

Simulador CARLA

Car Learning Approach

by. Rafael Peralta Blanco

¿Qué es CARLA?

“Es un simulador para desarrollar,
entrenar y validar sistemas de
conducción autónoma.”

<http://carla.org/>

Unreal Engine 4

El simulador fue desarrollado en Unreal Engine 4.

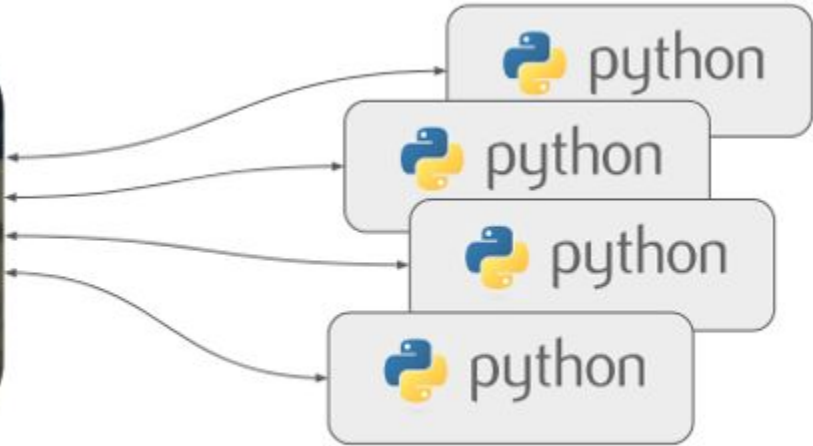


Arquitectura

El sistema de CARLA tiene una arquitectura cliente-servidor.



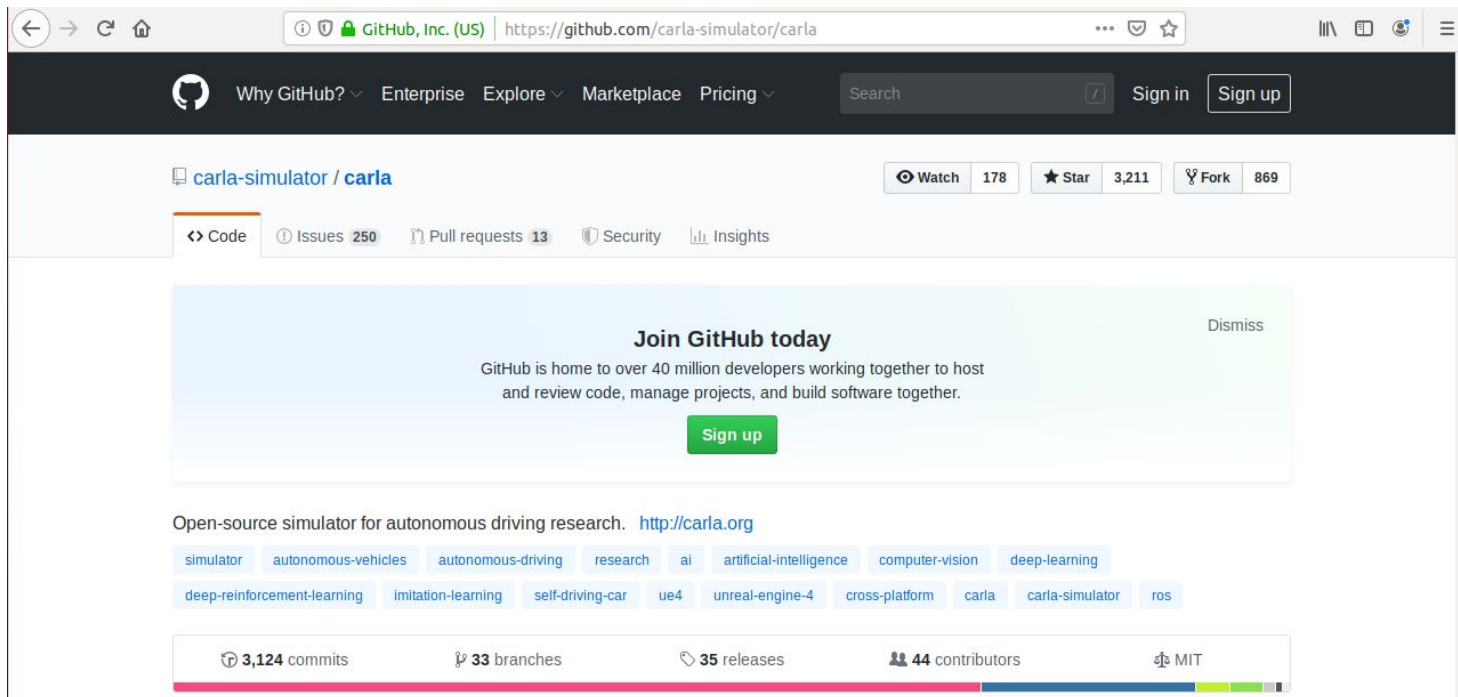
Simulator



User scripts

1. Ir al repositorio

<https://github.com/carla-simulator/carla>



2. Seleccionar releases

📄 3,124 commits

🌿 33 branches

🏷️ 35 releases

👤 44 contributors

🏛️ MIT

3. Descargar la versión compilada

CARLA_0.9.6.tar.gz



The screenshot shows the GitHub releases page for the CARLA project. At the top, there are two tabs: 'Releases' (active) and 'Tags'. Below the tabs, on the left, is a sidebar with a green box labeled 'Latest release', a tag icon followed by '0.9.6', and a commit hash 'e2c4dc1'. The main content area has a large blue heading 'CARLA 0.9.6 (development)'. Below this, it says 'nsubiron released this on Jul 12 · 24 commits to master since this release'. Under the heading 'Compiled version', there is a list of three items: '[Linux] CARLA_0.9.6.tar.gz', '[Linux] Town06_0.9.6.tar.gz', and '[Linux] Town07_0.9.6.tar.gz'. The first item, 'CARLA_0.9.6.tar.gz', is circled in red.

Releases Tags

Latest release

0.9.6

e2c4dc1

CARLA 0.9.6 (development)

nsubiron released this on Jul 12 · 24 commits to master since this release

Compiled version

- [Linux] [CARLA_0.9.6.tar.gz](#)
- [Linux] [Town06_0.9.6.tar.gz](#)
- [Linux] [Town07_0.9.6.tar.gz](#)

4. Descomprimir el archivo

4.1 Click derecho.

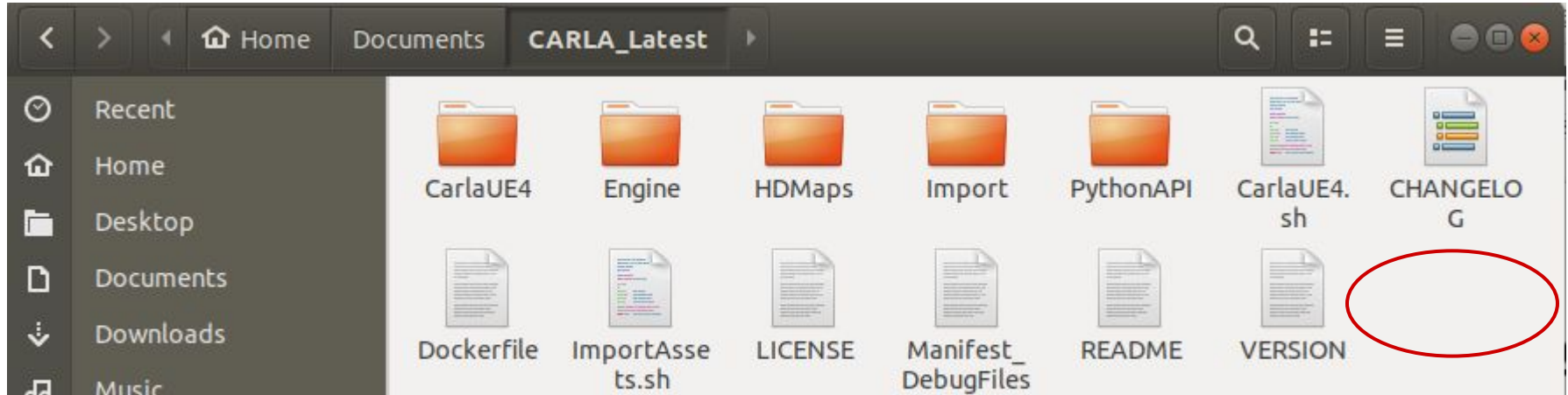
4.2 Seleccionar extraer aquí.



5. Entrar en la carpeta CARLA

5.1 Click derecho en alguna zona libre

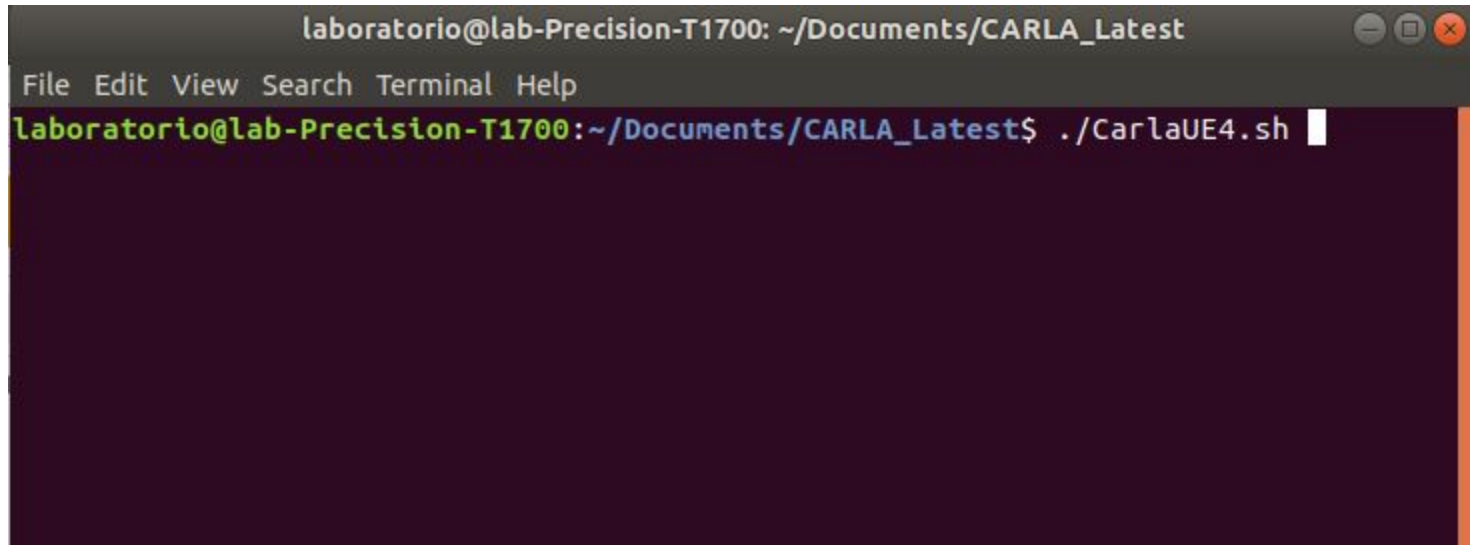
5.2 Seleccionar 'abrir en terminal'



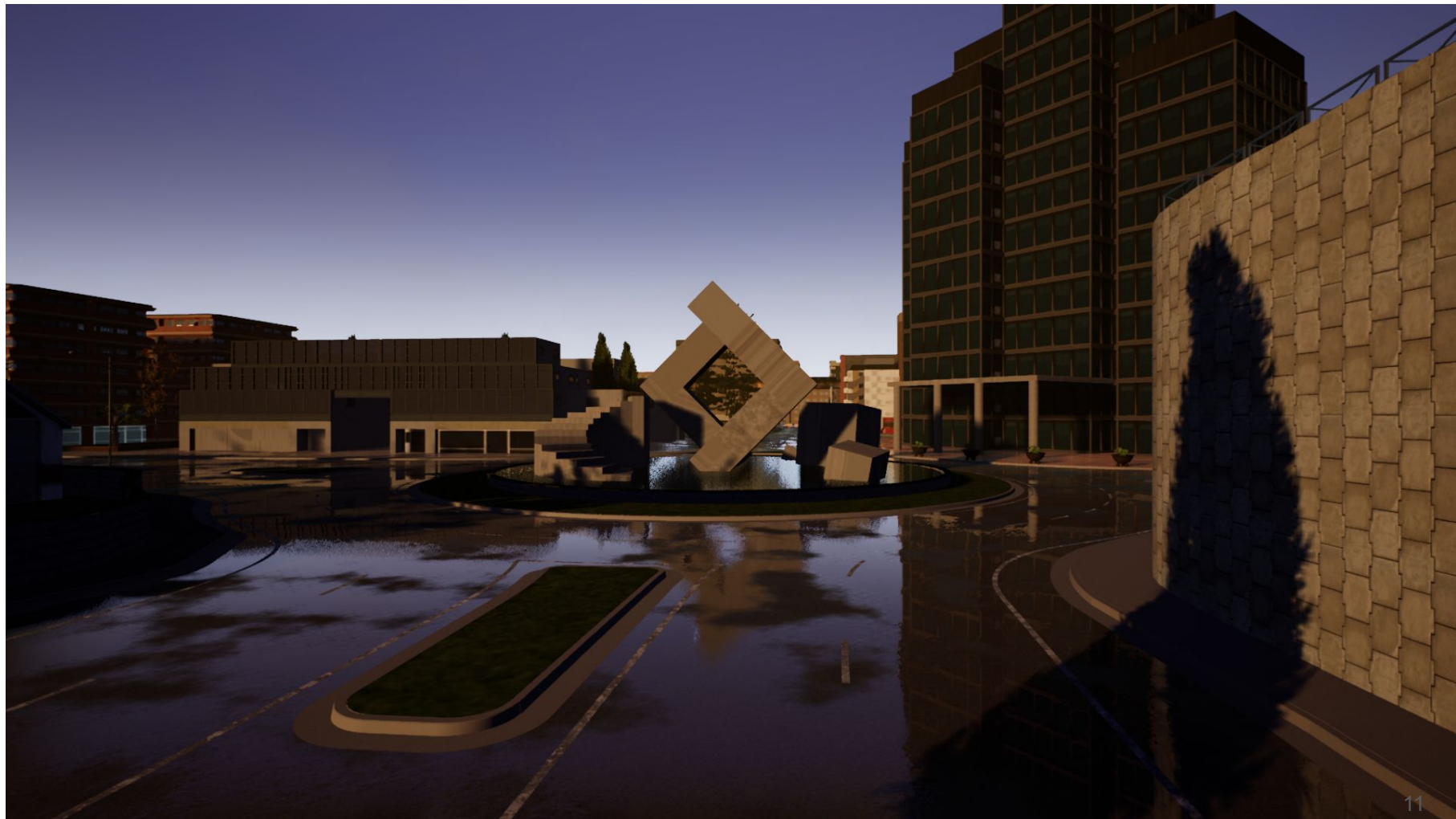
6. Ejecutar el simulador

6.1 Escribir el comando './CarlaUE4.sh' en la terminal

6.2 Presionar 'Enter'



```
laboratorio@lab-Precision-T1700: ~/Documents/CARLA_Latest
File Edit View Search Terminal Help
laboratorio@lab-Precision-T1700:~/Documents/CARLA_Latest$ ./CarlaUE4.sh
```

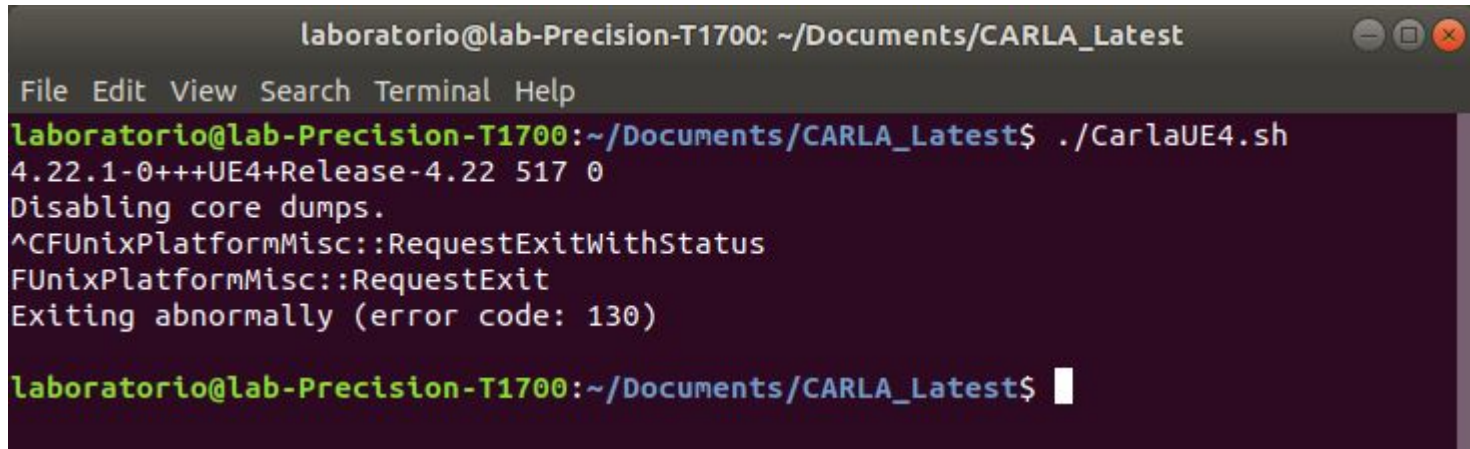


7. Cerrar el simulador

7.1 Presionar las teclas 'Ctrl' y 'Tab' al mismo tiempo.

7.2 Seleccionar la ventana de la terminal

7.3 Presionar las teclas 'Ctrl' y 'C' al mismo tiempo



```
laboratorio@lab-Precision-T1700: ~/Documents/CARLA_Latest
File Edit View Search Terminal Help
laboratorio@lab-Precision-T1700:~/Documents/CARLA_Latest$ ./CarlaUE4.sh
4.22.1-0+++UE4+Release-4.22 517 0
Disabling core dumps.
^CFUnixPlatformMisc::RequestExitWithStatus
FUnixPlatformMisc::RequestExit
Exiting abnormally (error code: 130)

laboratorio@lab-Precision-T1700:~/Documents/CARLA_Latest$
```

Mejorando el rendimiento

1. Instalar Vulkan

1.1 Ejecutar los siguientes comandos en una terminal (Ctrl+Alt+T).

```
sudo add-apt-repository ppa:graphics-drivers/ppa
```

```
sudo apt update
```

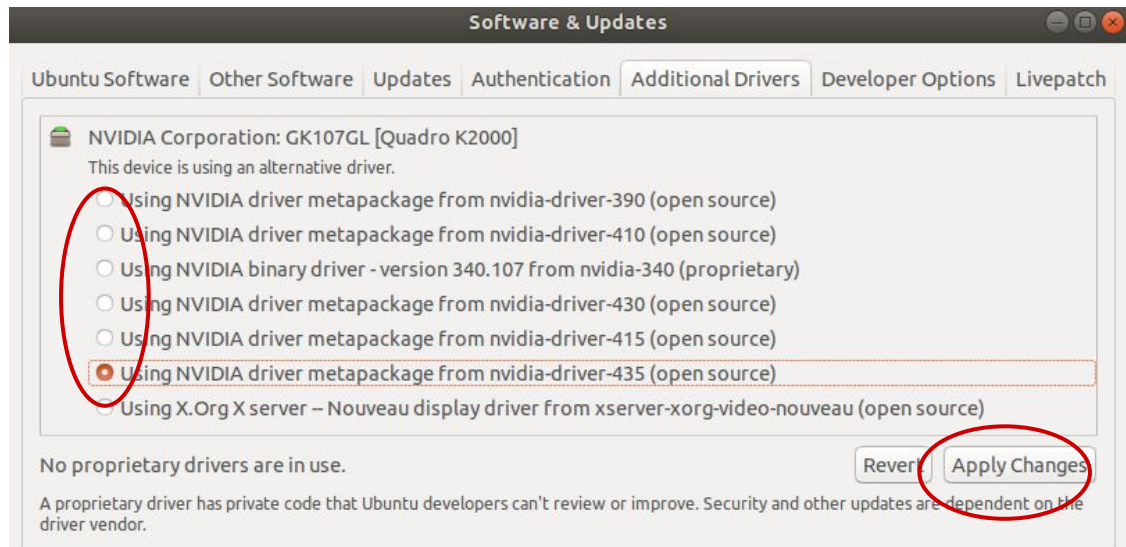
```
sudo apt upgrade
```

1. Instalar Vulkan

1.2 Abrir la aplicación 'Software & Actualizaciones'

1.3 Seleccionar la pestaña 'Controladores adicionales'

1.4 Seleccionar el driver adecuado para la tarjeta de video y luego aplicar cambios



Hay que tener cuidado, porque puede que el driver no sea compatible.

1. Instalar Vulkan

1.5 Volver a la terminal y ejecutar el siguiente comando

```
sudo apt install nvidia-settings vulkan-utils
```

*Suponiendo que se tiene una gpu Nvidia

2. No renderizar el simulador

Usar el siguiente comando cuando se quiera ejecutar el simulador.

```
DISPLAY= ./CarlaUE4.sh
```

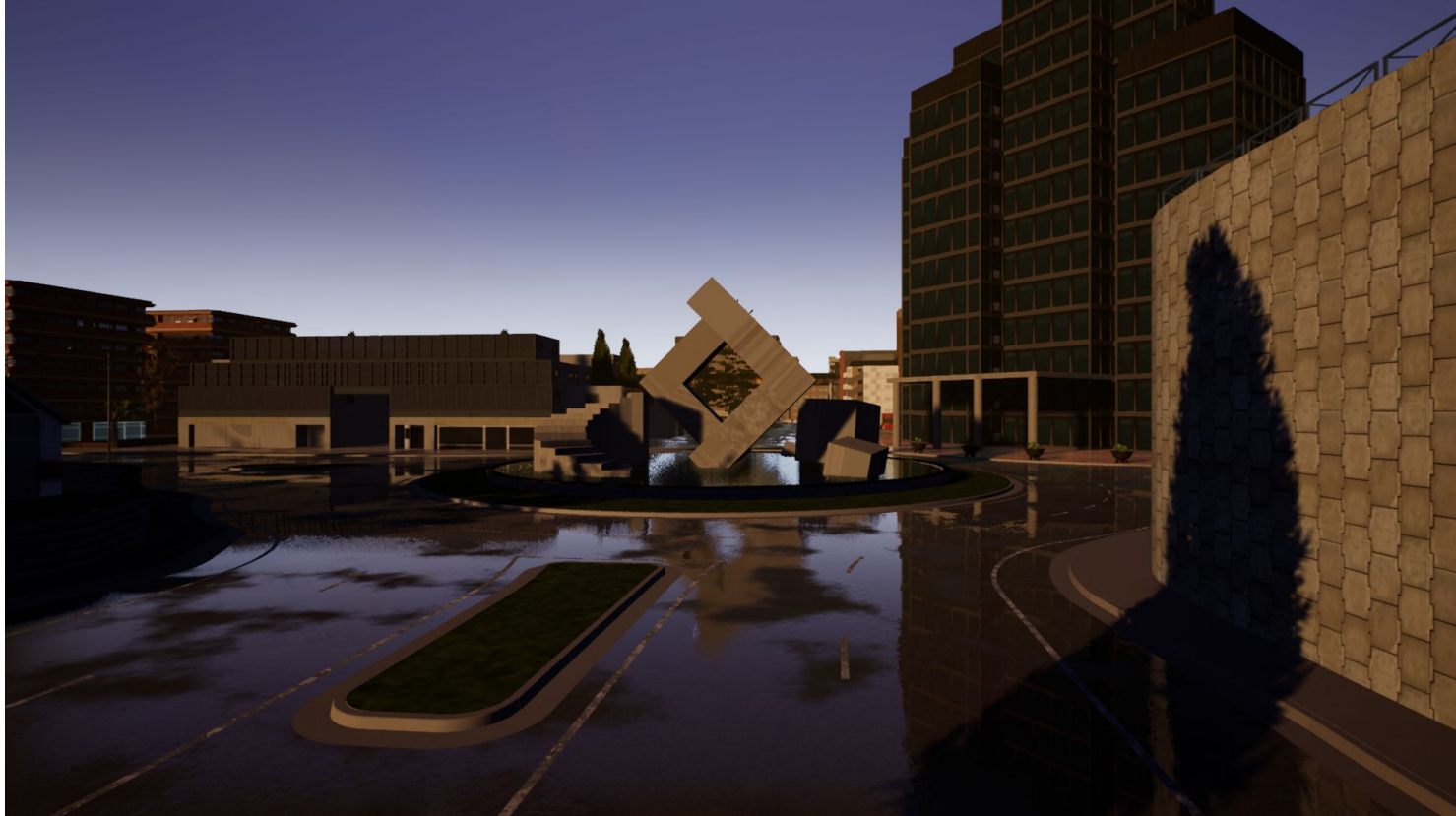
3. Reducir el tamaño de la ventana

Usar el siguiente comando cuando se quiera ejecutar el simulador con un tamaño distinto.

```
./CarlaUE4.sh -windowed -ResX=720 -ResY=480
```

Utilizando la API de Python

El servidor debe estar en ejecución



1. Script base

1.1 Hay que importar la librería de CARLA. Para poder manipular el servidor.

```
import sys  
egg_path='lib/dist/carla-0.9.6-py3.5-linux-x86_64.egg'  
sys.path.append(egg_path)  
import carla
```



NO
CODE
TIME

2. Conexión al servidor

Crear el archivo 'CarlaEnv.py' en la carpeta curso.

2. Conexión al servidor

2.1 Recuerda importar la librería.



```
import sys

egg_path='lib/dist/carla-0.9.6-py3.6-linux-x86_64.egg'
sys.path.append(egg_path)

import carla
```

2. Conexión al servidor

Se usa el paradigma de programación orientado a objetos para tener un código más limpio.



NO
CODE
TIME

```
class CarlaEnv:

    def __init__(self, host="127.0.0.1", port=2000):

        self.client = carla.Client(host, port)
        # Tiempo de espera sin respuesta del servidor.
        self.client.set_timeout(2.0)

        # Obtiene el mundo de CARLA
        self.world = self.client.get_world()
        # Mapa del mundo
        self.map = self.world.get_map()
        # Tiene todas los objetos utilizables en CARLA
        self.blueprint_library = self.world.get_blueprint_library()

        self.actors = []
```


2. Conexión al servidor

La variable 'self.client' contiene la conexión al servidor. Se requiere conocer la dirección ip del servidor y el puerto en el que se está ejecutando CARLA.



NO
CODE
TIME

```
class CarlaEnv:

    def __init__(self, host="127.0.0.1", port=2000):
        self.client = carla.Client(host, port)
        # Tiempo de espera sin respuesta del servidor.
        self.client.set_timeout(2.0)

        # Obtiene el mundo de CARLA
        self.world = self.client.get_world()
        # Mapa del mundo
        self.map = self.world.get_map()
        # Tiene todas los objetos utilizables en CARLA
        self.blueprint_library = self.world.get_blueprint_library()

        self.actors = []
```

Por defecto
CARLA utiliza el
puerto 2000.

2. Conexión al servidor

Esta línea evita que la red se bloquee para siempre.



NO
CODE
TIME

```
class CarlaEnv:

    def __init__(self, host="127.0.0.1", port=2000):

        self.client = carla.Client(host, port)
        # Tiempo de espera sin respuesta del servidor.
        self.client.set_timeout(2.0)

        # Obtiene el mundo de CARLA
        self.world = self.client.get_world()
        # Mapa del mundo
        self.map = self.world.get_map()
        # Tiene todas los objetos utilizables en CARLA
        self.blueprint_library = self.world.get_blueprint_library()

        self.actors = []
```

2. Conexión al servidor

Tiene el mapa del mundo actual. Este mapa tiene el formato OpenDrive.



NO
CODE
TIME

```
class CarlaEnv:

    def __init__(self, host="127.0.0.1", port=2000):

        self.client = carla.Client(host, port)
        # Tiempo de espera sin respuesta del servidor.
        self.client.set_timeout(2.0)

        # Obtiene el mundo de CARLA
        self.world = self.client.get_world()
        # Mapa del mundo
        self.map = self.world.get_map()
        # Tiene todas los objetos utilizables en CARLA
        self.blueprint_library = self.world.get_blueprint_library()

        self.actors = []
```

2. Conexión al servidor

La variable 'self.blueprint_library' se encuentran todos los posibles actores que se pueden invocar en el simulador.



NO
CODE
TIME

```
class CarlaEnv:

    def __init__(self, host="127.0.0.1", port=2000):

        self.client = carla.Client(host, port)
        # Tiempo de espera sin respuesta del servidor.
        self.client.set_timeout(2.0)

        # Obtiene el mundo de CARLA
        self.world = self.client.get_world()
        # Mapa del mundo
        self.map = self.world.get_map()
        # Tiene todas los objetos utilizables en CARLA
        self.blueprint_library = self.world.get_blueprint_library()

        self.actors = []
```

Aquí podemos encontrar diferentes modelos de automóviles.

2. Conexión al servidor

Indentación

La variable 'self.actors' guarda todos los actores invocados, por este cliente, en el mundo.

CODE
TIME

```
class CarlaEnv:
    def __init__(self, host="127.0.0.1", port=2000):
        self.client = carla.Client(host, port)
        # Tiempo de espera sin respuesta del servidor.
        self.client.set_timeout(2.0)

        # Obtiene el mundo de CARLA
        self.world = self.client.get_world()
        # Mapa del mundo
        self.map = self.world.get_map()
        # Tiene todas los objetos utilizables en CARLA
        self.blueprint_library = self.world.get_blueprint_library()

        self.actors = []
```

La necesitamos
para limpiar el
simulador.

2. Conexión al servidor

Cada que queramos invocar un actor necesitamos:

- El modelo, es decir, su blueprint.
- La posición en la que queremos invocarlo.



CODE
TIME

```
def spawn_actor(self, model, spawn_point, attach_to=None):  
    # Invoca un actor en el mundo  
    actor = self.world.spawn_actor(model, spawn_point, attach_to)  
    self.actors.append(actor)  
  
    return actor
```

■ ■ Indentación

```
import carla  
  
class CarlaEnv:  
    ■ ■ def __init__(self, host="127.0.0.1", port=2000):  
        ■ ■ self.client = carla.Client(host, port)  
            # Tiempo de espera sin respuesta del servidor  
            self.client.set_timeout(2.0)  
  
            # Obtiene el mundo de CARLA  
            self.world = self.client.get_world()  
            # Mapa del mundo  
            self.map = self.world.get_map()  
            # Tiene todas los objetos utilizables en CARLA  
            self.blueprint_library = self.world.get_blueprint_library()  
  
            self.actors = []  
  
    ■ ■ def spawn_actor(self, model, spawn_point, attach_to=None):  
        # Invoca un actor en el mundo  
        actor = self.world.spawn_actor(model, spawn_point, attach_to)  
        self.actors.append(actor)
```

Guardamos el actor
invocado en la lista
de actores.

2. Conexión al servidor

Indentación

Una vez que terminemos con la simulación hay que limpiar el servidor.

```
def destroy(self):  
    # Cuando el cliente termine de trabajar, hay que eliminar todos  
    # los actores que ha invocado  
    while len(self.actors) != 0:  
        a = self.actors.pop()  
        a.destroy()
```

CODE
TIME

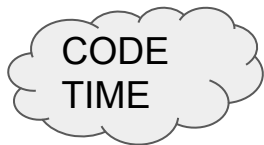
```
self.client.set_timeout(2.0)  
  
# Obtiene el mundo de CARLA  
self.world = self.client.get_world()  
# Mapa del mundo  
self.map = self.world.get_map()  
# Tiene todas los objetos utilizables en CARLA  
self.blueprint_library = self.world.get_blueprint_library()  
  
self.actors = []  
  
def spawn_actor(self, model, spawn_point, attach_to):  
    # Invoca un actor en el mundo  
    actor = self.world.spawn_actor(model, spawn_point, attach_to=attach_to)  
    self.actors.append(actor)  
  
    return actor  
  
def destroy(self):  
    # Cuando el cliente termine de trabajar, hay que  
    # los actores que ha invocado  
    while len(self.actors) != 0:
```

2. Conexión al servidor

Ya podemos conectarnos al servidor.

Creamos el archivo 'main.py' en la carpeta 'curso'.

Ahí importamos la clase 'CarlaEnv'



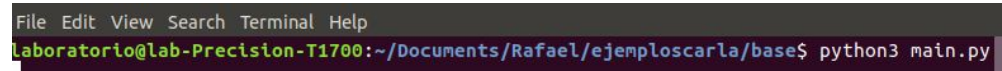
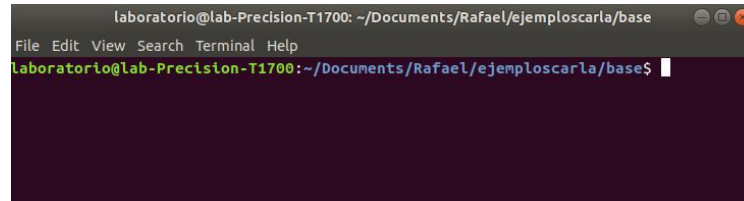
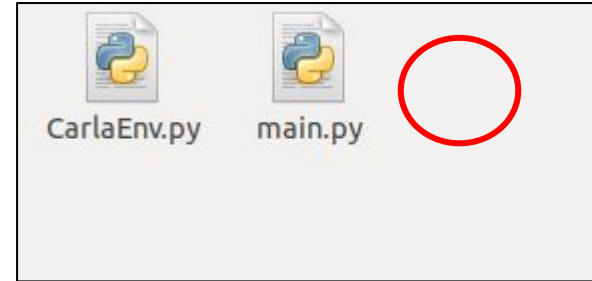
```
from CarlaEnv import *  
|  
env = CarlaEnv()
```

Listo, la conexión se encuentra en la variable 'env'.

Para ejecutar el programa...



1. Ir a la carpeta 'curso' con el administrador de archivos.
2. Click derecho en algún lugar libre.
3. Seleccionar la opción 'Open in Terminal'.
4. Escribir: `python3 main.py`
5. Y presionar la tecla 'Enter'.



Invocar un vehículo

3. Invocar un automóvil

Creamos el archivo 'Car.py' en la carpeta 'curso'.

No necesitamos
importar carla.

3. Invocar un automóvil

Mantenemos el paradigma de programación orientado a objetos.

```
class Car:

    def __init__(self, env, model, spawn_point, camera_config=None):
        # Invoca el vehiculo en el mundo
        self.vehicle = env.spawn_actor(model, spawn_point)
```



NO
CODE
TIME

3. Invocar un automóvil

En el parámetro 'env' tenemos la conexión al servidor y los recursos de CARLA.

```
class Car:
    ↓
    def __init__(self, env, model, spawn_point, camera_config=None):
        # Invoca el vehiculo en el mundo
        self.vehicle = env.spawn_actor(model, spawn_point)
```

NO
CODE
TIME

3. Invocar un automóvil

Con anterioridad necesitamos conocer el modelo del vehículo que deseamos invocar.

```
class Car:
    ↓
    def __init__(self, env, model, spawn_point, camera_config=None):
        # Invoca el vehiculo en el mundo
        self.vehicle = env.spawn_actor(model, spawn_point)
```


NO
CODE
TIME

Para eso
necesitamos la
variable
'blueprint_library'

3. Invocar un automóvil

También necesitamos saber en qué posición en el mapa vamos a invocar el vehículo.

```
class Car:
    def __init__(self, env, model, spawn_point, camera_config=None):
        # Invoca el vehiculo en el mundo
        self.vehicle = env.spawn_actor(model, spawn_point)
```




NO
CODE
TIME

Se obtiene de la
variable 'map'.

3. Invocar un automóvil

Luego agregaremos una cámara al vehículo...

```
class Car:
    def __init__(self, env, model, spawn_point, camera_config=None):
        # Invoca el vehiculo en el mundo
        self.vehicle = env.spawn_actor(model, spawn_point)
```



NO
CODE
TIME

Se debe tener la
clase 'Camera'.

3. Invocar un automóvil

En la variable 'self.vehicle' guardaremos el actor invocado en el mundo.

```
class Car:

    def __init__(self, env, model, spawn_point, camera_config=None):
        # Invoca el vehiculo en el mundo
        → self.vehicle = env.spawn_actor(model, spawn_point)
```

NO
CODE
TIME

La función 'spawn_actor'
está programada en el
archivo 'CarlaEnv.py'

3. Invocar un automóvil

```
class Car:

    def __init__(self, env, model, spawn_point, camera_config=None):
        # Invoca el vehiculo en el mundo
        self.vehicle = env.spawn_actor(model, spawn_point)
```

CODE
TIME

Indentación

3. Invocar un automóvil

¿Recuerdan el archivo 'main.py'?

Ahora hay que agregar la clase 'Car'.

```
from CarlaEnv import *  
|  
env = CarlaEnv()
```

```
from CarlaEnv import *  
from Car import *|  
  
env = CarlaEnv()
```



CODE
TIME

3. Invocar un automóvil

Ahora hay que obtener los parámetros necesarios para invocar el vehículo.

- conexión al servidor. ✓
- modelo del vehículo.
- posición en el mundo.

→

```
from CarlaEnv import *  
from Car import *  
  
env = CarlaEnv()
```

NO
CODE
TIME

```
def __init__(self, env, model, spawn_point, camera_config=None):
```

3. Invocar un automóvil

Para el modelo del vehículo, primero se necesita obtener la lista de vehículos disponibles.

```
vehicles = env.blueprint_library.filter('vehicle')
```

- conexión al servidor. ✓
- modelo del vehículo. ←
- posición en el mundo.

Un total de 26
modelos
disponibles.

NO
CODE
TIME

```
def __init__(self, env, model, spawn_point, camera_config=None):
```



3. Invocar un automóvil

Nombre, Marca, Etiqueta

Modelos disponibles:

A2, Audi, a2

TT, Audi, tt

e-tron, Audi, etron

CarlaCola, CarlaMotors, carlacola

C3, Citroen, c3

Charger, Dodge, police

Wrangler Rubicon, Jeep, jeep

YZF, Yamaha, yzf

Patrol, Nissan, patrol

Micra, Nissan, micra

Crossbike, BH, bh

Mustang, Ford, mustang

Isetta, BMW, isetta

Low Rider, Harley-Davidson, harley-davidson

Coupe, Mercedes-Benz, coupe

Grand Tourer, BMW, grandtourer

Prius, Toyota, prius

Diamondback, Century, century

Model 3, Tesla, model3

León, SEAT, leon

MKZ 2017, Lincoln, mkz2017

Ninja, Kawasaki, ninja

T2, Volkswagen, t2

Omafets, Gazelle, gazelle

Impala, Chevrolet, impala

Cooper ST, Mini, mini

3. Invocar un automóvil

Elegimos un modelo con su etiqueta y la librería de blueprints.

En el archivo main.py

```
env = CarlaEnv()

try:
    → model = env.blueprint_library.filter("model3")
finally:
    env.destroy()
```

NO
CODE
TIME

Aquí va la etiqueta
del vehículo que
ustedes prefieran.

3. Invocar un automóvil

Como vamos a invocar un vehículo, al final de programa necesitamos limpiar.

```
env = CarlaEnv()

try:
    model = env.blueprint_library.filter("model3")

finally:
    → env.destroy()
```

NO
CODE
TIME

Aún no lo
invocamos, pero
hay que estar
listos.


3. Invocar un automóvil

La estructura de control 'try' y 'finally' ayudan cuando surge algún error.

```
env = CarlaEnv()

try:
    ■■ model = env.blueprint_library.filter("model3")

finally:
    ■■ env.destroy()
```



CODE
TIME

 Indentación

3. Invocar un automóvil

Solamente falta elegir una posición en el mundo.

- conexión al servidor. ✓
- modelo del vehículo. ✓
- posición en el mundo. ←

Para esto
necesitamos la
variable 'map'.

NO
CODE
TIME

```
def __init__(self, env, model, spawn_point, camera_config=None):
```



3. Invocar un automóvil

Dependiendo del mapa, las posiciones de inicio cambian.



3. Invocar un automóvil

Con la variable 'map' obtenemos todos los puntos de inicio.

```
from CarlaEnv import *
from Car import *

env = CarlaEnv()

try:
    model = env.blueprint_library.filter("model3")[0]
    → spawn_points = env.map.get_spawn_points()
    spawn_point = points[0]

finally:
    env.destroy()
```

NO
CODE
TIME

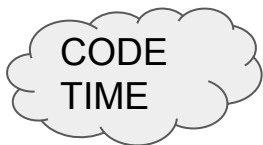
3. Invocar un automóvil

Y elegimos el primer punto.

```
from CarlaEnv import *
from Car import *

env = CarlaEnv()

try:
    model = env.blueprint_library.filter("model3")[0]
    spawn_points = env.map.get_spawn_points()
    spawn_point = spawn_points[0]
finally:
    env.destroy()
```



■ Indentación

3. Invocar un automóvil

Ya tenemos todo listo para invocar un automóvil.

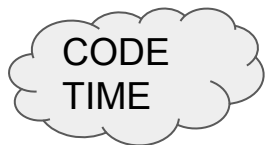
- conexión al servidor. ✓
- modelo del vehículo. ✓
- posición en el mundo. ✓

```
def __init__(self, env, model, spawn_point, camera_config=None):
```

3. Invocar un automóvil

Momento de invocar el automóvil.

```
spawn_points = env.map.get_spawn_points()  
■ spawn_point = spawn_points[0]  
➡ car = Car(env, model, spawn_point)
```



Seguimos en el
'main.py'

■ Indentación

3. Invocar un automóvil

¿Por qué el automóvil desaparece?

```
finally:  
    env.destroy()
```

Para mantener el automóvil hasta recibir una orden se necesita la siguiente línea.

```
car = Car(env, model, spawn_point)  
input('')
```

CODE
TIME

Se puede poner
cualquier mensaje
entre las comillas.

RUN
TIME

El comando input
espera a que el
usuario presione la
tecla 'Enter'.

Indentación

Agregar una cámara

4. Agregar una cámara

Creamos el archivo 'Camera.py' en la carpeta 'curso'.

Necesitamos importar la librería de carla.

```
import sys  
  
egg_path='lib/dist/carla-0.9.6-py3.6-linux-x86_64.egg'  
sys.path.append(egg_path)  
  
import carla
```



Estas mismas líneas
se encuentran en el
archivo 'CarlaEnv.py'

4. Agregar una cámara

Continuando con el paradigma de programación orientado a objetos, creamos la clase Camera.

```
import carla

import cv2
from helpers import *

class Camera:

    base_config = {
        "show": True,
        "width": 400,
        "height": 220,
        'fov': 110,
        'x': -7,
        'y': 0,
        'z': 4,
        'pitch': 0.0,
        'yaw': 0.0,
        'roll': 0.0
    }
```

NO
CODE
TIME

También se requiere importar la librería cv2.

Y unas funciones de ayuda del archivo 'helpers.py'.

4. Agregar una cámara

La configuración base de la cámara es solamente una recomendación.

```
import carla

import cv2
from helpers import *

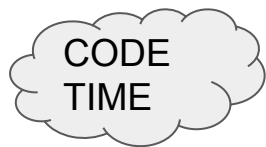
class Camera:

    base_config = {
        "show": True,
        "width": 400,
        "height": 220,
        'fov': 110,
        'x': -7,
        'y': 0,
        'z': 4,
        'pitch': 0.0,
        'yaw': 0.0,
        'roll': 0.0
    }
```

NO
CODE
TIME

4. Agregar una cámara

'show' muestra en una ventana lo que captura la cámara.



```
class Camera:
    base_config = {
        "show": True,
        "width": 400,
        "height": 220,
        'fov': 110,
        'x': -7,
        'y': 0,
        'z': 4,
        'pitch': 0.0,
        'yaw': 0.0,
        'roll': 0.0
    }
```

Indentación

4. Agregar una cámara

Ancho y alto de la imagen capturada.

```
class Camera:
    ■ base_config = {
        "show": True,
        ➡ "width": 400,
        ➡ "height": 220,
        'fov': 110,
        'x': -7,
        'y': 0,
        'z': 4,
        'pitch': 0.0,
        'yaw': 0.0,
        'roll': 0.0
    }
```

CODE
TIME

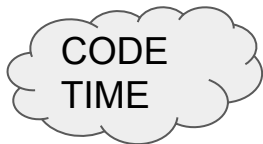
■ Indentación

Tamaños muy
grandes ralentizan el
servidor.

4. Agregar una cámara

Efecto curvo de la imagen.

```
class Camera:
    ■ base_config = {
        "show": True,
        "width": 400,
        "height": 220,
        ■ 'fov': 110,
        'x': -7,
        'y': 0,
        'z': 4,
        'pitch': 0.0,
        'yaw': 0.0,
        'roll': 0.0
    }
```



■ Indentación

4. Agregar una cámara

Posición relativa de la cámara con respecto al vehículo.

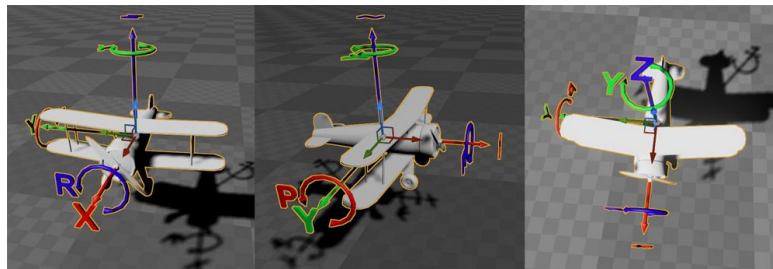
```
class Camera:
    base_config = {
        "show": True,
        "width": 400,
        "height": 220,
        'fov': 110,
        'x': -7,
        'y': 0,
        'z': 4,
        'pitch': 0.0,
        'yaw': 0.0,
        'roll': 0.0
    }
```



Indentación

4. Agregar una cámara

Rotación de la cámara.



```
class Camera:
    base_config = {
        "show": True,
        "width": 400,
        "height": 220,
        'fov': 110,
        'x': -7,
        'y': 0,
        'z': 4,
        'pitch': 0.0,
        'yaw': 0.0,
        'roll': 0.0
    }
```



Indentación

4. Agregar una cámara

Para inicializar la cámara necesitamos lo siguiente.

```
class Camera:

    base_config = { ...
    }

    def __init__(self, env, vehicle, camera_config={}):
```



NO
CODE
TIME


4. Agregar una cámara

La conexión al servidor.

```
class Camera:

    base_config = { ...
    }

    def __init__(self, env, vehicle, camera_config={}):
```



NO
CODE
TIME


4. Agregar una cámara

El vehículo en el que la vamos a montar.

```
class Camera:

    base_config = { ...
    }

    def __init__(self, env, vehicle, camera_config={}):
```



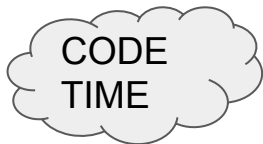

NO
CODE
TIME

Este vehículo debe
ser el actor en el
mundo.

4. Agregar una cámara

La configuración de la cámara.

```
class Camera:
    base_config = { ... }
    def __init__(self, env, vehicle, camera_config={}):
```



■ ■ Indentación

Es opcional, porque ya tenemos una configuración base.

4. Agregar una cámara

Seguimos con la inicialización de la cámara.

```
def __init__(self, env, vehicle, camera_config={}):  
  
    self.config = fill_config(camera_config, Camera.base_config)  
  
    self.camera_image = None  
  
    model = env.blueprint_library.find('sensor.camera.rgb')  
    # Tamaño de la imagen  
    model.set_attribute('image_size_x', "{}".format(self.config['width']))  
    model.set_attribute('image_size_y', "{}".format(self.config['height']))  
    model.set_attribute('fov', "{}".format(self.config['fov']))  
  
    # Posición relativa al vehículo  
    cam_pos = carla.Transform(carla.Location(x=self.config['x'], y=self.config['y'], z=self.config['z']))  
    self.camera = env.spawn_actor(model, cam_pos, vehicle)  
  
    # Cada vez que la cámara toma una imagen la procesa  
    self.camera.listen(lambda data: self.process_img(data))
```

NO
CODE
TIME

4. Agregar una cámara

Obtenemos la configuración de la cámara.

```
def __init__(self, env, vehicle, camera_config={}):  
    self.config = fill_config(camera_config, Camera.base_config)  
  
    self.camera_image = None  
  
    model = env.blueprint_library.find('sensor.camera.rgb')  
    # Tamaño de la imagen  
    model.set_attribute('image_size_x', "{}".format(self.config['width']))  
    model.set_attribute('image_size_y', "{}".format(self.config['height']))  
    model.set_attribute('fov', "{}".format(self.config['fov']))  
  
    # Posición relativa al vehículo  
    cam_pos = carla.Transform(carla.Location(x=self.config['x'], y=self.config['y'], z=self.config['z']))  
    self.camera = env.spawn_actor(model, cam_pos, vehicle)  
  
    # Cada vez que la cámara toma una imagen la procesa  
    self.camera.listen(lambda data: self.process_img(data))
```

NO
CODE
TIME

Por si hay que
cambiar la base.

4. Agregar una cámara

Obtenemos el modelo de la cámara.

```
def __init__(self, env, vehicle, camera_config={}):  
    self.config = fill_config(camera_config, Camera.base_config)  
    self.camera_image = None  
    model = env.blueprint_library.find('sensor.camera.rgb')  
    # Tamaño de la imagen  
    model.set_attribute('image_size_x', "{}".format(self.config['width']))  
    model.set_attribute('image_size_y', "{}".format(self.config['height']))  
    model.set_attribute('fov', "{}".format(self.config['fov']))  
  
    # Posición relativa al vehículo  
    cam_pos = carla.Transform(carla.Location(x=self.config['x'], y=self.config['y'], z=self.config['z']))  
    self.camera = env.spawn_actor(model, cam_pos, vehicle)  
  
    # Cada vez que la cámara toma una imagen la procesa  
    self.camera.listen(lambda data: self.process_img(data))
```

NO
CODE
TIME

4. Agregar una cámara

Dimensiones de la imagen.

```
def __init__(self, env, vehicle, camera_config={}):

    self.config = fill_config(camera_config, Camera.base_config)

    self.camera_image = None

    model = env.blueprint_library.find('sensor.camera.rgb')
    # Tamaño de la imagen
    model.set_attribute('image_size_x', "{}".format(self.config['width']))
    model.set_attribute('image_size_y', "{}".format(self.config['height']))
    model.set_attribute('fov', "{}".format(self.config['fov']))

    # Posición relativa al vehículo
    cam_pos = carla.Transform(carla.Location(x=self.config['x'], y=self.config['y'], z=self.config['z']))
    self.camera = env.spawn_actor(model, cam_pos, vehicle)

    # Cada vez que la cámara toma una imagen la procesa
    self.camera.listen(lambda data: self.process_img(data))
```

NO
CODE
TIME

4. Agregar una cámara

Distorsión de la imagen.

```
def __init__(self, env, vehicle, camera_config={}):  
  
    self.config = fill_config(camera_config, Camera.base_config)  
  
    self.camera_image = None  
  
    model = env.blueprint_library.find('sensor.camera.rgb')  
    # Tamaño de la imagen  
    model.set_attribute('image_size_x', "{}".format(self.config['width']))  
    model.set_attribute('image_size_y', "{}".format(self.config['height']))  
    model.set_attribute('fov', "{}".format(self.config['fov']))  
  
    # Posición relativa al vehículo  
    cam_pos = carla.Transform(carla.Location(x=self.config['x'], y=self.config['y'], z=self.config['z']))  
    self.camera = env.spawn_actor(model, cam_pos, vehicle)  
  
    # Cada vez que la cámara toma una imagen la procesa  
    self.camera.listen(lambda data: self.process_img(data))
```

NO
CODE
TIME

4. Agregar una cámara

Contiene la posición de la cámara en el mundo.

```
def __init__(self, env, vehicle, camera_config={}):  
  
    self.config = fill_config(camera_config, Camera.base_config)  
  
    self.camera_image = None  
  
    model = env.blueprint_library.find('sensor.camera.rgb')  
    # Tamaño de la imagen  
    model.set_attribute('image_size_x', "{}".format(self.config['width']))  
    model.set_attribute('image_size_y', "{}".format(self.config['height']))  
    model.set_attribute('fov', "{}".format(self.config['fov']))  
  
    # Posición relativa al vehículo  
    cam_pos = carla.Transform(carla.Location(x=self.config['x'], y=self.config['y'], z=self.config['z']))  
    self.camera = env.spawn_actor(model, cam_pos, vehicle)  
  
    # Cada vez que la cámara toma una imagen la procesa  
    self.camera.listen(lambda data: self.process_img(data))
```

NO
CODE
TIME

4. Agregar una cámara

Invocamos la cámara en el mundo.

```
def __init__(self, env, vehicle, camera_config={}):  
  
    self.config = fill_config(camera_config, Camera.base_config)  
  
    self.camera_image = None  
  
    model = env.blueprint_library.find('sensor.camera.rgb')  
    # Tamaño de la imagen  
    model.set_attribute('image_size_x', "{}".format(self.config['width']))  
    model.set_attribute('image_size_y', "{}".format(self.config['height']))  
    model.set_attribute('fov', "{}".format(self.config['fov']))  
  
    # Posición relativa al vehículo  
    cam_pos = carla.Transform(carla.Location(x=self.config['x'], y=self.config['y'], z=self.config['z']))  
    self.camera = env.spawn_actor(model, cam_pos, vehicle)  
  
    # Cada vez que la cámara toma una imagen la procesa  
    self.camera.listen(lambda data: self.process_img(data))
```

NO
CODE
TIME

Aquí elegimos le
decimos que siga al
vehículo.

4. Agregar una cámara

Cada vez que la cámara toma una imagen, hay que procesarla.

```
def __init__(self, env, vehicle, camera_config={}):  
  
    self.config = fill_config(camera_config, Camera.base_config)  
  
    self.camera_image = None  
  
    model = env.blueprint_library.find('sensor.camera.rgb')  
    # Tamaño de la imagen  
    model.set_attribute('image_size_x', "{}".format(self.config['width']))  
    model.set_attribute('image_size_y', "{}".format(self.config['height']))  
    model.set_attribute('fov', "{}".format(self.config['fov']))  
  
    # Posición relativa al vehículo  
    cam_pos = carla.Transform(carla.Location(x=self.config['x'], y=self.config['y'], z=self.config['z']))  
    self.camera = env.spawn_actor(model, cam_pos, vehicle)  
  
    # Cada vez que la cámara toma una imagen la procesa  
    self.camera.listen(lambda data: self.process_img(data))
```

NO
CODE
TIME

Aún no tenemos la
función 'process_img'

4. Agregar una cámara

Solamente falta definir la función que procesa la imagen.

```
def __init__(self, env, vehicle, camera_config={}):  
    self.config = fill_config(camera_config, Camera.base_config)  
  
    self.camera_image = None  
  
    model = env.blueprint_library.find('sensor.camera.rgb')  
    # Tamaño de la imagen  
    model.set_attribute('image_size_x', "{}".format(self.config['width']))  
    model.set_attribute('image_size_y', "{}".format(self.config['height']))  
    model.set_attribute('fov', "{}".format(self.config['fov']))  
  
    # Posición relativa al vehículo  
    cam_pos = carla.Transform(carla.Location(x=self.config['x'], y=self.config['y'], z=self.config['z']))  
    self.camera = env.spawn_actor(model, cam_pos, vehicle)  
  
    # Cada vez que la cámara toma una imagen la procesa  
    self.camera.listen(lambda data: self.process_img(data))
```

CODE
TIME

■ ■ Indentación

4. Agregar una cámara

En el mismo archivo 'Camera.py' y en la misma clase 'Camera', definimos la función 'process_img'

NO
CODE
TIME

```
class Camera:

    base_config = {
    }

    def __init__(self, env, vehicle, camera_config={}):

        self.config = fill_config(camera_config, Camera.base_config)

        self.camera_image = None

        model = env.blueprint_library.find('sensor.camera.rgb')
        # Tamaño de la imagen
        model.set_attribute('image_size_x', "{}".format(self.config['image_size_x']))
        model.set_attribute('image_size_y', "{}".format(self.config['image_size_y']))
        model.set_attribute('fov', "{}".format(self.config['fov']))

        # Posición relativa al vehículo
        cam_pos = carla.Transform(carla.Location(x=self.config['cam_x'], y=self.config['cam_y'], z=self.config['cam_z']))
        self.camera = env.spawn_actor(model, cam_pos, attach_to=vehicle)

        # Cada vez que la cámara toma una imagen la procesamos
        self.camera.listen(lambda data: self.process_img(data))

    def process_img(self, image):
        # La imagen tomada por CARLA es transformada a un vector de 3 canales de color
        i = get_rgb_image(image, self.config)

        # Aquí se muestra la imagen tomada por la cámara
        if self.config["show"]:
            cv2.imshow("", i)
            cv2.waitKey(1)

        # Imagen actual
        self.camera_image = i
```

4. Agregar una cámara

Primero transformamos la imagen de CARLA en una imagen visible.

```
def process_img(self, image):  
    # La imagen tomada por CARLA es transformada a un formato de cuatro canales en lugar  
    # de un solo vector  
    → i = get_rgb_image(image, self.config)  
  
    # Aqui se muestra la imagen tomada por la camara de CARLA  
    if self.config["show"]:  
        cv2.imshow("", i)  
        cv2.waitKey(1)  
  
    # Imagen actual  
    self.camera_image = i
```

NO
CODE
TIME

4. Agregar una cámara

Luego desplegamos la imagen en una ventana.

```
def process_img(self, image):  
    # La imagen tomada por CARLA es transformada a un formato de cuatro canales en lugar  
    # de un solo vector  
    i = get_rgb_image(image, self.config)  
  
    # Aqui se muestra la imagen tomada por la camara de CARLA  
    if self.config["show"]:  
        → cv2.imshow("", i)  
        cv2.waitKey(1)  
  
    # Imagen actual  
    self.camera_image = i
```

NO
CODE
TIME

4. Agregar una cámara

Al final guardamos la imagen actual en la variable 'self.camera_image'.

```
■ ■ def process_img(self, image):  
    # La imagen tomada por CARLA es transformada a un formato de cuatro canales en lugar  
    # de un solo vector  
    ■ ■ i = get_rgb_image(image, self.config)  
  
    # Aqui se muestra la imagen tomada por la camara de CARLA  
    if self.config["show"]:  
        cv2.imshow("", i)  
        ■ ■ cv2.waitKey(1)  
  
    # Imagen actual  
    ■ ■ self.camera_image = i
```



■ ■ Indentación


4. Agregar una cámara

Listo, ya definimos el comportamiento de la cámara, sólo falta invocarla.

Pero, será desde el archivo 'Car.py'


Porque necesitamos el actor del vehículo.

```
class Camera:
    base_config = {
    }
    def __init__(self, env, vehicle, camera_config={}):
```



4. Agregar una cámara

En el archivo 'Car.py' hay que importar la clase 'Camera'.



```
from Camera import *  
  
class Car:  
    def __init__(self, env, model, spawn_point, camera_config=None):  
        # Invoca el vehiculo en el mundo
```



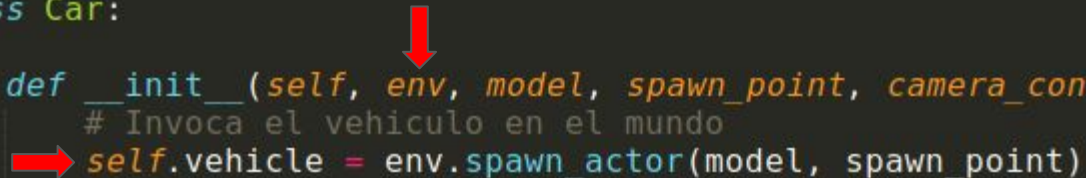
■ ■ Indentación

4. Agregar una cámara

Para crear una cámara necesitamos los siguientes parámetros:

- conexión al servidor. ✓
- actor del vehículo. ✓

```
class Car:
    def __init__(self, env, model, spawn_point, camera_config=None):
        # Invoca el vehiculo en el mundo
        self.vehicle = env.spawn_actor(model, spawn_point)
```



Ya los tenemos.
En 'env' y 'vehicle'

4. Agregar una cámara

El parámetro 'camera_config' de la clase 'Car' en el archivo 'Car.py' tiene el valor por defecto 'None'

```
from Camera import *  
  
class Car:  
    def __init__(self, env, model, spawn_point, camera_config=None):  
        # Invoca el vehiculo en el mundo
```

```
class Car:  
    def __init__(self, env, model, spawn_point, camera_config={}):  
        # Invoca el vehiculo en el mundo
```

Hay que cambiarlo
por '{}'

CODE
TIME

Indentación

4. Agregar una cámara

Ahora ya podemos invocar una cámara.

```
class Car:
    ■■ def __init__(self, env, model, spawn_point, camera_config={}):
        # Invoca el vehiculo en el mundo
        ■■ self.vehicle = env.spawn_actor(model, spawn_point)
        ➡ self.camera = Camera(env, self.vehicle, camera config)
```

CODE
TIME

■■ Indentación

4. Agregar una cámara

No es necesario modificar el archivo 'main.py'. Ya podemos ejecutar el programa.

```
from CarlaEnv import *
from Car import *

env = CarlaEnv()

try:
    model = env.blueprint_library.filter("model3")[0]

    spawn_points = env.map.get_spawn_points()
    spawn_point = spawn_points[0]

    car = Car(env, model, spawn_point)

    input('Enter')

finally:
    env.destroy()
```



Fin de la primer parte

¿Preguntas?

Dudas o aclaraciones:
rperalta@cicese.edu.mx