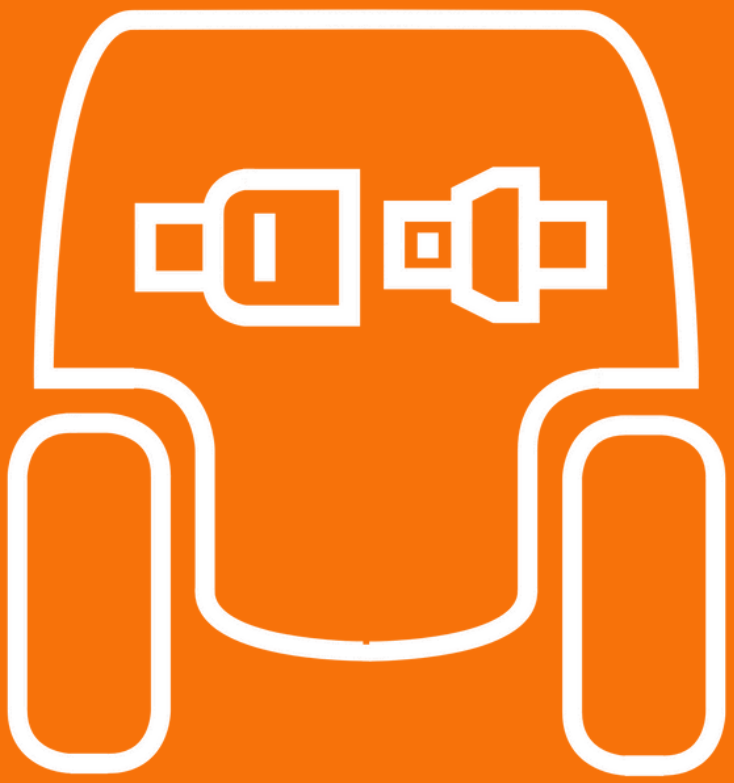
Munay 2024

BADEIGTS; Máximo - DUBAL, Agustín - GONZALEZ PAUTASO, Valentino - LUCENTINI, Juan Sebastian - MELCHOR, Francisco - TAMAI, Franco Nahuel



Informe Descriptivo

Escuela de Educación Técnica Nº7
Taller Regional Quilmes
Prácticas Profesionalizantes: Especialidad Aviónica

Intergrantes

BADEIGTS; Máximo
DNI: 47.066.639 - Tel.: 11 6442 5852

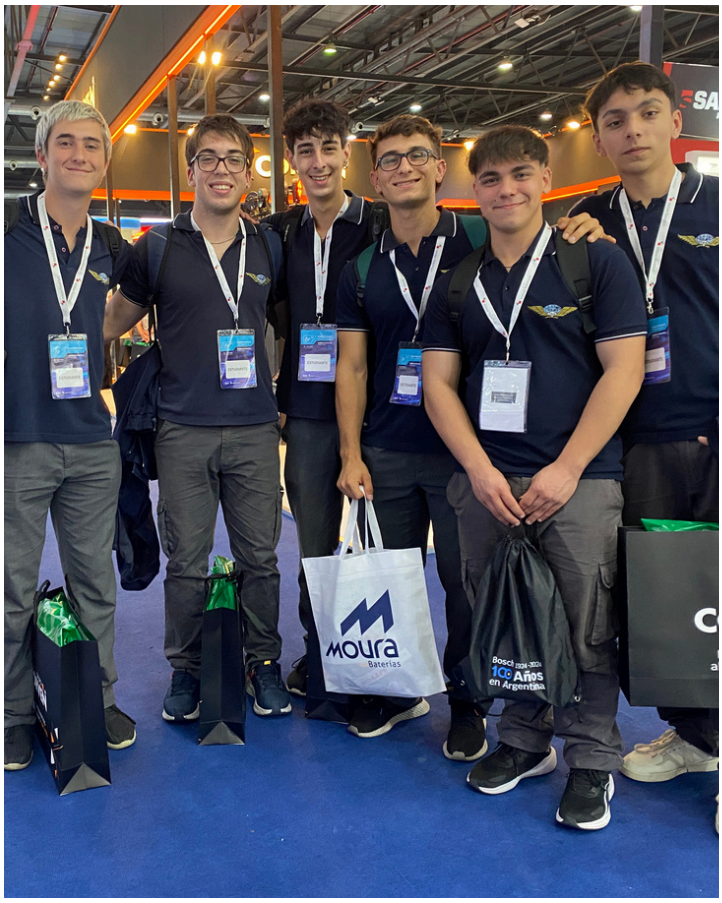
DUBAL, Agustín
DNI: 47.279.596 - Tel.: 11 2658 3058

GONZALEZ PAUTASO, Valentino
DNI: 46.680.272 - Tel.: 11 3873 4764

LUCENTINI, Juan Sebastian
DNI: 47.144.279 - Tel.: 11 2495 0701

MELCHOR, Francisco
DNI: 46.635.532 - Tel.: 11 6420 2998

TAMAI, Franco Nahuel
DNI: 46.955.036 - Tel.: 11 2707 4526



Docentes tutores:

BIANCO, Carlos

CARLASSARA, FABRIZIO

MEDINA, Sergio

PALMIERI, Diego

Fecha de inicio:

22 de Marzo de 2024

Duración

30 semanas

Esfuerzo de l proyecto

23 horas semanales
(690 de trabajo total)

Pagina web

https://munay-impa.web.app/

Instagram

https://www.instagram.com/proyecto_munay24/

Github

https://github.com/impatrq/munay

Índice

Introducción

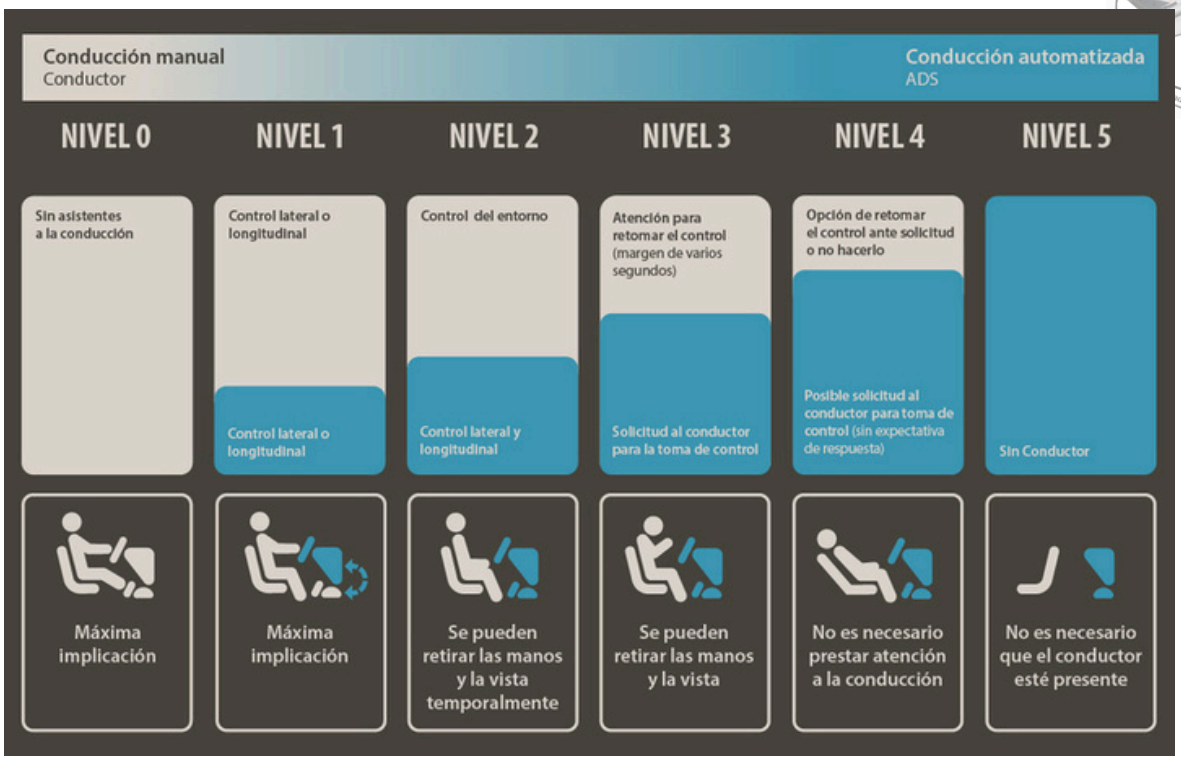
Objetivo

Desde su concepción, la intención del Proyecto “Munay”, presenta una preocupación por la gran huella de carbono dejada por los vehículos utilizados (usualmente camionetas Diésel) en la inspección de pista; dando así como respuesta la electrificación del vehículo. A su vez, mediante la aplicación de una IA especialmente entrenada para su tarea, busca reducir el personal y aumentar la eficacia en la detección de FOD´s.
Tras analizar las preocupaciones del anterior grupo, en las cuales coincidimos, notamos ciertas aristas en las soluciones encontradas, y que tenían aún más margen para seguir innovando y mejorando.

Por esta razón, el proyecto plantea efectivizar aún más el trabajo realizado por el operador del vehículo. El concepto inicial del proyecto Munay era ayudar en el trabajo de reconocimiento de FODs en pista, dejando una menor huella de carbono y reduciendo los gastos en cantidad de trabajadores innecesarios, ya que este reconocimiento se realizaba en camionetas y con varios trabajadores. En esta búsqueda de reducir gastos e impacto ambiental, encontramos una contrapartida. Por cómo se diseñó el programa de notificaciones de FODs durante sus búsquedas, el operador del Munay debe distraer su atención del camino para confirmar en el sistema si lo que avistó es precisamente un FOD o no.
Dado a que se insiste en reducir el gasto de personal, y este único operador tiene que realizar dos tareas en simultáneo, manejar y confirmar avistamientos, planteamos quitarle (parcialmente) la responsabilidad de una de esas dos tareas; La Conducción.

Para aplicar lo planteado, decidimos implementar ADAS (Advanced Driver Assistance Systems) para lograr un vehículo con un nivel de autonomía N°2

NIVELES DE AUTONOMÍA:



Descripcion General

El proyecto consta de tres sistemas principales; alerta de riesgo de colisión frontal, alerta de cambio de carril involuntario y control crucero.

- Alerta de riesgo de colisión frontal: El sistema considera la velocidad del vehículo, la distancia entre su frente y un supuesto obstáculo y la velocidad de dicho obstáculo mediante un sensor ultrasónico, y avisa al conductor si se aproxima de manera peligrosa a este.

- Alerta de cambio de carril involuntario: mediante sensores infrarrojos situados en su lateral se mide y compara entre sí la distancia del vehículo respecto de las líneas de carril trazadas en el asfalto. Se comparan ambas distancias para saber si el vehículo se encuentra centrado en el carril (dando un margen de error de unos centímetros). A medida que la distancia respecto a una de las líneas aumenta y la otra disminuye, se detecta el desvío de carril y se prende un testigo luminoso en el tablero, el cual lo indica de manera clara.

- Control crucero: Regula a los motores una velocidad constante manualmente establecida por el conductor. Los datos de velocidad son calculados entre el diámetro

de la rueda y los pulsos del motor por cada vuelta por el código. Puesto que utiliza motores de corriente, a cada cantidad de caudal de corriente le corresponde una cantidad de vueltas (que se refleja en velocidad del vehículo).

La tercera etapa (ADAS 2) consiste en desarrollar los anteriores sistemas lo suficiente para volverlos activos. El sistema "Alerta de Riesgo de Colisión Frontal" sería llevado a "Frenado Autónomo de Emergencia". El sistema de Alerta de cambio de carril involuntario se cambiaría por el sistema de Mantenimiento de Carril Activo. Por su lado, el Control Crucero no será desarrollado a su opción "activa" (Control Crucero Adaptativo) dado que este es una medida de seguridad contra tráfico de un mismo carril, situación a la cual el "Munay" no se enfrentaría.

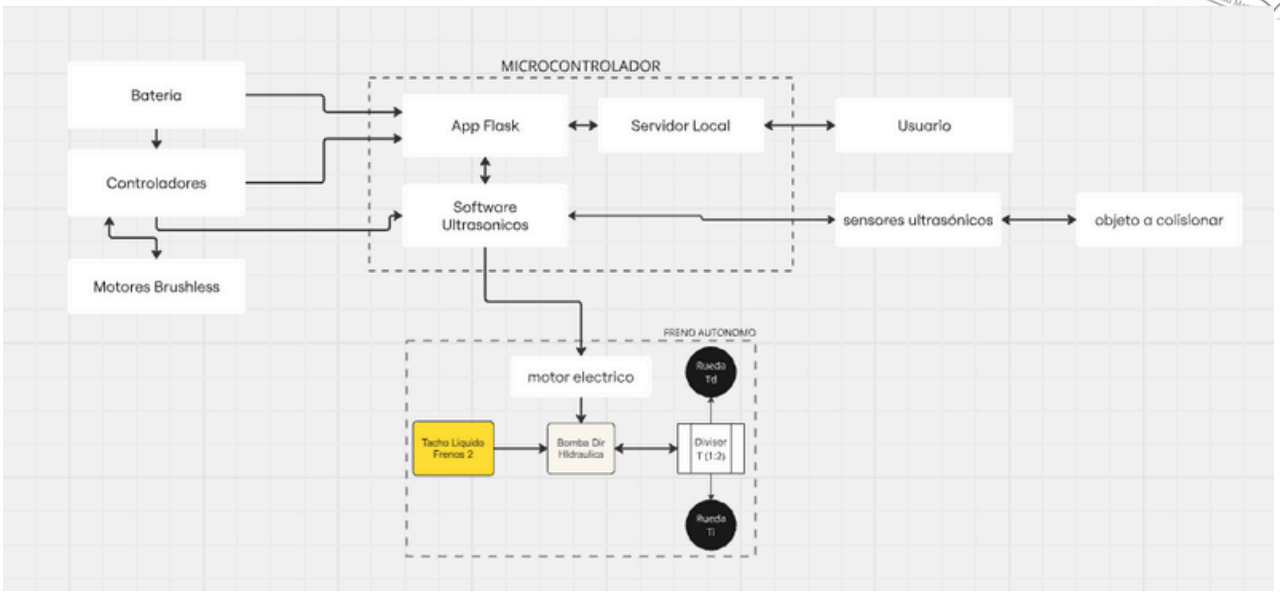
- Frenado Autónomo de Emergencia: la fase donde se reconoce la necesidad o no de frenar ya estaría cubierta, se utilizará la misma metodología que en su versión pasiva. Para el accionamiento del mismo freno, una vez ya reconocida la necesidad, el microcontrolador acciona un motor eléctrico que, al estar unido mediante una correa a una bomba de dirección hidraulica, le da presión al sistema de freno alternativo

Alcance

Munay está claramente destinado al trabajo aeroportuario, específicamente para el personal de seguridad del mismo.
A pesar de su claro enfoque al rubro Aeronáutico, el segmento al que apunta nuestro proyecto puede ser también de un consumo industrial mas general, de parte de empresas que busquen la automatización de sus trabajos.

Nos permitimos pensar en gran escala y creemos que tiene techos en ventas de uso industrial no visibles.
El vehículo se puede dividir en dos tecnologías; reconocimiento de FOD´s mediante IA y conducción semiautónoma. Esto permite pensar en el producto base como un vehículo semiautónomo, con una IA configurable al reconocimiento de cualquier tipo de objeto, para cualquier tipo de industria. Como ejemplo burdo y rápido; control de stock en un galpón de almacenamiento de productos.
Esta característica dota al vehículo una versatilidad inimaginable como producto, adaptándose a las necesidades industriales específicas de cada consumidor

Diagrama en bloques global



Electrónica, Software y Sistema Embebido

De base se utilizó una Raspberry Pi4 model B para el procesamiento de los códigos desarrollados respectivamente.

Los lenguajes utilizados fueron; CSS 45.6% , HTML 14.1% , JavaScript 9.3% , Python 5.1% y C 3.5%.

- CSS: utilizado para el estilo de la pagina

- HTML: utilizado para la estructura de la pagina web y la interfaz grafica del dashboard

- JavaScript: utilizado para el desarrollo de la parte dinámica de la pagina web y del dashboard (animaciones en la pagina web, actualizacion de datos de hora y parámetros de motor en el dashboard)

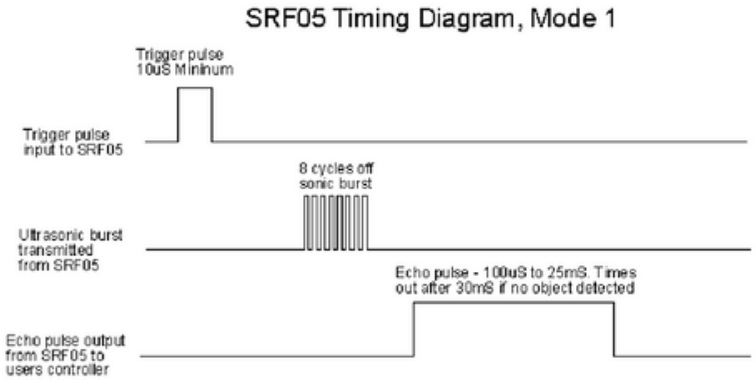
- Python: utilizado para la aplicación de flask, que sirve para la comunicación del servidor y el microcontrolador. Y también utilizado en el entrenamiento de una red

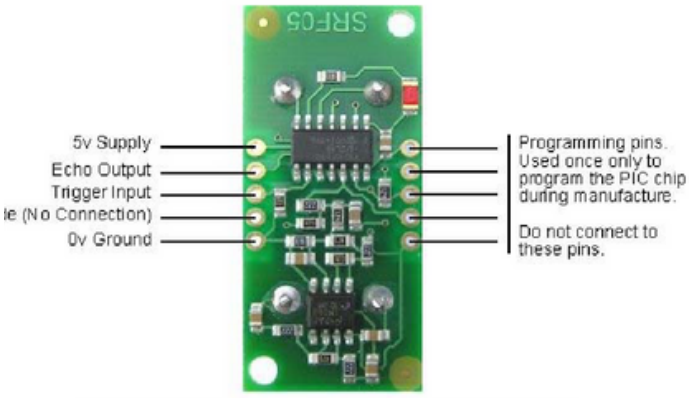
neuronal convolucional (archivo para el desarrollo del sistema de detección de cambio de carril involuntario)

- C: utilizado para el funcionamiento de los sensores ultrasónico (hy-sr05) y el accionamiento del motor eléctrico de freno.

Por su lado, los sensores ultrasónicos utilizados en la detección de objetos de posible colisión fueron los Hy-srf05. Modos de uso:

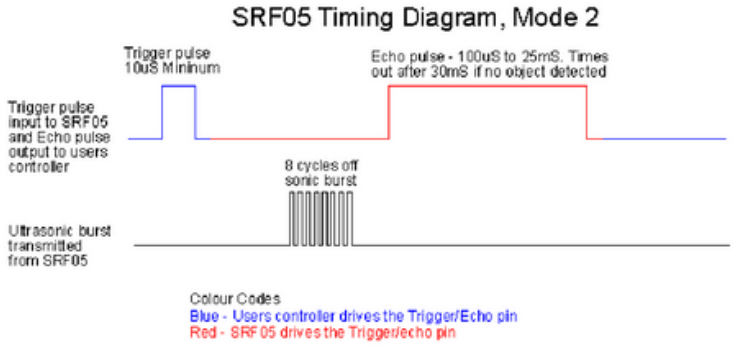
Mode 1 - Separate Trigger and Echo

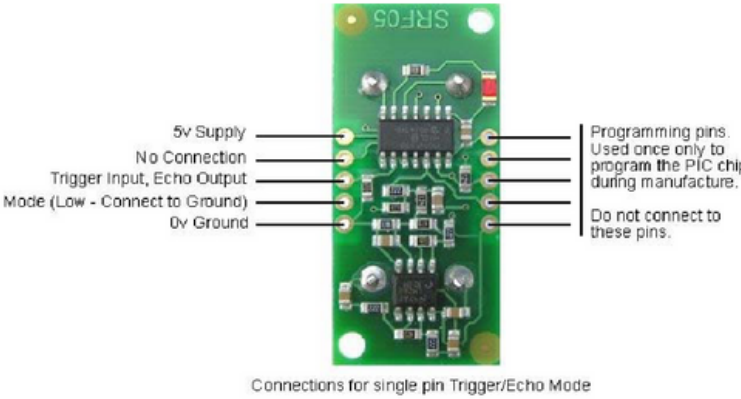




Connections for 2-pin Trigger/Echo Mode (SRF04 compatible)

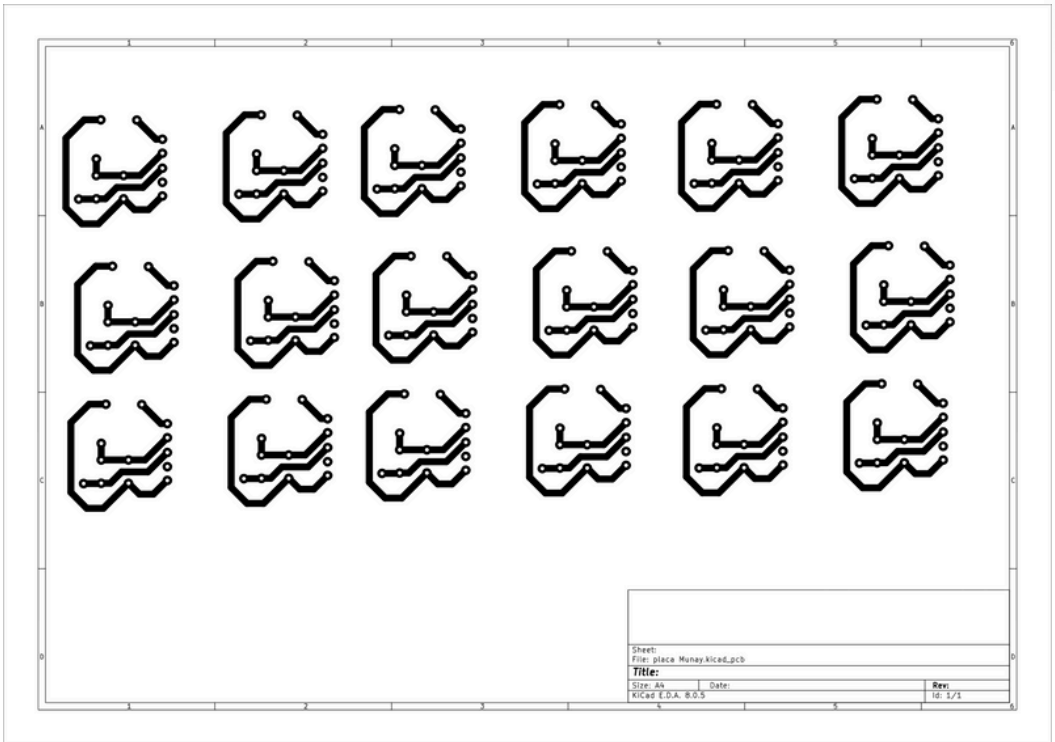
Mode 2 - Single pin for both Trigger and Echo





Connections for single pin Trigger/Echo Mode

Diseño de PCB de los sensores ultrasonicos:



Código de calibración de Sensores Ultrasónicos

```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <pigpio.h>
#include "libs/hysrf05.h"



Sensor sensors[NUM_SENSORS] = {
    {14, 15},//Completa con los pines a los que se conecte el sensor 1
    {18, 23},//Completa con los pines a los que se conecte el sensor 2
    {24, 25},//Completa con los pines a los que se conecte el sensor 3
    {8, 7},//Completar con los pines a los que se conecte el sensor 4
    {12, 16} //Completa con los pines a los que se conecte el sensor 5
};


void *measure_distances(void *arg) {
    while (1) {
        double min_distance = 9999;
        int closest_sensor = -1;


        for (int i = 0; i < NUM_SENSORS; i++) {
            double distance = sensor_get_distance(&sensors[i]);
            printf("Sensor %d mide: %.2f cm\n", i + 1, distance);

```


```
            if (distance < min_distance) {
                min_distance = distance;
                closest_sensor = i;
            }
        }
        sleep(1);
    }
    return NULL;
}

int main() {

    if (gpioInitialise() < 0) {
        printf("¡Error al iniciar pigpio!\n");
        return 1;
    }

    for (int i = 0; i < NUM_SENSORS; i++) {
        sensor_init(&sensors[i]);
    }

    pthread_t distance_thread;
    if (pthread_create(&distance_thread, NULL, measure_distances, NULL) != 0) {
        printf("Error al crear el hilo de medición\n");
        return 1;
```

```
        return 1;
    }

    pthread_join(distance_thread, NULL);
    /*agregar variable en la que se guarde el valor mas bajo medido, despues agregar un actuador que
    varía la cantidad de freno en funcion de qué tan cerca esté el objeto */
    gpioTerminate();

    return 0;
}
```

Pantalla y Dashboard

Para visualizar el Dashboard, se usará una tablet que estará colocada en el tablero del automóvil. Esta será alimentada por la batería de 12 V ubicada en la parte frontal del vehículo. Para cargar la tablet desde esta batería, se utilizará un convertidor LM2596, cuyo propósito es transformar los 12 V en 5 V, el voltaje adecuado para la tablet. Diagrama correspondiente a su alimentación:

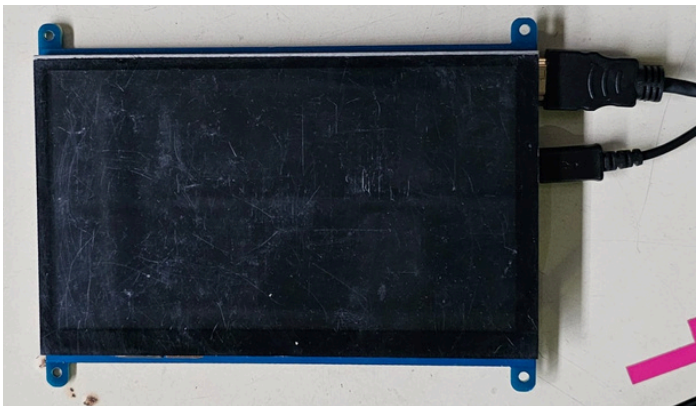
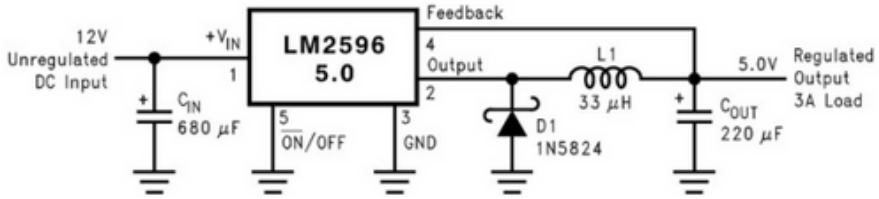


Diagrama correspondiente a su alimentación:



Servidor Flask para correr el dashboard:

Este proyecto emplea HTML para definir la estructura de la página web, CSS para su diseño, JavaScript para la lógica en el cliente (actualizaciones en tiempo real y manipulación del DOM), y Python con Flask en el backend para gestionar las solicitudes del cliente y ejecutar la lógica del servidor. Su objetivo es desarrollar una interfaz de usuario y un servidor web que permita monitorear el estado de una Raspberry Pi4. La organización del proyecto incluye archivos HTML, CSS y JavaScript para el frontend, y un servidor Flask en Python para manejar la lógica del servidor y las solicitudes del cliente.

Partes del Dashboard

- Interfaz: Es un panel visual que incorpora diversos indicadores. Participan los archivos ***index.html*** y ***style.css***. Este panel muestra elementos como un velocímetro y una sección para monitorear el estado de la batería

- Medidor de Velocidad: Muestra la velocidad de forma visual, empleando elementos y estilos CSS para generar un indicador circular con una aguja que representa la velocidad de manera gráfica.

- Batería: Indica el nivel de carga y el estado de la batería. Los archivos ***styles.css*** y ***main.js*** participan en esta función. Utiliza JavaScript para obtener datos del servidor Flask, como el porcentaje de batería, actualizando dinámicamente la interfaz de usuario. Incluye una visualización del nivel de carga y un mensaje sobre el estado de la batería.

Interfaz del dashboard:

Este es el panel de control que se ha diseñado y programado utilizando HTML, CSS y JavaScript. Este panel funcionará como el tablero típico de un automóvil, mostrando datos como la velocidad, el porcentaje de carga de las baterías y la hora. Las diferencias con el anterior diseño de Dashboard son notorias; comenzando por la implementación de botones, con los cuales el usuario podrá cambiar entre modos de frenado, los cuales están calibrados para frenar de diferentes formas dependiendo el clima (superficie seca, mojada o nevada). También, se buscó rediseñar la disposición de la información, haciéndola más legible y fresca al repasar con la vista rápidamente durante la conducción.

PENDIENTE AGUSTIN

Desarrollo Dashboard

```python
from flask import Flask, render_template, jsonify
import RPi.GPIO as GPIO
import time
speed=0

app = Flask(__name__)

try:
    # Intentar importar módulos específicos que dependen del ADC
    import board
    import busio
    import adafruit_ads1x15.ads1115 as ADS
    from adafruit_ads1x15.analog_in import AnalogIn

    # Intentar crear el objeto I2C bus y ADC
    i2c = busio.I2C(board.SCL, board.SDA)
    ads = ADS.ADS1115(i2c)

    # Crear un canal de entrada analógica en el ADC
    chan = AnalogIn(ads, ADS.P0)

    # Definir la función int0 después de haber importado el módulo
    def int0(channel):
        global up_times, index, m2
        m1 = time.time()
        up_times[index] = (m1 - m2) * 1E6
        index = (index + 1) % 3
        m2 = m1
except ImportError:
    print("Error: No se pudo importar el módulo del ADC. El ADC no estará disponible.")
    chan = None  # Configurar chan como None si el ADC no está disponible
    encoder_reading_pin = None  # Configurar encoder_reading_pin como None si el ADC no está disponible
```

```
except Exception as e:
    print(f"Error initializing ADC: {e}")
    chan = None  # Configurar chan como None si la inicialización falla
    encoder_reading_pin = None  # Configurar encoder_reading_pin como None si la inicialización falla

# Configuración de GPIO si encoder_reading_pin está definido
if encoder_reading_pin is not None:
    GPIO.setmode(GPIO.BCM)

    # Variables para el medidor de velocidad
    encoder_reading_pin = 17
    up_times = [0, 0, 0]
    index = 0
    m2 = time.time()
    speed = 0

    GPIO.setup(encoder_reading_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    GPIO.add_event_detect(encoder_reading_pin, GPIO.BOTH, callback=int0)


# Función para calcular el porcentaje de carga de la batería
def calculate_percentage_charge(adc_voltage):
    if adc_voltage is None:
        return 0  # Default to 0% if ADC is not initialized

    # Define las variables de la batería
    voltage_divider_ratio = 15
    battery_voltage_min = 0.0 * 12  # 2.4V per cell for a 12-cell battery
    battery_voltage_max = 4.0 * 12  # 4.2V per cell for a 12-cell battery

    # Calcula el voltaje de la batería y el porcentaje de carga
    battery_voltage = adc_voltage * voltage_divider_ratio

    if battery_voltage < battery_voltage_min or battery_voltage > battery_voltage_max:
        return 0  # Default to 0% if voltage is out of range
```

```
    percentage_charge = ((battery_voltage - battery_voltage_min) / (battery_voltage_max - battery_voltage_min)) * 100

    if percentage_charge < 0:
        percentage_charge = 0
    elif percentage_charge > 100:
        percentage_charge = 100

    return percentage_charge

# Variable global para el perfil de frenado
braking_profile = "dry"  # Valor predeterminado

# Ruta para cambiar el perfil de frenado
@app.route('/set_braking_profile')
def set_braking_profile():
    global braking_profile
    profile = request.args.get('profile')

    if profile in ["dry", "rain", "snow"]:
        braking_profile = profile
        return f"Perfil de frenado cambiado a: {braking_profile}"
    else:
        return "Perfil no válido", 400


# Endpoint para obtener el porcentaje de batería
@app.route("/get_battery_percentage", methods=["GET"])
def get_battery_percentage():
    # Leer el voltaje del ADC
    if chan is None:
        adc_voltage = 0  # Default to 0 if ADC is not initialized
    else:
        adc_voltage = chan.voltage

    # Calcular el porcentaje de carga
```

```
    # Calcular el porcentaje de carga
    percentage_charge = calculate_percentage_charge(adc_voltage)

    # Devolver el porcentaje en formato JSON
    return jsonify({"percentage": percentage_charge})

# Endpoint para obtener la velocidad actual
@app.route("/get_motor_speed", methods=["GET"])
def get_motor_speed():
    global speed
    return jsonify({"speedd": speed})

# Renderizar la plantilla con los valores del porcentaje y la velocidad
@app.route("/")
def index():
    return render_template("index.html", percentage=0, speedd=0)

if __name__ == "__main__":
    app.run()
```

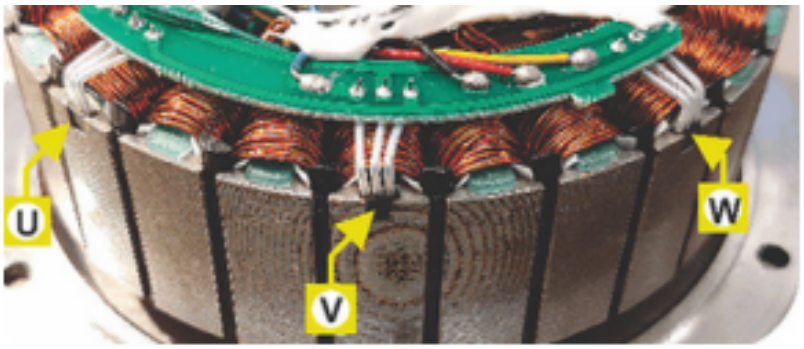
Drivers (controladores)

Los controladores utilizados con los Motores Brushless fueron los Kelly KLS6022H. Estos controladores están diseñados para motores brushless que incorporan tres sensores Hall. Su fuente de alimentación se compone de dos baterías aeronáuticas de 24 voltios conectadas en serie, proporcionando un suministro de 48 voltios. Incorpora un acelerador que opera a 5 voltios, lo cual permite regular la velocidad del motor. También dispone de una palanca de mando bidireccional para controlar la dirección de los motores, permitiendo avanzar y retroceder. Además, cuenta con una llave de corte general que habilita o deshabilita la alimentación eléctrica de todo el sistema, brindando un control esencial sobre la energía suministrada al motor y al conjunto.

Motores BLDC

- Par de polos: 16
- Potencia nominal: 2000w
- Potencia máxima: 4000w
- Tensión nominal: 48v
- Máx. torque: 190 N.M
- Angulo de fase del sensor hall: 120°
- Máx. corriente: 55A T
- ipo de freno: freno de disco
- N.W./ G..W. : 15 kg / 16 kg
- Tamaño: 34*34*33 CM

La velocidad se medirá mediante la lectura de los tres sensores Hall ubicados alrededor del estator del motor. Estos sensores detectan cambios en el campo magnético generados por los imanes permanentes del rotor. Los sensores Hall emiten una señal de 12 V, la cual se reducirá a 3.3 V mediante un divisor de tensión, permitiendo su conexión al pin GPIO de la Raspberry Pi4. A través de este pin, se contarán los pulsos del motor para, junto con el diámetro de la rueda, calcular la velocidad.



Sensores hall

Batería de alimentación de los controladores:

- Voltaje Nominal: 24V
- Capacidad: 27Ah
- Autonomía: 1h a una velocidad de 30km/h

Batería de alimentación de Perisféricos:

se ubica en la parte delantera del vehículo. Destinada a alimentar el conjunto de luces y faros normativos, ademas de las múltiples cámaras y tablet.

- Voltaje nominal: 12v
- Largo: 270mm
- Ancho: 180mm
- Alto: 180mm

Adaptación Mecanica

Para lograr la automatización del freno se necesitó desglosar el sistema de freno de accionamiento manual que el vehículo poseía en dos. La primera parte se mantuvo igual, comandando el freno de las ruedas delanteras manualmente, mientras que el freno de las ruedas traseras se adaptó para ser accionado mediante una bomba dirección hidráulica. A esta bomba la alimentaba mediante una correa dentada un motor eléctrico alza cristales.
Ya que todas las piezas utilizadas provenían de sistemas, marcas y hasta cumplían funciones diferentes al uso que se le quería dar, se necesitó adaptarlas físicamente (cortando, soldando y roscando) a varias de ellas.

Pieza fabricada para la conexión entre el ducto hidráulico de los frenos traseros y la rosca del orificio de salida de la bomba hidráulica.:

Solución desarrollada en la busqueda de transferencia de movimiento circular; inclusión de poleas para correas dentadas en ambos ejes (tanto del motor alza cristales como de la bomba hidráulica):

La estructura principal (chasis) sufrió solo pequeños cambios, como un soporte adaptador para la instalación de un cinturón y en el frontal, sobre dónde se encuentra la dirección del vehículo, se implementó un soporte donde se empotraron tanto la bomba de dirección hidráulica como el motor eléctrico alza cristales.


AVIÓNICA
Identidad Maestría Progreso Acción