UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

Diseño e implementación de un prototipo de vehículo autónomo a escala con capacidades de visión artificial

Christian Eduardo López Armas

Joseph Daniel Espinosa Campos

Kevin Paul Villarreal Medina

Ingeniera Electrónica

Trabajo de fin de carrera presentado como requisito

para la obtención del título de

Ingeniero Electrónico

Quito, 07 de 12 de 21

Universidad San Francisco de Quito USFQ

Colegio de Ciencias e Ingenierías

HOJA DE CALIFICACIÓN

DE TRABAJO DE FIN DE CARRERA

Design and Implementation of a scale autonomous vehicle prototype with computer vision capabilities

Christian Eduardo López Armas

Joseph Daniel Espinosa Campos

Kevin Paul Villarreal Medina

Nombre del profesor, Título académico Rene Játiva, Ingeniero Electrónico

Quito, 07 de 12 de 21

© DERECHOS DE AUTOR

Nombres y apellidos: Christian Eduardo López Armas
Joseph Daniel Espinosa Campos
Kevin Paul Villarreal Medina

Código: 00129919
00135009
00129060

Cédula de identidad: 1723586556
1717999658
1718645391

Lugar y fecha: Quito, 07 de 12 de 21

RESUMEN

El presente informe detalla la implementación y desarrollo de un prototipo de auto autónomo (nivel de autonomía de nivel 3 - 4) a escala, que utiliza una red neuronal y un mando Dualshock cuatro de la consola Play Station 4 (PS4) para su accionamiento. El mecanismo del vehículo consta con dos principales modos de interacción: un modo manual (en el que se utiliza el mando de PS4 para el movimiento y acciones dentro de la plataforma) y un modo automático (a través del cual, el vehículo se desplaza de manera automática reaccionando con lo que se encuentra a su entorno tanto con los sensores de proximidad; así como también con la red neuronal).

El sistema se ha planteado desde una perspectiva académica para posibilitar su escalabilidad progresiva en proyectos posteriores pasando de entornos controlados a escenarios realistas de mayor complejidad, y eventualmente la implementación de estas tecnologías en un vehículo automotor real.

Palabras Clave: Vehículo Eléctrico, Autonomía, Sensores de Proximidad, Red Neuronal, PS4, CNN, SDK, NCS2, GPU.

ABSTRACT

The following document presents the detailed development and implementation of an automatic scale vehicle (autonomy level 4). The driving system includes the use of a neural network and a Dualshock Playstation 4 controller. The vehicle's mechanism is programmed with two main interaction modes: a manual mode (where the console controller is used to perform actions displayed by the console) and automatic mode (where the vehicle displaces itself thanks to the sensor's reaction and the neural network interpretation of different objects from the environment).

The system has been proposed from an academic perspective to enable its progressive scalability in subsequent projects, moving from controlled environments to more complex realistic scenarios, and eventually to the implementation of these technologies in an actual automotive vehicle.

Keywords: Electric Vehicle, Autonomy, Proximity Sensor, Neural Network, A.I, PS4, CNN, SDK, NCS2, GPU.

Tabla de contenido

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

OBJETIVOS

Tomando en consideración los diferentes niveles de automatización disponibles para sistemas de conducción autónoma; se ha desarrollado el proyecto con la intención de obtener la mayor aproximación posible hacia un nivel de autonomía cuatro. Para la corroboración de esto, se realizaron pruebas bajo las siguientes tres condiciones:

- Conducción realizada a partir del mando de PlayStation cuatro en un ambiente controlado.
- Conducción manual reconociendo los objetos que posiblemente llegarían a interactuar con el vehículo.
- Conducción automática del vehículo evitando obstáculos por medio del mecanismo de sensores.
- Conducción automática del vehículo por medio del reconocimiento de objetos.

INTRODUCCIÓN

Muy recientemente la intervención de dispositivos con inteligencia artificial ha dejado de ser únicamente un idílico recurso literario presente en relatos de ciencia ficción; para convertirse en una plausible realidad. Autos que se conducen solos y cámaras que pueden identificar y clasificar objetos son algunos de los ejemplos más notorios del fenómeno tecnológico previamente descrito. Una de las aplicaciones más llamativas (y esperadas) de la última década es la implementación de este tipo de tecnología en conducción autónoma; permitiendo a un vehículo ser capaz de transportar pasajeros sin la necesidad de un conductor al volante.

Como ha sido previamente mencionado en los objetivos, la intención del presente proyecto es continuar con las etapas de desarrollo de un vehículo autónomo. Para esto se parte de la

modificación de un vehículo recreacional para niños; al mismo que se pretende implementar un sistema de control eléctrico que le permita desplazarse sin la necesidad de un piloto a bordo.

En las etapas iniciales se realizaron pruebas para verificar el estado de funcionamiento del vehículo. Tras observar múltiples problemas en el esquema implementado inicialmente que no lo hacían funcional, se realizó un rediseño del sistema eléctrico. Se añadió la implementación de cuatro sensores de proximidad que fueron utilizados en el mecanismo de control.

En cuanto al sistema controlador del vehículo se refiere, se utilizó como núcleo de control un dispositivo Raspberry Pi 4. Las señales obtenidas por el sistema de sensores son recibidas a través de Raspberry para su interpretación y utilización como evasores de obstáculos. El programa principal del mecanismo de control utiliza la librería PyGame del leguaje Python y se integró un modo de funcionamiento manual con la utilización de un mando Dual Shock 4 de Play Station 4.

Por otro lado, la visión artificial fue implementada en un Nvidia Jetson Nano. El entrenamiento de la red neuronal fue realizado a través de la interpretación de bancos de imágenes; especializándola en el reconocimiento de objetos relacionados con el tránsito vehicular; como lo son carros, buses, señales de tránsito y personas.

MARCO TEÓRICO

Red Neuronal y Visión Artificial:

Las Redes Neuronales son aplicaciones tecnológicas involucradas con el diseño y creación de inteligencias artificiales. Se encuentran inspiradas por el complejo funcionamiento y estructura del cerebro humano, es decir, imitan la interconexión de neuronas encargadas de transmitir señales entre si hasta generar una salida en el sistema.

Aunque su clasificación y la forma de generarlas es muy variada y extensa; para el propósito del proyecto se enfoca en un área específica de la inteligencia artificial conocida como Visión Artificial.

La Visión Artificial utiliza métodos computacionales para adquirir, procesar y analizar objetos para ser interpretados como imágenes por una máquina. La función principal en el caso específico del proyecto es la de identificación y calificación de objetos en vehículos, señales de tránsito y personas. De esta manera, una vez obtenidas y procesadas las imágenes en tiempo real, el vehículo reaccionará a su entorno. (Contaval, 2016)

Niveles de Automatización:

A los múltiples sistemas de conducción autónoma, se los ha podido clasificar en diversos niveles dependiendo de la naturaleza de sus características de funcionamiento. Es decir, tomando en consideración la manera en que los sistemas consiguen interactuar y desempeñar funciones con un mayor o menor grado de intervención humana. Estos niveles de autonomía son los siguientes (Gasser, 2013):

- Nivel 0: Sin automatización en la conducción
    - Todas las acciones son realizadas por el conductor.
- Nivel 1: Asistencia en la conducción.

    - El vehículo posee algún mecanismo de control automatizado, ya sea para el control del movimiento longitudinal o lateral pero no ambos de forma simultánea.
- Nivel 2: Automatización parcial de la conducción.
    - El vehículo posee un sistema de automatización donde el conductor no necesita realizar tareas derivadas del movimiento. Sin embargo, debe estar atento a todo lo que sucede en caso de que una intervención deba ser necesaria.
- Nivel 3: Automatización condicionada de la conducción.
    - El vehículo posee un sistema de automatización donde el conductor no necesita realizar tareas derivadas del movimiento, como la detección y respuesta ante objetos y posibles eventualidades de manera automática. En este nivel, el conductor puede intervenir solamente si el sistema lo solicita o un fallo se encuentra presente.
- Nivel 4: Automatización elevada de la conducción.
    - El vehículo posee todas las características anteriores, además del detalle que el conductor no necesita intervenir. Sin embargo, sigue limitado bajo condiciones particulares donde el sistema podría presentar alguna falla.
- Nivel 5: Automatización completa de la conducción.
    - El vehículo posee todas las funcionalidades instanciadas con antelación en los niveles previo. Además, cuenta con la cualidad de no poseer ningún limitante dentro de su funcionamiento autónomo; es decir, el conductor no figura de ningún tipo de intervención dentro del sistema.

Python:

Python es un lenguaje de programación cuya característica principal es el estar basado en la legibilidad del lenguaje inglés para la codificación de sus comandos. Es decir, se encuentra simplificado para utilizar las propias palabras de la lengua inglesa para la estructura de programación multiparadigma. El lenguaje se encuentra orientado con soporte hacia objetos; consiguiendo que la programación se lleve a cabo mediante la digitación de comandos en líneas de código que de manera posterior se ejecutarán de manera secuencial.

Este lenguaje es de código abierto (de colaboración abierta y no necesita ningún tipo de licencia para su uso) y considerado actualmente como uno de los más populares debido a que es de fácil interpretación, dinámico y multiplataforma. Se utilizó este lenguaje de programación para el desarrollo del proyecto por su facilidad de ejecución en sistemas compatibles con Ubuntu (sistema operativo utilizado en los dos microcomputadores del proyecto).

Raspberry Pi 4:

Se trata de la cuarta versión de la placa SBC (Single Board Computer) de Raspberry la cual ha aumentado su cantidad de memoria RAM a 4GB; incrementando sus capacidades, así como también su capacidad de procesamiento. Este ordenador se encuentra posicionado como uno de los más completos del mercado en cuanto a su funcionalidad, manteniendo un precio accesible.

Dada la versatilidad de sus posibles aplicaciones; en el proyecto se lo utilizó como procesador central. Es el encargado de manejar el sistema de control y monitoreo para el vehículo mediante los sensores de proximidad.

Jetson Nano:

Es la plataforma de NVIDIA diseñada para las aplicaciones de inteligencia artificial. Este dispositivo integra una GPU (128-core NVIDIA Maxwell™) capaz de ejecutar múltiples redes neuronales a la vez.

Esta microcomputadora fue destinada al uso de la inteligencia artificial que se implementó; la misma que tiene como encargo la clasificación y reconocimiento de los objetos delante del vehículo. (NVIDIA, NVIDIA Jetson Linux Driver Package Software Features, 2021)

DESARROLLO DE PROYECTO

Lista de componentes.

- **BMW i8 Spyder Toy Rollplay**

  Vehículo eléctrico diseñado comercialmente como un juguete a ser conducido por niños de hasta 6 años. Fue utilizado como base del proyecto. Del vehículo original se tomaron las luces, los sistemas de tracción, su motor trasero, su estructura de dirección, el chasis y su carrocería. En la carrocería se instalaron los sensores de proximidad. El motor de dirección fue cambiado debido a inconvenientes con el anterior motor, entre las características del nuevo motor se tiene que es de 12V y tiene una velocidad de 1000 rpm (revoluciones por minuto). En su interior se instaló el Raspberry pi 4 como núcleo de control para la conducción en modos manual y automática. Finalmente se colocó al Jetson Nano para implementar la visión artificial.

- **Motor RS 500**

  Motor utilizado para el movimiento vertical del vehículo, con 12V de alimentación a partir de las baterías de ácido plomo (detalladas más adelante). Mecánicamente el

motor trasero mueve la llanta posterior derecha que transmite el movimiento al vehículo.

Este mismo tipo de motor fue utilizado para la dirección del vehículo. El motor se instaló haciendo una adaptación mecánica en la dirección. Se realizó la ampliación del espacio dispuesto inicialmente para el motor y la adaptación de una guía metálica para el sistema de piñones de la caja reductora. Debido a la gran potencia del motor, se implementó la capacidad de regulación de potencia usando la funcionalidad de modulación PWM dentro del código del controlador.

- **Step Down LM2596**

  Conversores utilizados para la reducción de voltaje DC-DC. De esta manera se alimentó al Driver, los sensores, al Raspberry pi 4 y al Jetson nano juntamente con sus periféricos respectivos.

- **Raspberry Pi 4 Computer Model B 4GB RAM**

  Microcomputador destinado como núcleo de control (ya sea en modo manual o automático). Gracias al uso de sus GPIOS es plausible controlar el movimiento y las lecturas de los sensores de proximidad.

- **Baterías de Plomo Ácido**

  Tres baterías de 12 volteos utilizadas para alimentar los sistemas del proyecto.

- **Conectores EC5**

  Conectores usados para las baterías de manera que el diseño para la alimentación del sistema sea modular y fácil para recargarlas de manera independiente de ser necesario.

- **Cables AWG 14**

  Cable utilizado para la circuitería del vehículo.

- **Jumpers de protoboard**

  Cables normalizados para la conexión entre los GPIO y pineras genéricas.

- **DualShock 4 Wireless Controller**

  Control inalámbrico perteneciente a la consola de videojuegos “Playstation 4”. Establece su vínculo con el sistema mediante conexión bluetooth. El uso de las funciones de la librería de pygame consigue el mapeo de las señales emitidas por el control al pulsar cualquier botón.

- **SparkFun Monster Moto Shield**

  Driver dual que permite el acoplamiento del sistema, mediante la conexión de los GPIOS, con un full-bridge para el control de dos motores de alto consumo de corriente, precisamente el tipo de motores que fueron instalados.

ETAPAS DE DESARROLLO

Nociones Previas

Como antecedente a la realización de este proyecto, se trabajó durante la clase de Sistemas de Comunicaciones, en la implementación y entrenamiento de una Inteligencia Artificial (IA) con la que se logró identificar vehículos de dos ruedas, bicicletas, peatones y automotores de varios tamaños. Se utilizó la plataforma Raspberry Pi 4 juntamente con el coprocesador denominado Neural Computer Stick 2 (NCS2). A través de la integración y manipulación de sistemas en estos dispositivos se obtuvo un primer acercamiento a una inteligencia artificial.

Con miras a la realización de este proyecto, se propuso el diseño de un vehículo con capacidades de navegación autónoma que integre funcionalidades de IA. El principal interés de la IA es la asistencia a la navegación, que incluye el monitoreo permanente del entorno circundante y la detección y catalogación de eventos en tiempo real con capacidad de perjudicar la integridad del vehículo o de terceros a lo largo del recorrido, facilitando la reacción oportuna y pertinente del vehículo ante estos sucesos. Se utilizan uno o varios microcomputadores interconectados para maximizar la potencia de procesamiento y proporcionar seguridad a través del uso de redundancia. Puesto que ya se disponía de un prototipo de vehículo autónomo de tamaño reducido, equipado con un Raspberry pi 3 y un sensor LIDAR, se decidió aumentar la escala del vehículo y utilizar como procesador central uno con una GPU integrada para mejorar el desempeño de la red neuronal (José Barbosa, 2021).

Dado que este proyecto se inició a partir de otros anteriores, el primer curso de acción fue el revisar y evaluar los puntos fuertes y las posibles fallas de dichos trabajos; para buscar las mejorías a plantear en este proyecto. A continuación, se muestra el estado del vehículo previo a la manipulación realizada por el grupo.

Ilustración 1 Vehículo del proyecto anterior en el que se trabajó

Ilustración 2 Vehículo del proyecto Anterior instrumentación

Rediseño Eléctrico

Una vez realizadas las pruebas preliminares sobre el vehículo recibido, se determinó la pertinencia de realizar un rediseño del sistema eléctrico; puesto que el anterior presentaba varias fallas que imposibilitaron comprobar con seguridad el funcionamiento del vehículo. Particularmente, el cableado desorganizado y muy escasamente documentado se prestaba a confusión al no respetar las normativas de color para las polarizaciones del mecanismo de alimentación.

El rediseño inició con la construcción de placas de los sistemas de encendido y de carga para las distintas baterías utilizadas. Así mismo el tipo de cable fue reemplazado con otro del mismo calibre, pero con mayor maleabilidad, para soportar la corriente de trabajo de los motores. Estas modificaciones fueron diseñadas en Cade Simu y Proteus para posteriormente ser implementados en el vehículo.

A continuación, se muestran los diagramas circuitales correspondientes:

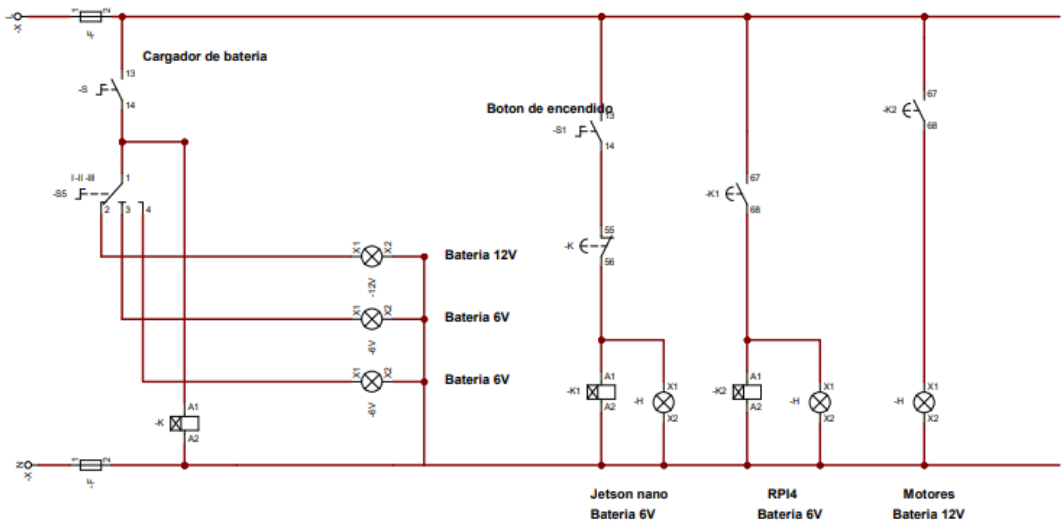


Ilustración 3 Diseño de circuito para la carga de las baterías

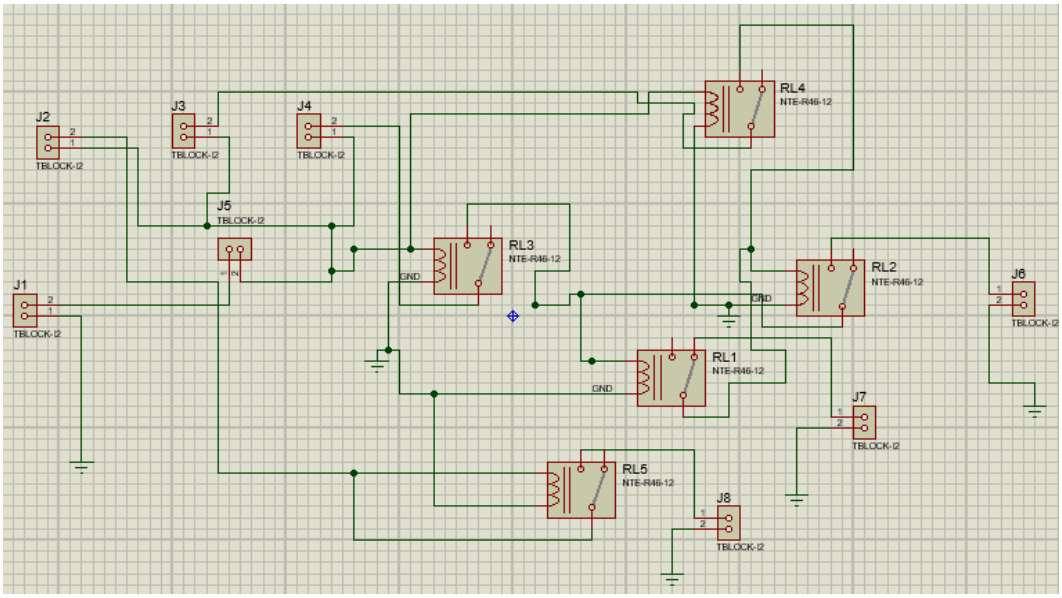


Ilustración 4 Prueba y Diseño de PCB para la carga de Baterías

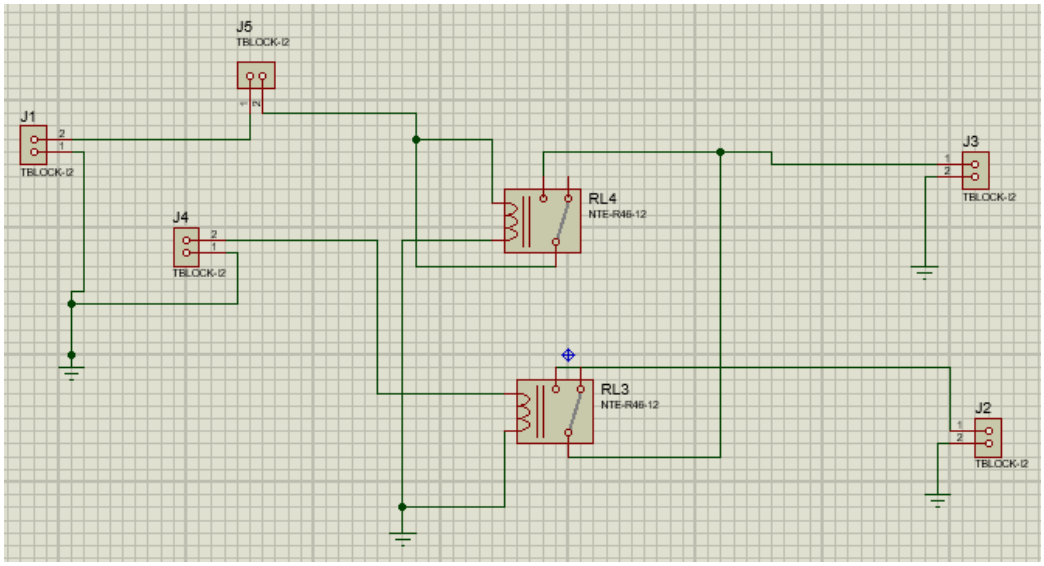


Ilustración 5 Prueba y Diseño PCB para la alimentación del procesador central

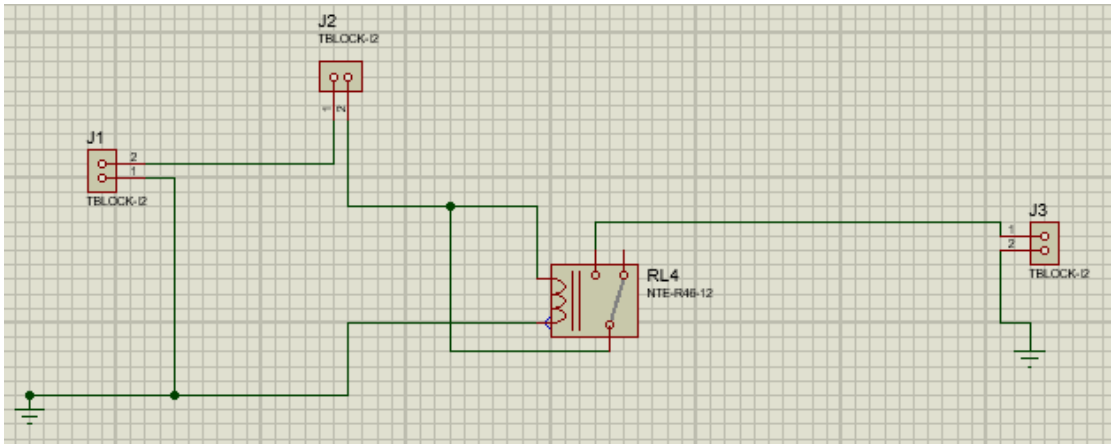


Ilustración 6 Prueba y Diseño PCB para la alimentación de los motores

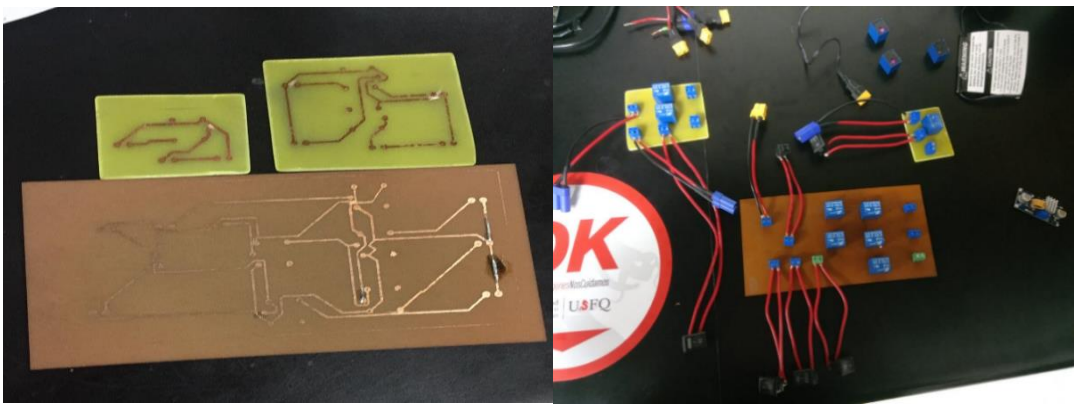


Ilustración 7 Implementación y Construcción de las diferentes placas

Una vez probadas y terminadas las placas en físico se implementó el nuevo cableado del vehículo. Otro cambio que se realizó en el hardware del vehículo fue el reemplazo del motor de la dirección a uno de 12 V, juntamente con la adición de piñones que facilitaron la instalación y el reemplazo del eje del motor. Las siguientes imágenes denotan los procesos previamente mencionados:

Ilustración 8 Vehículo desmantelado

Ilustración 9 Eje de dirección del cual se reemplazó el motor

Finalmente se instalaron cuatro sensores de proximidad: tres en la parte delantera del vehículo (izquierda, frente, derecha) y uno en la parte posterior. El último detalle que se adicionó fue el cableado de alimentación separado del Jetson Nano. La configuración ilustrada se muestra en las siguientes imágenes:

Ilustración 10 Sensores de proximidad instalados en el chasis del vehículo

Diagramas de Conexiones

A continuación, se presentan los diagramas correspondientes a la conexión de la circuitería del vehículo.

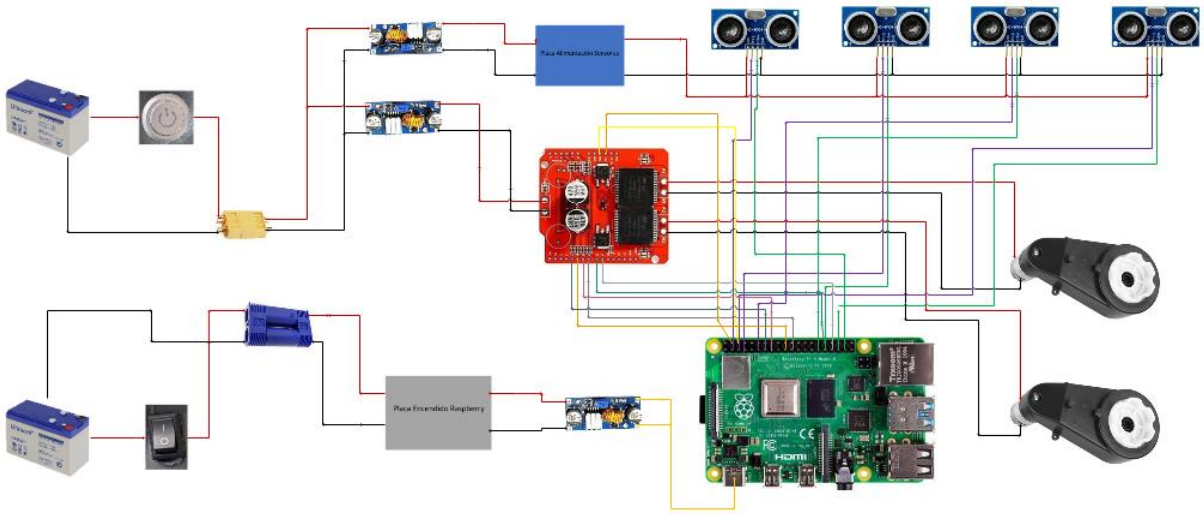


Ilustración 11 Diagrama completo de conexiones para el sistema de control del vehículo

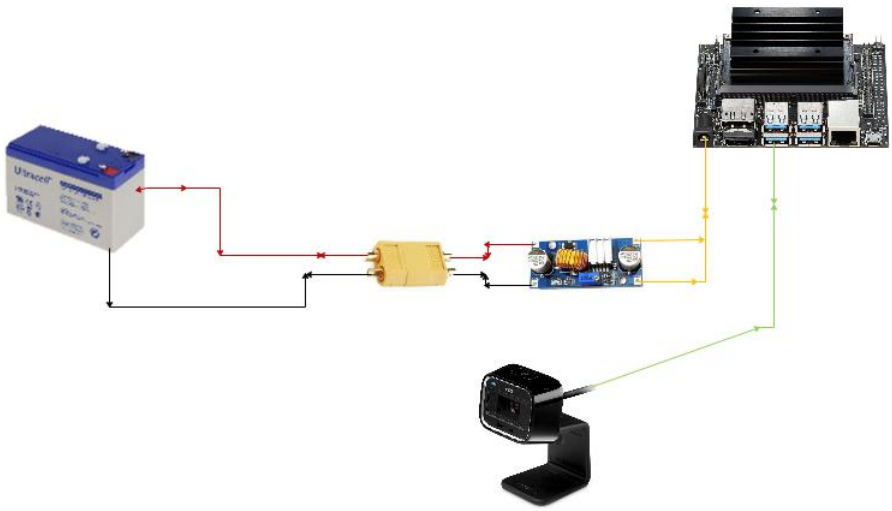


Ilustración 12 Diagrama de conexiones para la inteligencia artificial del vehículo

A continuación, se muestra la configuración de pines tanto para el Raspberry pi 4 y el Jetson nano. En las mismas se puede observar cómo estos dispositivos, aunque poseen la misma cantidad de pines y cuentan con configuraciones muy similares; también son diferenciadas en los pines dedicados para algún sistema especifico (como lo es el Jetson nano

con las configuraciones de GPIO y PWM). Los diagramas de configuración de pines para los dispositivos se muestran en las siguientes imágenes extraídas de sus respectivas datasheets:

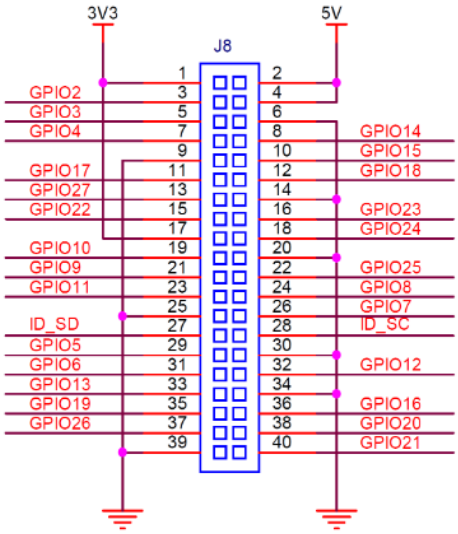


Ilustración 13 Diagrama de conexiones pinera Raspberry Pi 4

SoC GPIO	Linux GPIO #	Alternate Function	Default Function			Default Function	Alternate Function	Linux GPIO #	SoC GPIO
			3.3 VDC	1	2	5 VDC			
PJ.03	75	GPIO	I2C1_SDA	3	4	5 VDC			
PJ.02	74	GPIO	I2C1_SCL	5	6	GND			
PBB.00	216	AUD_CLK	GPIO	7	8	UART1_TXD	GPIO	48	PG.00
			GND	9	10	UART1_RXD	GPIO	49	PG.01
PG.02	50	UART1_RTS	GPIO	11	12	GPIO	I2S0_SCLK	79	PJ.07
PB.06	14	SPI1_SCK	GPIO	13	14	GND			
PY.02	194		GPIO	15	16	GPIO	SPI1_CS1	232	PDD.00
			3.3 VDC	17	18	GPIO	SPI1_CS0	15	PB.07
PC.00	16	SPI0_MOSI	GPIO	19	20	GND			
PC.01	17	SPI0_MISO	GPIO	21	22	GPIO	SPI1_MISO	13	PB.05
PC.02	18	SPI0_SCK	GPIO	23	24	GPIO	SPI0_CS0	19	PC.03
			GND	25	26	GPIO	SPI0_CS1	20	PC.04
PB.05	13	GPIO	I2C0_SDA	27	28	I2C0_CLK	GPIO	18	PC.02
PS.05	149	CAM_MCLK	GPIO	29	30	GND			
PZ.00	200	CAM_MCLK	GPIO	31	32	GPIO	PWM	168	PV.00
PE.06	38	PWM	GPIO	33	34	GND			
PJ.04	76	I2S0_FS	GPIO	35	36	GPIO	UART1_CTS	51	PG.03
PB.04	12	SPI1_MOSI	GPIO	37	38	GPIO	I2S0_DIN	77	PJ.05
			GND	39	40	GPIO	I2S0_DOUT	78	PJ.06

Ilustración 14 Diagrama de conexiones de la pinera del Jetson Nano

La definición de pines ha sido efectuada mediante dos librerías específicas para cada dispositivo: RPi.GPIO (para la Raspberry pi 4) y Jetson.GPIO (para el Jetson nano). Estas librerías necesitan la declaración de la forma de reconocimiento; ya sea BCM (Broadcom SOC channel) o BOARD (numeración dada por el número de pin).

Este tipo de definición varía en el código implementado según el uso de la numeración de pines o del número de GPIO, detallado en la imagen anterior, para cada uno de los sistemas. De esta manera se puede definir la configuración y uso que se le dará a cada uno de los pines. Es decir, tolera la definición del comportamiento de cada pin incluido en el programa instanciado. A continuación, se detalla cada una de las definiciones en la siguiente tabla:

INA y INB	Pin	BCM
INA1	13	27
INB1	19	10
INA2	15	22
INB2	21	9
Triggers		
Frente	3	2
Atrás	5	3
Izquierda	7	4
Derecha	11	17
Pwms		
PWM1	32	12
PWM2	33	13
ECHOS		
Frente	37	26
Atrás	35	19
Izquierda	31	6
Derecha	29	5
VCC y GND		
5V	4	
GND	6	

Tabla 1 Descripción de Pines y BCMs

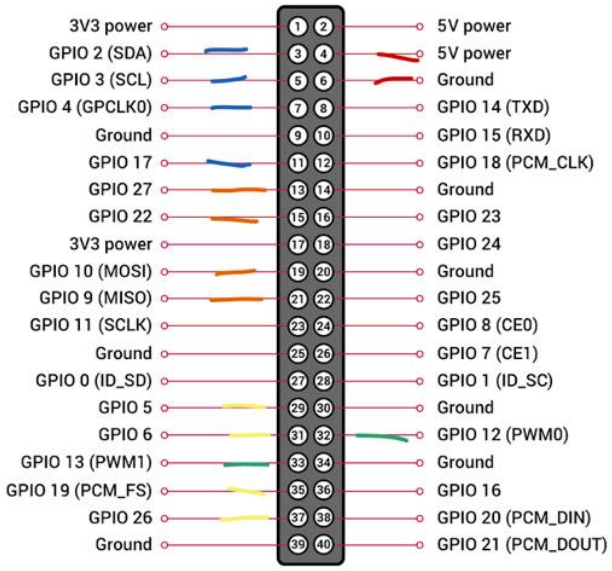


Ilustración 15 Definición visual de los pines

En la Ilustración 15 se han añadido unas líneas de color para enfatizar gráficamente el contenido en la tabla 1, es decir para visualizar mejor la forma en la que se conectan las polarizaciones y los GPIO junto con los PWM en el Raspberry.

Diseño del sistema de control

El diagrama de flujo de decisiones con la lógica del programa controlador diseñado es el siguiente:

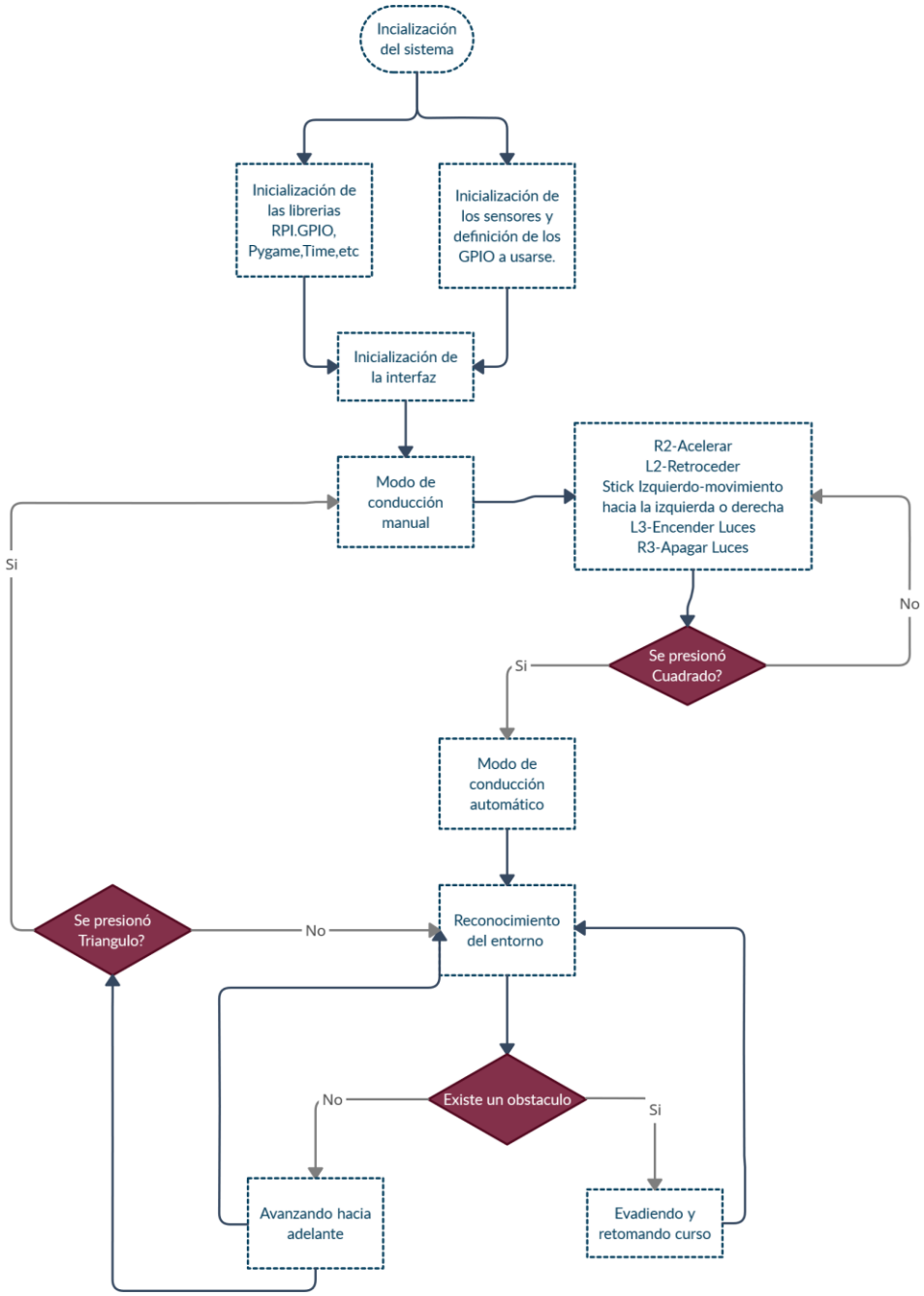


Ilustración 16 Árbol de decisiones para el sistema

Bajo la instanciación de la previa configuración lógica trabajan ambos modos de conducción. Siguiendo la lógica de los siguientes modos y situaciones predefinidas:

Modo Manual:

El modo manual muestra las lecturas de los sensores, pero no afectan al tipo de conducción, donde pulsando L2 se da reversa, con R2 se acelera y con el Stick análogo izquierdo se controla el movimiento de izquierda a derecha del vehículo.

Modo Automático:

Al activar el modo automático el vehículo avanzara por sí solo detectando cualquier elemento que se encuentre frente al sensor determinado; detonando la respuesta al movimiento de este. De esta forma espera la interacción de uno o más objetos frente al vehículo cayendo en alguno de los siguientes casos:

Objeto frente al vehículo: si detecta un objeto frente al vehículo se detendrá y evitará el obstáculo girando hacia la izquierda y luego retomando su posición para avanzar en línea recta.

Objeto frente al vehículo ya sea a la izquierda o la derecha del mismo: Al encontrarse con un objeto frente al vehículo a su izquierda o a la derecha tomará el curso de acción para evitar el objeto. Dependiendo de la localización del obstáculo, el vehículo lo evitará dirigiéndose en la dirección contraria para luego retomando su posición para avanzar en línea recta.

Múltiples objetos provenientes de la dirección frontal al vehículo: Al encontrarse con múltiples objetos limitando la movilidad hacia adelante, el vehículo retrocederá para poder evaluar otra vía de salida.

Múltiples objetos provenientes de las direcciones frontales y posteriores del vehículo: Al encontrarse el vehículo completamente rodeado por obstáculos (encajonado), el avance se interrumpirá y se espera la intervención de un usuario desplegando el mensaje de **“**Vehículo atrapado, espere asistencia”.

A continuación, se detallarán ciertos fragmentos del código implementado con la finalidad de presentar parte de la estructura del sistema:

```
def VCCLUCESON():
    GPIO.output(VCCLED,True)

def VCCLUCESOFF():
    GPIO.output(VCCLED,False)

def distanciaFrente():
    GPIO.output(TRIG_F, True)
    time.sleep(0.00001)
    GPIO.output(TRIG_F, False)

    while GPIO.input(ECHO_F) ==0:

        global pulse_startf

        pulse_startf = time.time()


    while GPIO.input(ECHO_F)==1:

        global pulse_endf

        pulse_endf = time.time()

    pulse_durationf = pulse_endf - pulse_startf

    distancef = pulse_durationf * 17150

    distancef = round(distancef, 2)

      #print ( "Distance:",distance,"cm")
    return distancef


def distanciaDerecha():
    GPIO.output(TRIG_D, True)
    time.sleep(0.00001)
    GPIO.output(TRIG_D, False)

    while GPIO.input(ECHO_D)==0:
          global pulse_start
          pulse_start = time.time()

    while GPIO.input(ECHO_D)==1:
          global pulse_end
          pulse_end = time.time()

    pulse_duration = pulse_end - pulse_start
```

Se define el comportamiento de las funciones para cada acción que se desea para el vehículo por medio de los GPIO.

```
        distance = pulse_duration * 17150

        distance = round(distance, 2)

          #print ( "Distance:",distance,"cm")

        return distance


    def distanciaIzquierda():
         GPIO.output(TRIG_I, True)
         time.sleep(0.00001)
         GPIO.output(TRIG_I, False)

         while GPIO.input(ECHO_I)==0:

               global pulse_start3
               pulse_start3 = time.time()

         while GPIO.input(ECHO_I)==1:

               global pulse_end3
               pulse_end3 = time.time()

         pulse_duration3 = pulse_end3 - pulse_start3

         distance3 = pulse_duration3 * 17150

         distance3 = round(distance3, 2)

         #print ( "Distance:",distance,"cm")

         return distance3

    def distanciaAtras():


    GPIO.output(TRIG_A, True)

    time.sleep(0.00001)
    GPIO.output(TRIG_A, False)

    while GPIO.input(ECHO_A)==0:
          global pulse_starta
          pulse_starta = time.time()

    while GPIO.input(ECHO_A)==1:
          global pulse_enda
          pulse_enda = time.time()

    pulse_durationa = pulse_enda - pulse_starta

    distancea = pulse_durationa * 17150

    distancea = round(distancea, 2)

    #print ( "Distance:",distancea,"cm")

    return distancea

def adelante():
    GPIO.output(INB1,False)
    GPIO.output(INA1,True)

def alto():
    GPIO.output(INA1,False)
    GPIO.output(INA2,False)
    GPIO.output(INB1,False)
    GPIO.output(INB2,False)

def izquierda():
    GPIO.output(INB2,True)
    GPIO.output(INA2,False)
```

En esta sección se definen las funciones que accionan los mecanismos de evasión. Haciendo uso de las señales tanto de entrada como salidas definidas; juntamente con las lecturas para los sensores de proximidad en cada una de las posiciones que se describieron en la ilustración 15.

```
    def derecha():
        GPIO.output(INA1,False)
        GPIO.output(INB1,True)


    def atras():
        GPIO.output(INA2,True)
        GPIO.output(INB2,False)


    def objetofrente():

        for i in range (4): #gira izquierda
            izquierda()
            time.sleep(0.1)

        GPIO.output(INA2,False)
        GPIO.output(INB2,False)

        for i in range(35):  #avanzar
            adelante()
            time.sleep(0.1)

        GPIO.output(INB1,False)
        GPIO.output(INA1,False)


    for i in range(4):  #enderezar
        derecha()
        time.sleep(0.1)

    GPIO.output(INA2,False)
    GPIO.output(INB2,False)

    for i in range(55):  #avanzar
        adelante()
        time.sleep(0.1)

    GPIO.output(INB1,False)
    GPIO.output(INA1,False)

    for i in range(2):  #enderezar
        izquierda()
        time.sleep(0.1)

    GPIO.output(INA2,False)
    GPIO.output(INB2,False)
    GPIO.output(INA1,False)
    GPIO.output(INB1,False)

def objetoizquierda():

    for i in range (4): #gira izquierda
        derecha()
        time.sleep(0.1)

    GPIO.output(INA2,False)
    GPIO.output(INB2,False)

    for i in range(15):  #avanzar
        adelante()
        time.sleep(0.1)
```

Se crean las acciones que tomaran parte para la conducción automática, así como el intervalo que tardaran las mismas.

Visión Artificial

La visión artificial en el proyecto se sustente a en la documentación de Nvidia™. Para la construcción de la inteligencia artificial primero se debe entender el pipeline que utiliza la Jetson nano para el reentrenamiento de la red y la utilización de cada parte en el sistema.

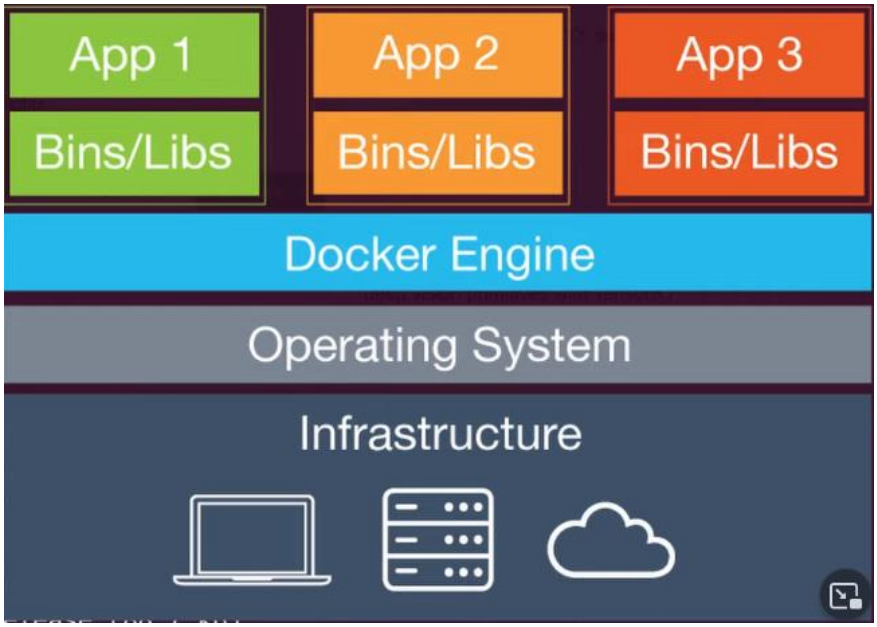


Ilustración 17 Pipeline de trabajo para la Jetson Nano

Esta imagen ejemplifica a grosso modo la manera en que el pipeline funciona:

- El nivel de mayor potencia y donde generalmente se entrena las redes neuronales suelen ser: Computadores de alto rendimiento (o al menos uno mayor a la plataforma a utilizarse)
- El sistema operativo que contiene todo el código a ejecutarse junto con las librerías y prerrequisitos del sistema.
- El Docker un contenedor de todos los prerrequisitos para el sistema en el cual las lecturas, señales y clasificaciones se encuentran en conjunto a todas las librerías y prerrequisitos necesarios sin necesidad de correrlo directamente sobre el sistema operativo evitando errores o posibles incompatibilidades de librerías.

- La aplicación como tal que para este caso es la red neuronal para la visión artificial del sistema.

Para el desarrollo de la visión artificial se utilizará las funciones predefinidas para este tipo de microcomputadores (en las cuales nada más cambian los resultados debido a la performance de cada plataforma particular).

La red Neuronal que se usará será Detectnet V2.0 la cual, es una red Neuronal Convolucional (red neural de percepción multicapa), que de acuerdo con la documentación se estructura basada en Mobilenet (optimizada para dispositivos móviles) definida en el siguiente diagrama de bloques (Sandler, 2019):

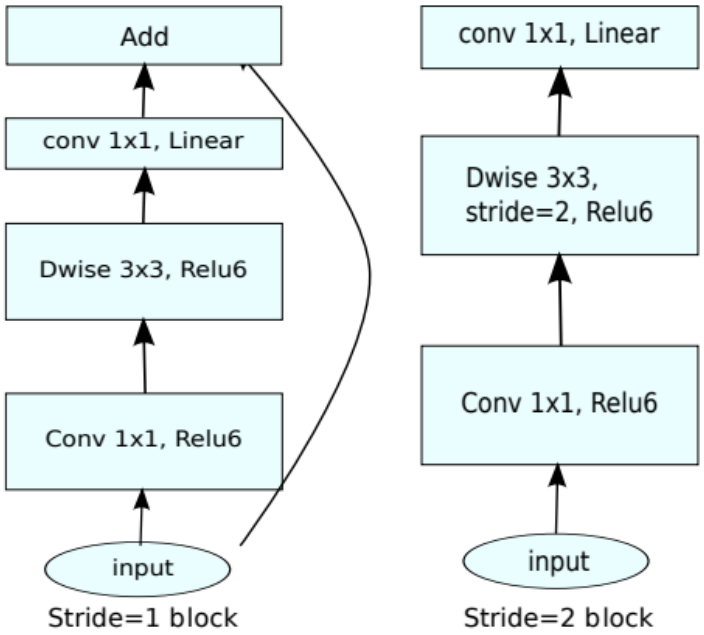


Ilustración 18 Estructura de la Red Neuronal

Se usarán las especificaciones recomendadas, con un número de 18 capas, lo que brindará una precisión limitada pero la suficiente para esta aplicación, de aumentar el

número de capaz su rendimiento decaería de manera sustancial por la cantidad de cálculos necesarios, pero la detección sería mucho más fina y se tendría un menor rango de error.

La validación se realiza mediante la propia documentación de la red que utiliza sus propios comandos dentro de Docker para verificar no solo su correcto entrenamiento y funcionamiento sino el porcentaje de acierto de esta que se detalla en esta documentación.

https://docs.nvidia.com/metropolis/TLT/tlt-user-guide/text/object_detection/detectnet_v2.html#evaluating-the-model

Siguiendo la guía de Rusty de Jetson-inference para estos sistemas, se tiene que descargar el repositorio con el siguiente comando en la carpeta de nuestra elección:

git clone --recursive https://github.com/dusty-nv/jetson-inference

luego se debe dirigir al apartado de Docker para inicializarlo con el comando:

docker/run.sh

del cual se desplegará el sistema de descarga de los modelos para las redes neuronales previstas para el sistema en el que se puede seleccionar de acuerdo con las necesidades:

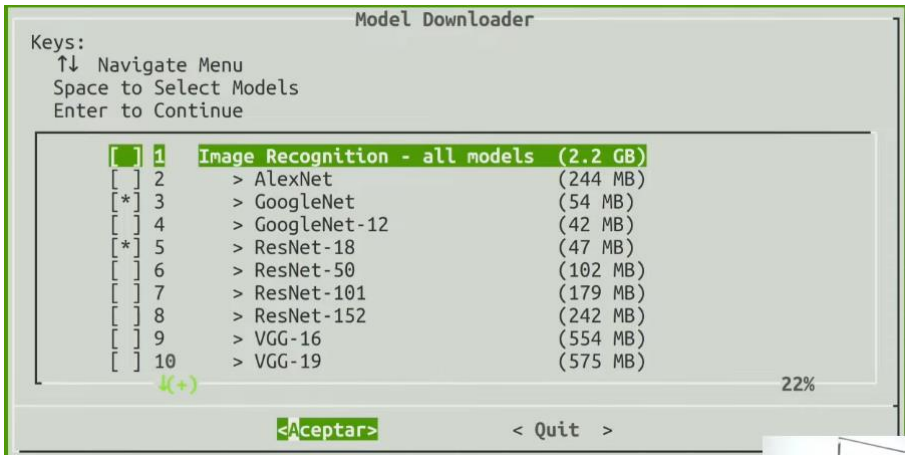


Ilustración 19 Ventana de descarga para los modelos disponibles

Al terminar la descarga en el sistema, el Docker se ejecutará de manera inmediata, mostrando los dispositivos conectados que pueden usarse dentro del Docker, que para este caso se trata de una webcam

Ilustración 20 Docker en la ventana de CMD

Al ir al apartado de Python y de ejemplos se puede ejecutar los ejemplos que en este caso es la red de detectnet (por ser una red robusta y fácil de entrenar para el sistema), al cual se modificará con los siguientes comandos de acuerdo con la documentación de Nvidia (NVIDIA, NVIDIA Jetson Linux Driver Package Software Features, 2021):

Instalación y descarga de mobilenet

cd Documentos/Tesis/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

Declaración de la ubicación para la descarga y referencia en los archivos del sistema, para la ruta de descarga de los archivos necesarios para la red neuronal e instalación de los requisitos previos del sistema.

Conteo de imágenes

python3 open_images_downloader.py --stats-only --class-names "Bicycle,Car,Land vehicle,Motorcycle,Traffic light, Traffic sign,Bus,Person,Stop sign" --data=data/Tesis

Descargador de imágenes con limitación

python3 open_images_downloader.py --max-images=3000 --class-names "Bicycle,Car,Land vehicle,Motorcycle,Traffic light,Traffic sign,Bus,Person,Stop sign" --data=data/Tesis

Descargador de imagenes

python3 open_images_downloader.py --class-names "Bicycle,Car,Land vehicle,Motorcycle,Traffic light,Traffic sign,Bus,Person,Stop sign" --data=data/Tesis

Codigo para entrenar

python3 train_ssd.py --data=data/Tesis --model-dir=models/Tesis --batch-size=2 --workers=1 --epochs=60

Conteo de imagenes

En esta sección mediante el Downloader la cantidad de imágenes de prueba y muestra que existen en el repositorio para su posterior descarga, ubicándolos con el nombre dentro del mismo.

Descargador de imágenes con limitación

Dada la limitada capacidad de almacenamiento y de procesamiento del microcomputador limitaremos la descarga a 3000 imágenes.

Descargador de imágenes

Este elemento es el mismo código de descarga, pero sin ninguna limitante para el mismo lo que mejoraría la precisión si se tuviera mayor almacenamiento y capacidad de procesamiento.

Código para entrenar

Esta es la sección más importante donde configuraremos las especificaciones del entrenamiento el mismo que se encuentra limitado, pero sea suficiente para nuestra aplicación.

Los comandos mostrados son empleados para el reentrenamiento de la red neuronal en base a la documentación proporcionada por Nvidia. “Tesis” es la carpeta raíz que se utilizará, donde se limitarán los recursos de procesamiento para el sistema a solo 3000 imágenes de muestra de la librería Open Images Dataset V6. El uso de esta librería otorgará una buena aproximación para las necesidades requeridas. Sin embargo, se podría incrementar la precisión del modelo al usar más imágenes y un mayor número de epochs (entrenamiento de la red neuronal con todos los datos por un ciclo). Para conseguir este incremento, se debe realizar un entrenamiento con una duración de tiempo mayor.

La validación del modelo se realizó con el comando “Evaluate”descrito en su documentación. Sin embargo, dado que la única “Flag” es “validation succesfull”, y no posee una escala o una variable fácilmente identificable; se tuvo que validarla por método de prueba y error, hasta conseguir verificar que el entrenamiento ya no arrojaba ningún falso positivo. En este punto del análisis el nivel de certeza en el sistema era superior al ochenta y ocho por ciento en cada una de las clases.

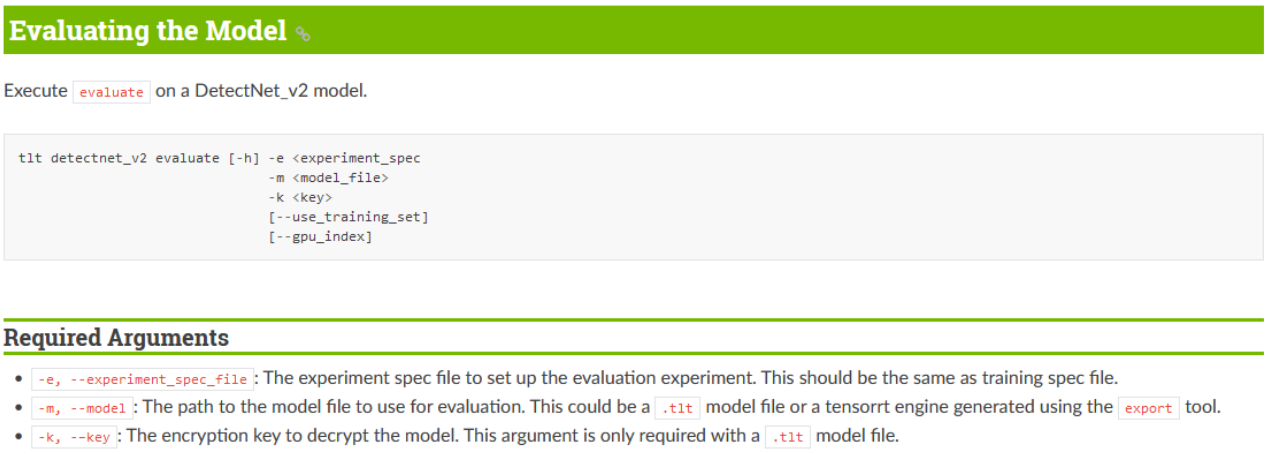


Ilustración 21 Evaluación de Detecnet

Una vez reentrenada la red se probó con elementos varios para verificar su correcto funcionamiento y posible toma de datos (la cual muestra sus resultados en la

sección de Pruebas de Inteligencia artificial en el vehículo). Así mismo se observa la manera en que el desempeño de la red neuronal difiere al empezar el reentrenamiento (ilustración 23) a cuando este se finalizó y valido (ilustración 28). Dado que el primer acercamiento a esta red lo realizamos en una computadora de mayor nivel de procesamiento podemos observar en la parte superior la tasa de refresco que en un principio era de 27 a 30 FPS (Frames per Second), para luego al migrarla al microcomputador reduzca su rendimiento hasta los 12 a 15 FPS.

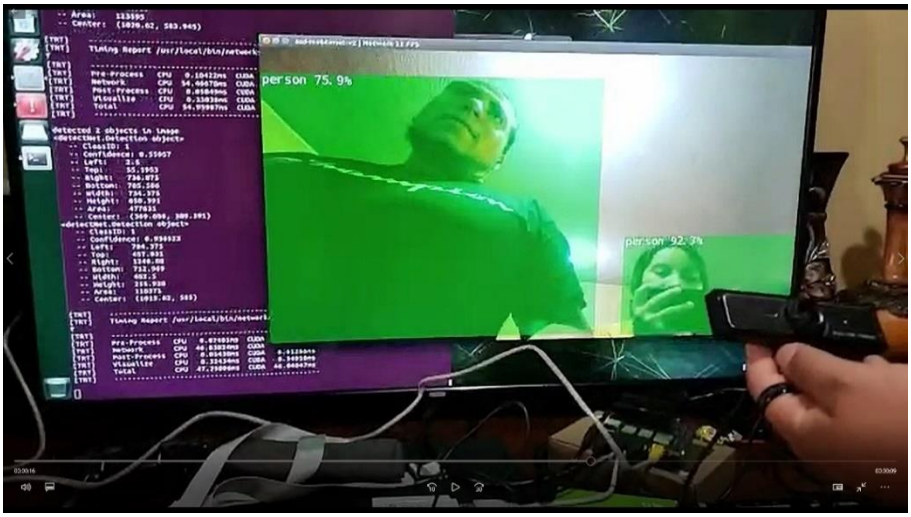
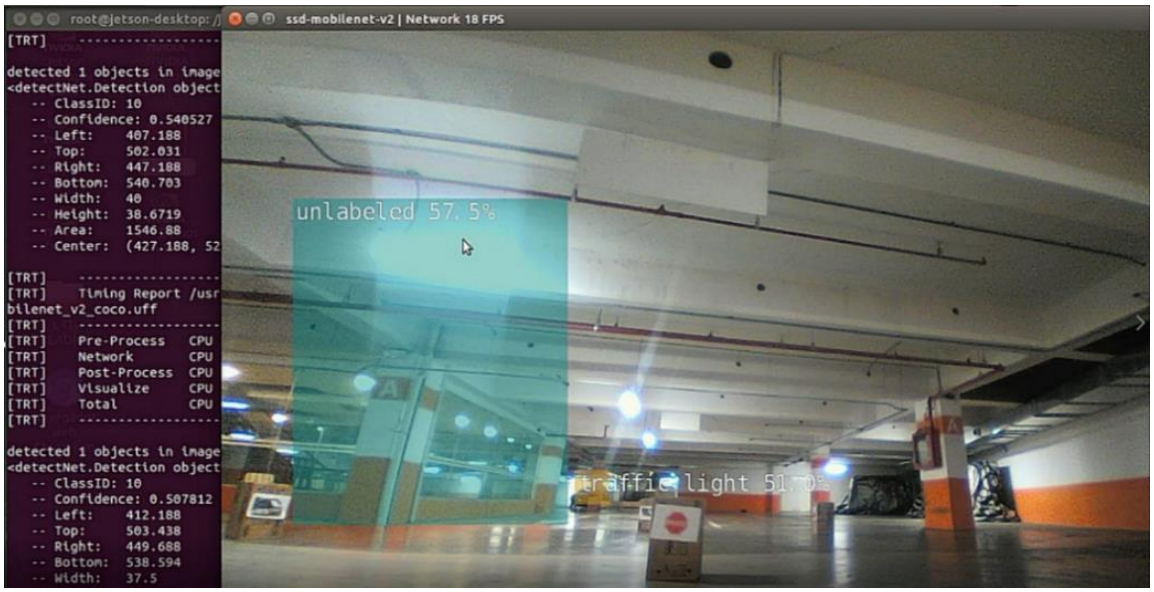


Ilustración 22 Primer Acercamiento a la red neuronal a utilizarse



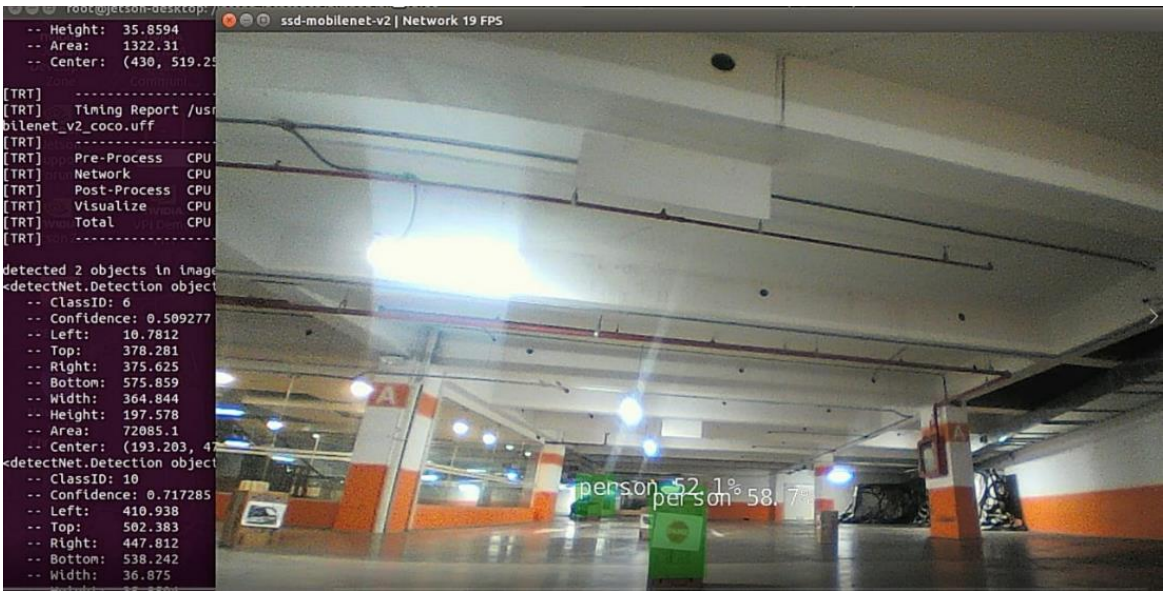


Ilustración 23 Proceso de Validación de la red neuronal en el espacio de trabajo.

RESULTADOS FINALES

Armado final

Ilustración 24 Instalación y prueba de sistemas

En el armado final como se aprecia en la ilustración 25 se montaron las placas y drivers, se colocó el Raspberry en un lugar específico para realizar las conexiones entre los sensores y driver que controla los motores. También se aprecian los voltímetros colocados para medir el voltaje de las baterías que alimentan los motores y el Raspberry, la utilidad de estos es para evitar una descarga profunda que dañe las baterías. En la última imagen se ve una vista completa de todo el vehículo en donde está la pantalla del Raspberry que permite monitorear los procesos que se están realizando.

Una vez implementadas las modificaciones del Hardware de forma de trabajar con seguridad y agilidad, se realizó la programación del sistema en lenguaje Python (por su cercanía a Linux). Esto permitió no solo desarrollar nuevas habilidades de programación en un lenguaje desconocido sino utilizar las siguientes librerías especializadas que simplificaron notablemente el trabajo (especialmente la interfaz gráfica) (Python, n.d.):

RPi.GPIO: Librería que controla los GPIO de la microcomputadora

Pygame: Librería para desarrollo de aplicaciones multimedia con amplia extensión a aplicaciones que requieran una GUI.

Time: Librería para el control y definición de unidades de tiempo y funciones cronometradas.

Numpy: Librería que controla la utilización de cálculos a manera de arreglos para facilitar las operaciones matemáticas complejas.

Para la unidad de CPU se utilizó principalmente al Raspberry pi 4, de aplicación general y uso extensivo en la comunidad afín a los microcomputadores. Para este sistema se empleó la librería de pygame como base tanto para la interfaz visual como para la lectura de los comandos emitidos en la palanca de PS4, de esta forma lo primero fue entender la manera en que pygame tomaba los valores del mando, por lo que se utilizó un código propio para la calibración y declaración de las teclas del mando a utilizar.

Después de esto se definieron tanto los pines de entrada y de salida del dispositivo. Esta configuración se realizó con sumo cuidado, de tal manera que se puede reemplazar el Raspberry pi 4 por el Jetson nano sin necesidad de recableado o de reprogramación del sistema.

Dado que los motores no son los más precisos en cuanto a mediciones o ángulos de movimiento, cada acción a ejecutar se definió mediante iteraciones previamente

cronometradas para confirmar su correcto funcionamiento en un ambiente controlado. Una vez que se logró ejecutar el movimiento deseado del vehículo con el uso del control de PS4, se añadieron los sensores de proximidad para detectar cualquier objeto alrededor del vehículo, los sensores empleados tienen un alcance de 5 metros por lo que se los programo para que detecten objetos a un metro con veinte centímetros con lo cual se le da tiempo suficiente al Raspberry para reaccionar y así tomar una decisión apropiada.

Estos sensores ultrasónicos debido a su naturaleza utilizan una emisión de señal que solo puede interpretarla este dispositivo y de esta manera determina la distancia del objeto en el cual rebota la señal de ultrasonido. Estas señales se calibraron en relación con las dimensiones del vehículo para conseguir que éste reaccione en tiempo real ante la potencial amenaza de la presencia de obstáculos a su alrededor. El código se estructuró para monitorear los sensores secuencialmente, es decir que la primera interacción será con un objeto de frente, para luego analizar el mejor curso de acción dependiendo de la proximidad de la obstrucción hacia el sensor de la izquierda o de la derecha. De darse el caso de que los tres sensores frontales estén activados el vehículo retrocederá. En última instancia, si los cuatro sensores llegaran a activarse simultáneamente, el sistema emitirá un mensaje solicitando asistencia por encontrarse atrapado.

El sistema de navegación se complementa con la inteligencia artificial diseñada en una red de Detecnet. Se trata precisamente de la red neuronal convolucional diseñada para la detección o clasificación de objetos comentada en la sección de “Visión Artificial”, optimizada para el proyecto, utilizando imágenes de una base de datos de Open Images Dataset para el reentrenamiento (es decir modificar el funcionamiento por defecto de la red para optimizar y definir de manera más precisa el funcionamiento de la misma, incluyendo

todas sus características pero reestructurada para la aplicación realizada) de la red neuronal de la cual ya se tenía una base.

Esto se realizó de manera progresiva y con bastante esmero dado el limitado poder de procesamiento con el que cuenta estos microcomputadores y la falta de documentación actual, debido a que solo se encontraba documentación de versiones antiguas que no eran de mucha ayuda para el proyecto.

Se tardo alrededor de dos a tres semanas en una lectura, procesamiento y reconocimiento de las imágenes en el sistema, para este proceso una recomendación es realizar el reentrenamiento de la red neuronal en una computadora con potencia de procesamiento mucho mayor para mejorar el tiempo de reentrenamiento y aumentar la precisión de reconocimiento.

Uno de los problemas importantes que limitó el alcance de este proyecto fue la escasa información sobre la utilización del Header de 40 pines de la Jetson nano. A pesar de que el código no mostró errores y su programación aparentaba ser satisfactoria, las medidas en el osciloscopio mostraron la presencia únicamente de ruido, lo cual impidió que la Jetson nano reemplazara al Raspberry pi 4 como procesador central. Puesto que el controlador de motores y el resto de los procesos no reconocieron ninguna señal de entrada proveniente de la Jetson nano, se optó finalmente por separar la utilización de la red neuronal a cargo de la Jetson nano del módulo de control del vehículo a cargo del Raspberry pi 4.

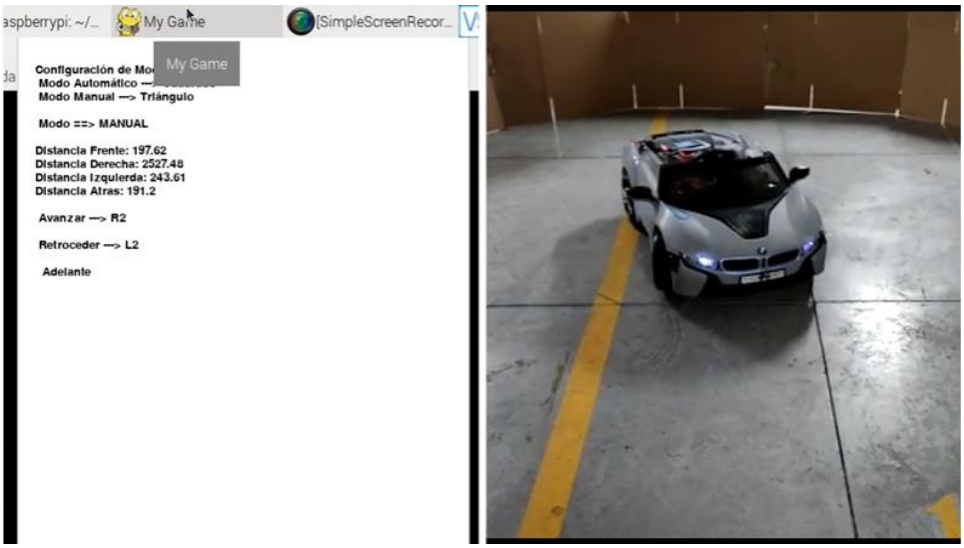
Funcionamiento específico del control del sistema.

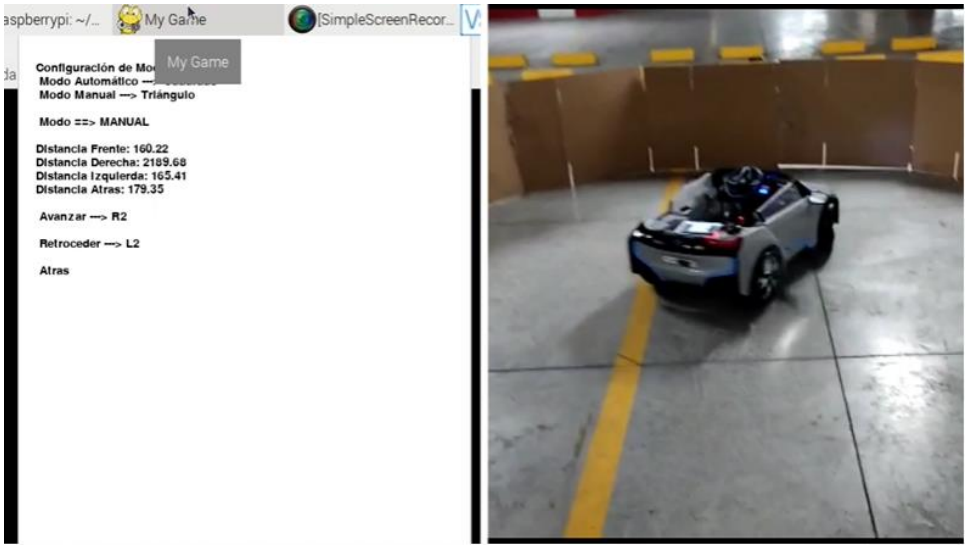
El sistema se desarrolló en lenguaje de Python por lo que para inicializar el sistema dentro de la Raspberry pi 4 se debe localizar el archivo de control del vehículo llamado “BB-

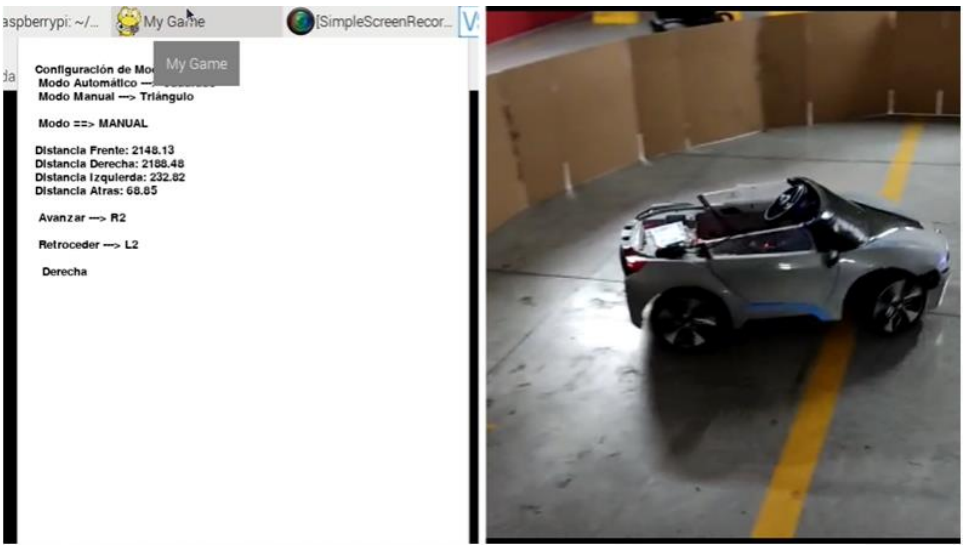
Eleven” una vez inicializado en la consola de comandos o Python Genie (para controlar de mejor manera el funcionamiento del sistema), se desplegará una ventana que indica no solo los botones a presionar para cada una de las acciones, sino también el modo en el que se encuentra el sistema. El vehículo siempre inicializa por seguridad en el modo manual, que se comenta más adelante. Para cambiar a modo automático se deberá presionar la tecla “cuadrado”.

Pruebas modo Manual

Para las pruebas que se realizaron en el modo manual se construyó una pista con cartones la cual permitió poder ejecutar movimientos específicos que pusieron a prueba tanto el código como la dirección del vehículo que fue calibrada para que no tenga giros bruscos y permita un manejo más sencillo. En la parte izquierda se puede apreciar la interfaz gráfica que se tiene en donde se muestra el funcionamiento de los sensores y la acción que se está realizando que en este caso es adelante, atrás, izquierda y derecha.




My Game
[SimpleScreenRecor...
My Game
Modo Manual ---> Triángulo
Modo ==> MANUAL
Distancia Frente: 160.22
Distancia Derecha: 2189.68
Distancia Izquierda: 165.41
Distancia Atras: 179.35
Avanzar ---> R2
Retroceder ---> L2
Atras


My Game
[SimpleScreenRecor...
My Game
Modo Manual ---> Triángulo
Modo ==> MANUAL
Distancia Frente: 2148.13
Distancia Derecha: 2188.48
Distancia Izquierda: 232.82
Distancia Atras: 68.85
Avanzar ---> R2
Retroceder ---> L2
Derecha

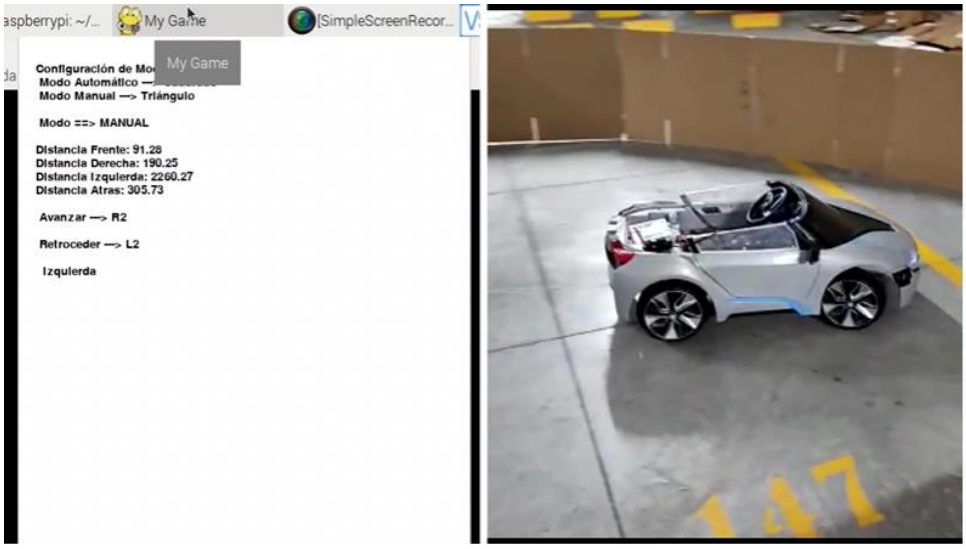
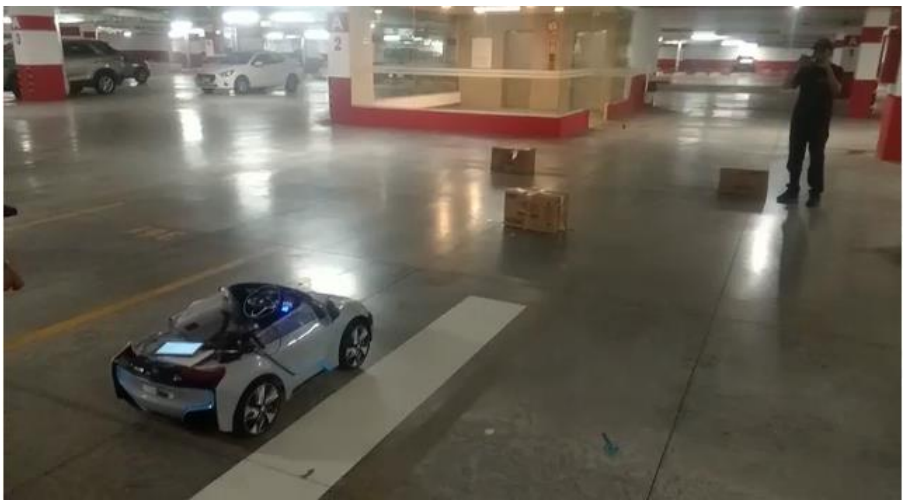
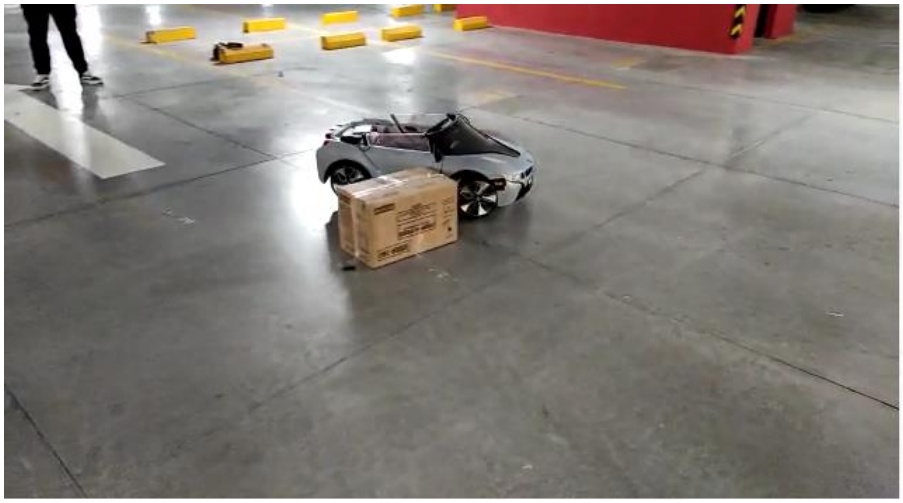


Ilustración 25 imágenes capturadas de la interfaz y conducción a tiempo real del sistema

Pruebas modo Automático

Las pruebas que se realizaron fueron con cajas que servían como obstáculo las cuales el vehículo pudo esquivar, la prueba más importante fue la de un objeto inesperado es decir cuando una persona se para delante del carrito de improvisto este puede reaccionar y esquiva sin problemas a la persona, otro aspecto que se pudo ver es que tras esquivar a la persona el vehículo siguió su ruta hasta que la finalizó estacionándose.





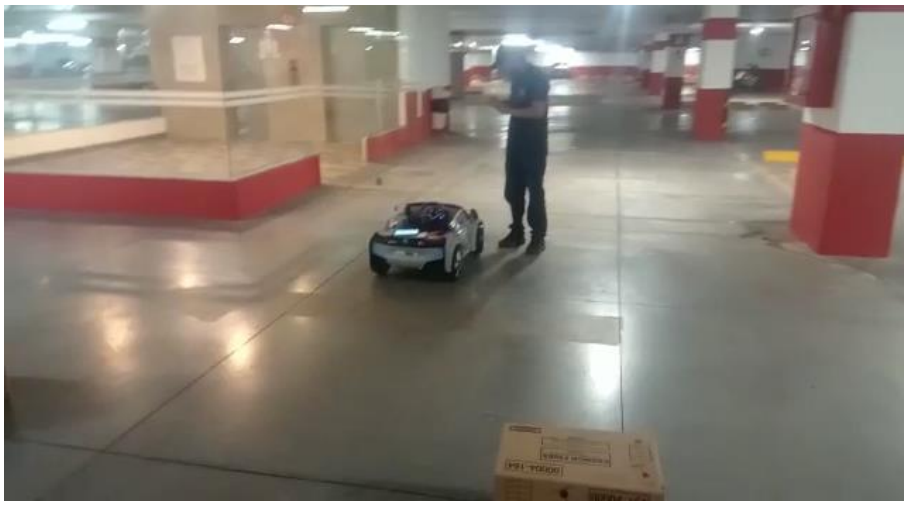


Ilustración 26 Resultados del senado y la conducción automática

Pruebas Inteligencia artificial en el vehículo

A diferencia de las anteriores pruebas en donde se ocupó el Raspberry Pi4, para la inteligencia artificial se empleó el Jetson Nano. La inteligencia artificial que se entrenó podía detectar una cantidad enorme de objetos, pero para evitar una saturación se limitó a nueve clases (Bicycle,Car,Land vehicle,Motorcycle,Traffic light,Traffic sign,Bus,Person,Stop sign) las cuales se consideraron fundamentales en cualquier red vial. Para la prueba se ocuparon cajas en las que se colocó diferentes imágenes, entre las cuales tenemos: una señal de pare, un bus, una motocicleta, un vehículo y también se detectó a personas.

La cámara que se empleó poseía un autoenfoque el cual ayudaba al momento de detectar los objetos, pero a su vez también limitó la tasa de refresco de las tramas medidas en tramas por segundo (FPS), la cual no superó los 15 FPS por lo que se originó un retraso en la imagen.

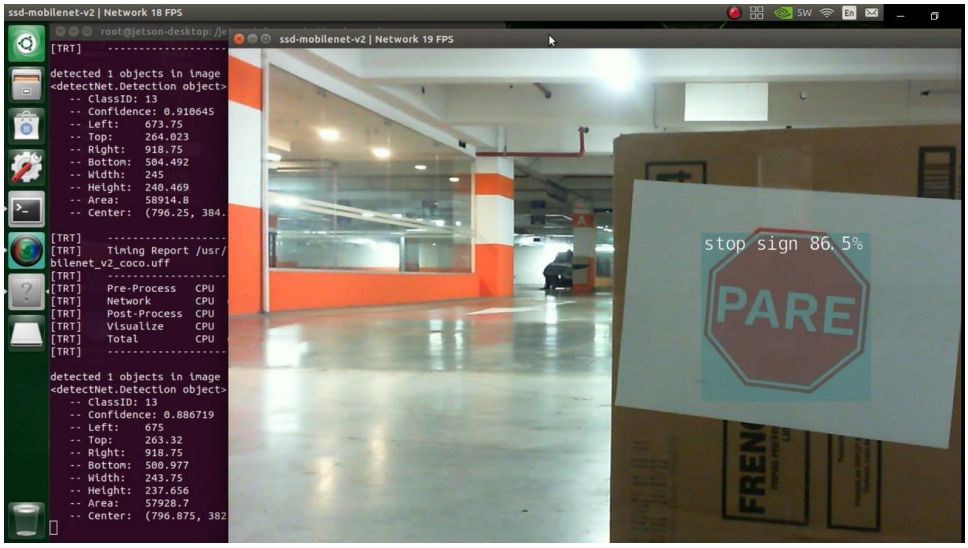


Ilustración 27 Reconocimiento de una señal de "PARE"

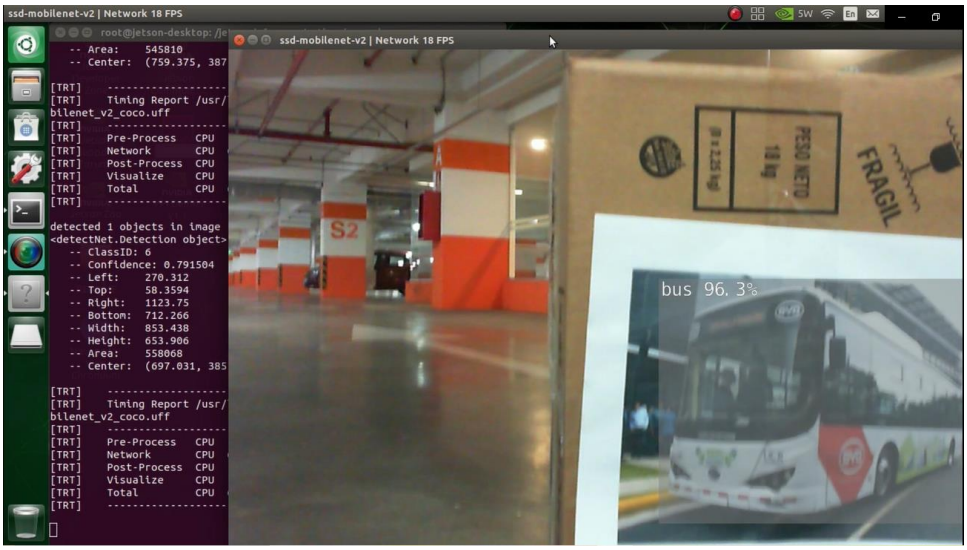


Ilustración 28 Reconocimiento de un Autobús

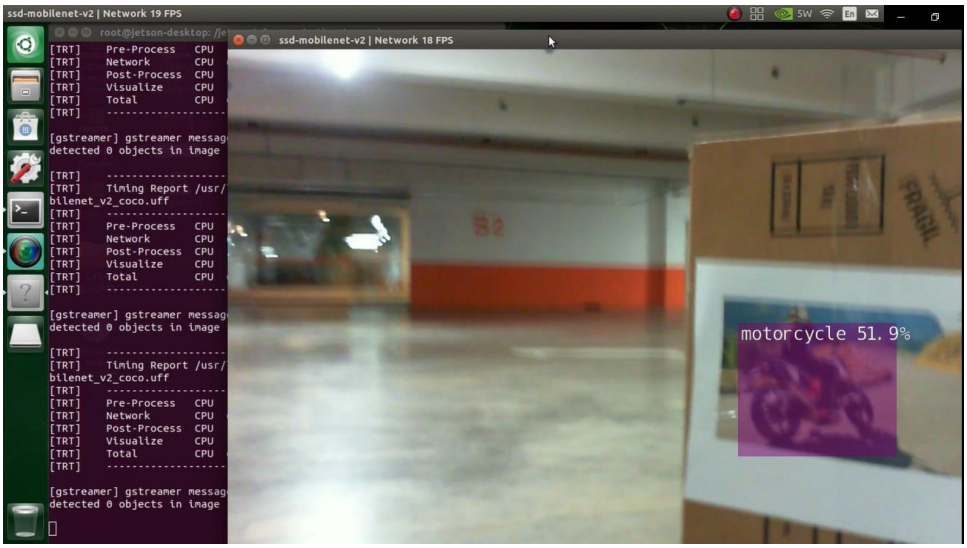


Ilustración 29 Reconocimiento de una motocicleta

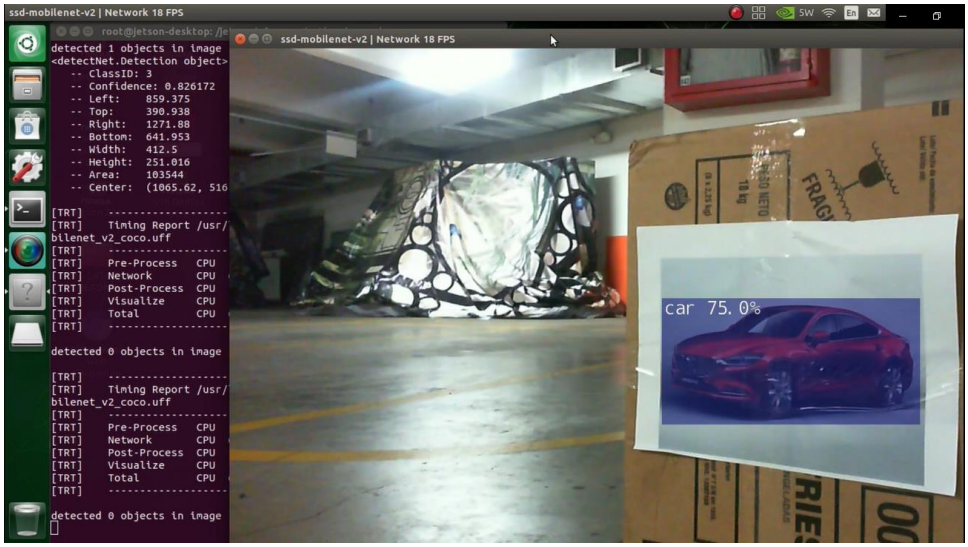


Ilustración 30 Reconocimiento de un Automóvil.

Conclusiones

Se concluye que el Raspberry, aunque sirve como un centro de control no es muy veloz a la hora de tener que reaccionar varias señales a la vez, esto se pudo notar al momento de hacer pruebas con el control manual en el cual hay muchos objetos alrededor.

Los problemas mecánicos afectaron considerablemente el avance del proyecto, cambios como el motor delantero representaron un tiempo invertido en la calibración para el giro.

El Jetson Nano a pesar de ser más potente en procesamiento que el Raspberry, este tiene un serio problema en lo que son las señales PWM que eran necesarios para controlar los motores.

El vehículo, aunque originariamente era solo un juguete fue rediseñado para facilitar la realización de diferentes pruebas y añadir satisfactoriamente la IA. El rediseño efectuado permitirá realizar nuevos desarrollos y escalar progresivamente la complejidad, funcionalidades y aplicaciones del sistema.

Escalabilidad y Recomendaciones

- Incorporación de subsistemas de monitoreo con el sistema controlador principal utilizando librerías como Kivy las cuales muestran una mejora tanto para el rendimiento como la visualización.
- Encontrar una solución para interpretar las señales provistas de la visión artificial con un dispositivo que no cuente con open CV. Esta recomendación se la realiza puesto que se consideró el uso de ROS dentro de la propuesta original planteada. Sin embargo, se la tuvo que desestimar dado que las señales de ambos microcomputadores no podían comunicarse de manera adecuada por la utilización de Docker en el Jetson Nano y la falta de open CV en la Raspberry pi 4.
- Utilización de un hardware con mayor potencia de procesamiento que cuente con una GPU para la utilización sincrónica de la red neuronal, ya sea con un clúster (conexión de dos o más dispositivos iguales) de Jetson nano o sus periféricos de mayor performance.
- Actualización de los sensores de proximidad con sensores que devuelvan lecturas más precisas y rápidas para aun control digital más preciso facilitando la programación y restructurando el porcentaje de error en los mismos.
- Cambio a una placa controladora central subdivida para las diferentes secciones del proyecto (es decir: alimentación, control, procesador central, censado, etc).
- Reemplazo de motores por servomotores para una adaptación más fácil y retornos de trayectoria más precisos.
- En cuanto a la implementación de ROS este solo es una buena idea siempre y cuando se tenga un clúster ya que de forma individual no hay necesidad de ocuparlo, para el

proyecto si se desea usar el Jetson nano y el Raspberry en conjunto si fuera beneficioso.

Referencias

Brico Geek. (s.f.). *Raspberry Pi 4 Modelo B 4GB RAM*.

Contaval. (18 de febero de 2016). *Contaval*. Obtenido de ¿Qué es la visión artificial y para qué sirve?: https://www.contaval.es/que-es-la-vision-artificial-y-para-que-sirve/

ELECTRONICS, U. (s.f.). *Nvidia Jetson Nano Developer Kit*.

ErikDotDev. de 2020). Configure and Deploy "Intelligent Video Analytics" to IoT Edge Runtime on NVIDIA Jetson.

Gasser, T. M. (2013). *Legal consequences of an increase in vehicle automation.* Bergisch Gladbach: Bundesanstalt für Straßenwesen.

hangzhang. (06 de Oct de 2013). Obtenido de how to edit the ~/.bashrc file: https://answers.ros.org/question/87866/how-to-edit-the-bashrc-file/

Hardwick, M. (20 de Mar de 2017). *Medium*. Obtenido de Simple Lane Detection with OpenCV: https://medium.com/@mrhwick/simple-lane-detection-with-opencv-bfeb6ae54ec0

JetsonHacks. (28 de November de 2019). *JetsonHacks*. Obtenido de Jetson Nano – Even More Swap: https://www.jetsonhacks.com/2019/11/28/jetson-nano-even-more-swap/

Julián, G. (30 de Diciembre de 2014). *Xataka*. Obtenido de Las redes neuronales: qué son y por qué están volviendo : https://www.xataka.com/robotica-e-ia/las-redes-neuronales-que-son-y-por-que-estan-volviendo

Lin, D. C.-E. (17 de Dec de 2018). *Towards data science*. Obtenido de Tutorial: Build a lane detector: https://towardsdatascience.com/tutorial-build-a-lane-detector-679fd8953132

LUCA. (24 de Febrero de 2019). *¿Qué es Python?* Obtenido de https://web.archive.org/web/20200224120525/https://luca-d3.com/es/data-speaks/diccionario-tecnologico/python-lenguaje

maximechevalierb. (27 de Marzo de 2019). *Python GPIO support?* Obtenido de https://forums.developer.nvidia.com/t/python-gpio-support/72092

NVIDIA. (27 de October de 2021). Obtenido de Quickstart Guide for Deepstream: https://docs.nvidia.com/metropolis/deepstream/dev-guide/text/DS_Quickstart.html#run-deepstream-app-the-reference-application

NVIDIA. (27 de Oct de 2021). *Docker Container for Jetson*. Obtenido de https://docs.nvidia.com/metropolis/deepstream/dev-guide/text/DS_docker_containers.html#a-docker-container-for-jetson

NVIDIA. (3 de Agosto de 2021). *NVIDIA Jetson Linux Driver Package Software Features*. Obtenido de Configuring Jetson Expansion Header: https://docs.nvidia.com/jetson/l4t/index.html#page/Tegra%20Linux%20Driver%20Package%20Development%20Guide/hw_setup_jetson_io.html

nv-zhliu. (27 de Oct de 2021). *How To* --. Obtenido de Write a DeepStream Application in Python: https://github.com/NVIDIA-AI-IOT/deepstream_python_apps/blob/master/HOWTO.md

Pygame. (s.f.). *CompileUbuntu*. Obtenido de https://www.pygame.org/wiki/CompileUbuntu?parent=

Pygame. (s.f.). *GettingStarted*. Obtenido de Pygame Installation¶: https://www.pygame.org/wiki/GettingStarted

Raspberry Pi 4 Model B. (21 de Junio de 2019). Obtenido de https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf

Sahir, S. (25 de Jan de 2019). *Towards data science*. Obtenido de Canny Edge Detection Step by Step in Python — Computer Vision: https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

SudKul. (25 de Oct de 2021). *Finding Lane Lines on the Road*. Obtenido de https://github.com/udacity/CarND-LaneLines-P1

ANEXOS`

RobotShop INC.

La robotique à votre service

Banebots RS-550 Motor - 12V Specifications

Performance

Operating v	6v – 14.4v
Nominal v	12v
No Load RPM	11780
No Load A	0.8A
Stall Torque	46.9 oz-in / 331.2 mN-m
Stall Current	35A
Kt	1.34 oz-in/A / 9.5 mN-m/A
Kv	982 rpm/V
Efficiency	73%
RPM - Peak Eff	10275
Current - Peak Eff	5.1A

Physical

Weight	7.8 oz (221g)
Length - for motor	2.24 in (57mm)
Diameter (with flux ring)	1.52 in (38.5mm)
Diameter (no flux ring)	1.41 in (35.8mm)
Shaft Diameter	0.12 in (3.2mm)
Shaft Length	0.3 in (7.6mm)
Mounting Screws (2)	M3

TEXAS INSTRUMENTS
LM2596
SNVS124F – NOVEMBER 1999 – REVISED APRIL 2021

LM2596 SIMPLE SWITCHER® Power Converter 150-kHz 3-A Step-Down Voltage Regulator

1 Features

- New product available: LMR33630 36-V, 3-A, 400-kHz synchronous converter
- 3.3-V, 5-V, 12-V, and adjustable output versions
- Adjustable version output voltage range: 1.2-V to 37-V ±4% maximum over line and load conditions
- Available in TO-220 and TO-263 packages
- 3-A output load current
- Input voltage range up to 40 V
- Requires only four external components
- Excellent line and load regulation specifications
- 150-kHz Fixed-frequency internal oscillator
- TTL shutdown capability
- Low power standby mode, I_Q, typically 80 µA
- High efficiency
- Uses readily available standard inductors
- Thermal shutdown and current-limit protection
- Create a custom design using the LM2596 with the WEBENCH Power Designer

2 Applications

- Appliances
- Grid infrastructure
- EPOS
- Home theater

3 Description

The LM2596 series of regulators are monolithic integrated circuits that provide all the active functions for a step-down (buck) switching regulator, capable of driving a 3-A load with excellent line and load regulation. These devices are available in fixed output voltages of 3.3 V, 5 V, 12 V, and an adjustable output version.

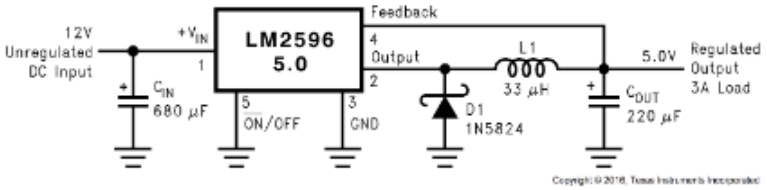
Requiring a minimum number of external components, these regulators are simple to use and include internal frequency compensation, and a fixed-frequency oscillator.

The LM2596 series operates at a switching frequency of 150 kHz, thus allowing smaller sized filter components than what would be required with lower frequency switching regulators. Available in a standard 5-pin TO-220 package with several different lead bend options, and a 5-pin TO-263 surface mount package.

The new product, LMR33630, offers reduced BOM cost, higher efficiency, and an 85% reduction in solution size among many other features. Start WEBENCH Design with the LMR33630.

PART NUMBER	PACKAGE(1)	BODY SIZE (NOM)
LM2596	TO-220 (5)	14.986 mm × 10.16 mm
	TO-263 (5)	10.10 mm × 8.89 mm

(1) For all available packages, see the orderable addendum at the end of the data sheet.



(Fixed Output Voltage Versions)

Typical Application

VNH2SP30-E

Automotive fully integrated H-bridge motor driver

Features

Type	$R_{DS(on)}$	I_{out}	V_{ccmax}
VNH2SP30-E	19mΩ max (per leg)	30A	41V

- 5V logic level compatible inputs
- Undervoltage and overvoltage shut-down
- Overvoltage clamp
- Thermal shut down
- Cross-conduction protection
- Linear current limiter
- Very low stand-by power consumption
- PWM operation up to 20 kHz
- Protection against loss of ground and loss of V_{CC}
- Current sense output proportional to motor current
- Package: ECOPACK®

MultiPowerSO-30™

Description

The VNH2SP30-E is a full bridge motor driver intended for a wide range of automotive applications. The device incorporates a dual monolithic high side driver and two low side switches. The high side driver switch is designed using STMicroelectronic's well known and proven proprietary VIPower™ M0 technology which permits efficient integration on the same die of a true Power MOSFET with an intelligent signal/protection circuitry.

The low side switches are vertical MOSFETs manufactured using STMicroelectronic's proprietary EHD ('STripFET™') process. The three die are assembled in the MultiPowerSO-30 package on electrically isolated leadframes. This package, specifically designed for the harsh automotive environment offers improved thermal performance thanks to exposed die pads. Moreover, its fully symmetrical mechanical design allows superior manufacturability at board level. The input signals IN_A and IN_B can directly interface to the microcontroller to select the motor direction and the brake condition. The $DIAG_A/EN_A$ or $DIAG_B/EN_B$, when connected to an external pull-up resistor, enable one leg of the bridge. They also provide a feedback digital diagnostic signal. The normal condition operation is explained in *Table 12: Truth table in normal operating conditions on page 14*. The motor current can be monitored with the CS pin by delivering a current proportional to its value. The speed of the motor can be controlled in all possible conditions by the PWM up to 20 kHz. In all cases, a low level state on the PWM pin will turn off both the LS_A and LS_B switches. When PWM rises to a high level, LS_A or LS_B turn on again depending on the input pin state.

Table 1. Device summary

Package	Order codes	
	Tube	Tape and Reel
MultiPowerSO-30	VNH2SP30-E	VNH2SP30TR-E

UL 7-12
12V 7AH
General

Ultracell®
'Quality in Every Language'

UL7-12

Physical Specification

Part Number:	**UL7-12**
Length:	**151 ± 2 mm (5.94 inches)**
Width:	**65 ± 2 mm (2.56 inches)**
Container Height:	**93.5 ± 2 mm (3.68 inches)**
Total Height (with terminal):	**99 ± 2 mm (3.90 inches)**
Approx Weight:	**Approx 2.05kg (4.52lbs)**

Specifications

	Normal Voltage	12V
	Normal Capacity (20HR)	7AH
Terminal Type	Standard Terminal	F1
	Optional Terminal	F2
Container Material	Standard Option	ABS
	Flame Retardant Option (FR)	UL94:VO
Rated Capacity	7.00 AH/0.350A	(20hr, 1.80V/cell, 25°C / 77°F)
	6.51 AH/0.653A	(10hr, 1.80V/cell, 25°C / 77°F)
	6.00 AH/1.20A	(5hr, 1.75V/cell, 25°C / 77°F)
	5.37 AH/1.79A	(3hr, 1.75V/cell, 25°C / 77°F)
	4.55 AH/4.55A	(1hr, 1.60V/cell, 25°C / 77°F)
Max Discharge Current	105A (5s)	
Internal Resistance	Approx 23mΩ	
Discharge Characteristics	Operating Temp. Range	Discharge: -15 ~ 50°C (5 ~ 122°F)
		Charge: 0 ~ 40°C (5 ~ 104°F)
		Storage: -15 ~ 40°C (5 ~ 104°F)
	Nominal Operating Temp. Range	25 ± 3°C (77 ± 5°F)
	Cycle Use	Initial Charging Current less than 2.1A.Voltage 14.4V ~ 15.0V at 25°C (77°F) Temp. Coefficient -30mV/°C
	Standby Use	No limit on Initial Charging Current Voltage 13.5V ~ 13.8V at 25°C (77°F) Temp. Coefficient -20mV/°C
	Capacity affected by Temperature	40°C (104°F) 103%
		25°C (77°F) 100%
		0°C (32°F) 86%
Design Floating Life at 20°C	5 Years	
Self Discharge	Ultracell batteries may be stored for up to 6 months at 25°C(°77F) and then a refresh charge is required. For higher temperatures the time interval will be shorter.	