografías de carreteras, con otras señales diferentes, etcétera. En este punto, como es muy aburrido conseguir un banco de fotos de unas 1000 fotografías, podemos exportar de una película MP4 todos los fotogramas. Por ejemplo, si tenemos ffmpeg instalado podemos ejecutar el siguiente comando para obtener de un vídeo INPUT.mp4 grabado con el móvil 5 imágenes cada segundo, con nombre image-000001.jpg (incrementándose el número de imagen de forma consecutiva). Así, si el vídeo dura 100 segundos, conseguiremos 500 imágenes en un momento etc.