



Manual de usuario para el sistema diseñado para el proyecto:
Diseño y validación de algoritmos de detección de objetos para la capa de
percepción del Sistema Avanzado de Asistencia para Conducción (ADAS) en
vehículos autónomos

Table of Contents

1	Introducción	3
2	Requisitos computacionales	3
2.1	Simulador Monodrive 1.16	3
2.2	CUDA 11.7	4
2.3	Librerías necesarias para cada prototipo	4
3	Configurar comunicación	5
4	Funcionamiento general	6
5	Posibles errores y sus respectivas soluciones	8

1 INTRODUCCIÓN

En este documento se describe el proceso que se debe llevar a cabo para poder utilizar correctamente el sistema diseñado para la solución del proyecto denominado: “Diseño y validación de algoritmos de detección de objetos para la capa de percepción del Sistema Avanzado de Asistencia para Conducción (ADAS) en vehículos autónomos”, la finalidad de este es brindar al usuario todo lo necesario para el uso adecuado del sistema diseñado (en este caso, para las tres soluciones realizadas).

Es importante recalcar que este manual se encuentra dividido en secciones y subsecciones para facilitar la navegación y búsqueda de información específica. Cada sección se enfoca en un aspecto particular del producto o servicio, proporcionando instrucciones paso a paso y descripciones detalladas para su correcta utilización. Además, se encuentra una sección dedicada a posibles errores que se pueden encontrar durante el uso del sistema.

2 REQUISITOS COMPUTACIONALES

El sistema diseñado tiene requisitos distintos dependiendo de cuál de los modelos se desee utilizar, no obstante, existen requisitos necesarios para cualquiera de estos modelos.

2.1 Simulador Monodrive 1.16

Las pruebas de funcionamiento del sistema diseñado dependen de un entorno de simulación, este sistema fue hecho basado en el funcionamiento del simulador “NI Monodrive 1.16”, este software se encarga de simular el ambiente virtual de pruebas de funcionamiento, así como mantener una comunicación entre la capa de percepción, la capa de control y el mismo entorno de simulación.

Para la descarga de este software se requiere una cuenta de NI Monodrive y descargar el cliente desde la página oficial del mismo (<https://www.monodrive.io>). Es importante recalcar que los requisitos mínimos para poder utilizar este software son los siguientes:

- Intel Core i9-9900 o mejor
- NVIDIA RTX 2080Ti o mejor
- 32GB RAM o más
- NVMe SSD

Una vez descargados los archivos, se obtendrá el archivo de nombre VehicleAI.exe el cual funciona como el simulador de escenarios, no obstante, para el correcto funcionamiento de este se requiere que el computador que se utilice tenga instalado Unreal Engine 4 (<https://www.unrealengine.com/>)

Ahora que se puede utilizar el simulador de Monodrive, se requiere utilizar alguno de los tres clientes que se pueden utilizar, el de Python, el de LabView o bien, el de C++. El proyecto fue realizado en base al cliente de Python por lo que se recomienda el uso de este.

El cliente de Python cuenta con múltiples scripts base con ejemplos de las distintas funciones, se recomienda leer la documentación oficial de monodrive para conocer más acerca de estas (<https://monodrive.readthedocs.io/en/latest/>). Los scripts más relevantes para el uso del proyecto son los relacionados con el funcionamiento ‘fixed step’, el cual permite controlar cuanto tiempo de la simulación se va a correr por iteración. Dentro del script encargado del funcionamiento fixed step se realiza el llamado a las diferentes funciones propias al sistema diseñado.

2.2 CUDA 11.7

Los tres prototipos diseñados tienen una dependencia en común y esta es el uso de CUDA, el cual es una herramienta que permite usar un uso óptimo de la GPU para tareas relacionadas con inteligencia artificial, como lo son entrenamientos e inferencias con modelos en tiempo real. Para el proyecto se utilizó la versión de CUDA 11.7, se recomienda seguir paso a paso las instrucciones presentes en la página oficial de Pytorch para realizar una instalación correcta de esta herramienta (<https://pytorch.org>). Incluso si el sistema computacional no posee una GPU, existen versiones de CUDA que permiten el uso de estos algoritmos sin la necesidad de la misma, no obstante, se verá reflejado en un mayor tiempo de procesamiento en la mayor parte de las acciones realizadas con el sistema diseñado.

2.3 Librerías necesarias para cada prototipo

Cada prototipo requiere de Python 1.7, así como las librerías que requiere cada prototipo, se recomienda generar un entorno virtual para el uso de cada uno con el fin de evitar incompatibilidades entre dichas librerías.

Primer Prototipo: Yolov8

La principal ventaja de utilizar Yolov8 en vez de otras versiones es la facilidad con la que se puede instalar, puesto que Ultralytics tiene su propia librería de pip, que permite instalar todas las dependencias inmediatamente a un entorno virtual que utilice Python. Solo se necesita abrir una terminal en el entorno virtual y utilizar el siguiente código:

```
pip install ultralytics (El proyecto se realizó en la versión 8.0.104)
```

Segundo prototipo: Detectron2

El modelo de Detectron2 tiene el contra de que ha sido desarrollado para Linux, por lo que al intentar desarrollar aplicaciones que utilicen este software en otros sistemas operativos, pueden destacar errores de compatibilidad e inclusive puede no funcionar en su totalidad.

Primeramente, se deben instalar las dependencias de terceros, las cuales se encuentran en el archivo llamado requirements.txt presente en la carpeta llamada 'docs' en el github oficial de Detectron2. [2]

La forma más sencilla de instalar todas las dependencias necesarias es clonar el repositorio oficial en la máquina que se va a utilizar (preferiblemente en Linux) y correr en la terminal el siguiente código:

```
git clone https://github.com/facebookresearch/detectron2.git
python -m pip install -e detectron2
cd docs
python -m pip install -e requirements.txt
```

De esta manera se instalará tanto la versión de Detectron2 (para el proyecto se utilizó la 0.0.6) y todas las dependencias relacionadas a este.

Tercer prototipo: SSD-NAS

Dentro de las ventajas de Yolo-NAS existe la facilidad con la que se pueden instalar las dependencias necesarias, se necesita un entorno virtual dedicado para el uso de aplicaciones que dependan de la librería dedicada a Yolo-NAS y como este posee un comando oficial por medio de pip, solo se requiere correr la siguiente línea de código y automáticamente se instalarán todas las dependencias requeridas.

```
pip install super-gradients
```

La instalación de librerías de cada prototipo se detalla en el apéndice A.1 del informe del proyecto, por lo que se recomienda una lectura de dicho documento para realizar una instalación correcta de dichas librerías.

3 CONFIGURAR COMUNICACIÓN

Al tener los entornos virtuales preparados de la forma correcta, se puede configurar la comunicación por medio del cliente de Monodrive, en el cliente de Python se puede acceder a múltiples archivos con la extensión. json los cuales se encargan de la configuración de las distintas funciones del simulador.

El archivo nombrado runApp.json contiene la información requerida para realizar una comunicación por medio de TCP/IP, en este archivo se debe configurar el puerto de comunicación por utilizar, así como la dirección IP tanto del servidor como del receptor. Una vez se ingresó esta información de forma correcta, se utiliza el módulo incorporado en Python llamado 'socket'. El primer paso es crear un objeto de tipo socket el cual se encarga de dicha comunicación.

```
import socket
# Crear un objeto socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Seguidamente se utiliza la información de runApp.json para conectarse y enviar datos al servidor.

```
# Conectar al servidor
```

```
sock.connect((server_ip, server_port))
```

Si los datos ingresados son correctos, se realizará una conexión entre el script principal del sistema diseñado con el simulador, luego se debe repetir el proceso para conectar de forma efectiva el script encargado de la capa de control. Se recomienda realizar pruebas de comunicación enviando mensajes sencillos por medio de código de Python (ejemplo a continuación) para corroborar que la comunicación es efectiva.

```
# Se envían datos al servidor

data = 'Hola Mundo'

sock.send(data.encode())

# Se reciben datos del servidor

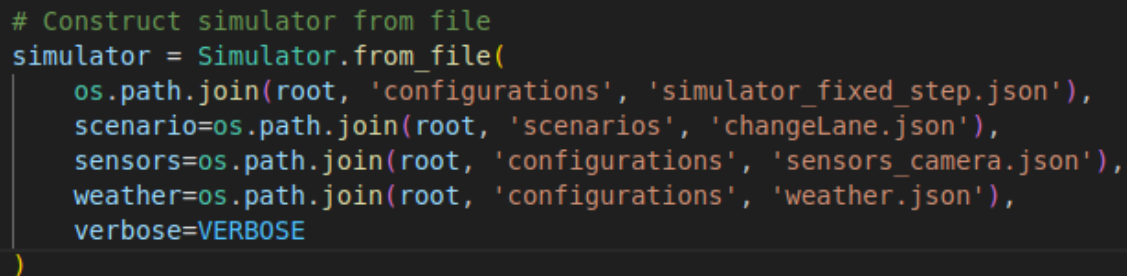
received_data = sock.recv(1024).decode()

print ('Recibido:', received_data)
```

4 FUNCIONAMIENTO GENERAL

Una vez configurados todos los pasos previos al funcionamiento, se puede poner a prueba el funcionamiento del sistema diseñado. El primer paso para utilizar el SD es activar el entorno virtual del prototipo que se desee utilizar, seguidamente en la computadora que se desee utilizar como servidor se debe de inicializar el simulador por medio del archivo llamado VehicleAI.exe, presente dentro de los documentos de la instalación de Monodrive. Una vez se encuentre inicializado se puede correr el script llamado ‘Simulación’ presente en la carpeta de anexos del informe del proyecto.

Este script se encarga de realizar la puesta en funcionamiento del escenario de simulación que se encuentre seleccionado, cada escenario tiene asociado un archivo .json y un mapa, el escenario 1 utiliza el archivo de nombre ‘Scenario2Infinity.json’, el escenario 2 utiliza ‘changeLane.json’ y el escenario 3 utiliza ‘Scenario3.json’, para seleccionar el escenario que se desea utilizar, se debe cambiar el escenario en la línea 118 en la función ‘simulator’

A screenshot of a code editor showing Python code for constructing a simulator. The code is as follows:

```
# Construct simulator from file
simulator = Simulator.from_file(
    os.path.join(root, 'configurations', 'simulator_fixed_step.json'),
    scenario=os.path.join(root, 'scenarios', 'changeLane.json'),
    sensors=os.path.join(root, 'configurations', 'sensors_camera.json'),
    weather=os.path.join(root, 'configurations', 'weather.json'),
    verbose=VERBOSE
)
```

Figura 1. Selección de escenario

El script está diseñado de tal forma que puede utilizar cualquiera de los prototipos, el único cambio que se debe realizar es activar el entorno virtual destinado al prototipo que se desea utilizar. Para seleccionar cuál prototipo se va a utilizar, en la línea 244 se encuentra la línea de código destinada a llamar al script necesario para cada uno de los prototipos, se debe quitar

el comentario (signo # a la izquierda) del que se desee utilizar y mantener el símbolo en los que no se utilizan.

```
nuevalista = Track-Yolo.detect_track_ODT(im)
# nuevalista = Detectron2.detect_track_ODT(im)
# nuevalista = SSD_NAS.detect_track_ODT(im)
```

Figura 2. Selección de prototipo

Luego de seleccionar y configurar todo lo necesario, se puede correr el script 'Simulacion.py' en la consola y se debería mostrar en pantalla la simulación con los resultados tanto de la detección como el rastreo en tiempo real, tal y como se ve en la figura 3.

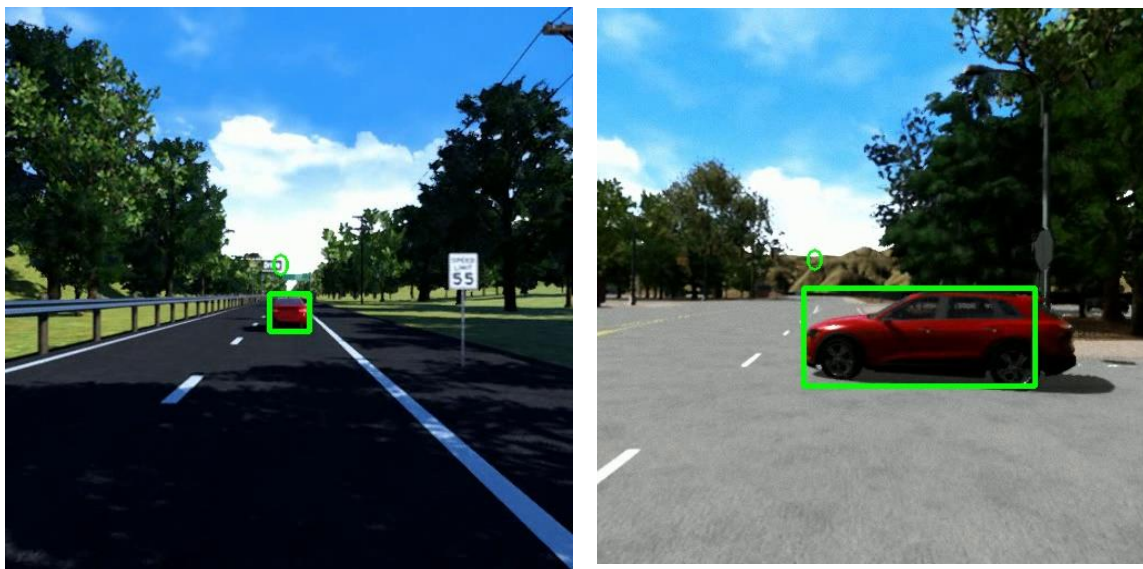


Figura 3. Resultados de la simulación en tiempo real

5 POSIBLES ERRORES Y SUS RESPECTIVAS SOLUCIONES

El script crashea y no corre la simulación

Este error tiene varios posibles orígenes:

- No se inició el ejecutable VehicleAI.exe antes de correr el script
- El entorno virtual utilizado no posee todos los requerimientos necesarios
- No se instaló CUDA de forma correcta
- No se activó el entorno virtual de forma correcta

Se recomienda revisar cuál de esas fuentes es el origen del error y seguir los pasos de las secciones anteriores de este documento.

La cámara del automóvil solo muestra el interior del automóvil

Este problema se da por un error en la configuración de la posición de la cámara para la simulación, para corregirlo se debe cambiar el objeto de cámara en el archivo llamado 'sensors_camera.json' y aumentar el valor de la posición en z.

```
{  
  "type": "Camera",  
  "packet_size": 23552,  
  "listen_port": 8000,  
  "location": {  
    "x": 0.0,  
    "y": 0.0,  
    "z": 150.0  
  },  
}
```

Error: CUDA status not Initialized

Este error aparece si la versión instalada de CUDA no es compatible con el entorno virtual que se desea utilizar. Para solucionar este problema se recomienda desinstalar la versión actual de CUDA e instalar la versión 11.7, la cual se puede encontrar en: <https://pytorch.org/get-started/previous-versions/>