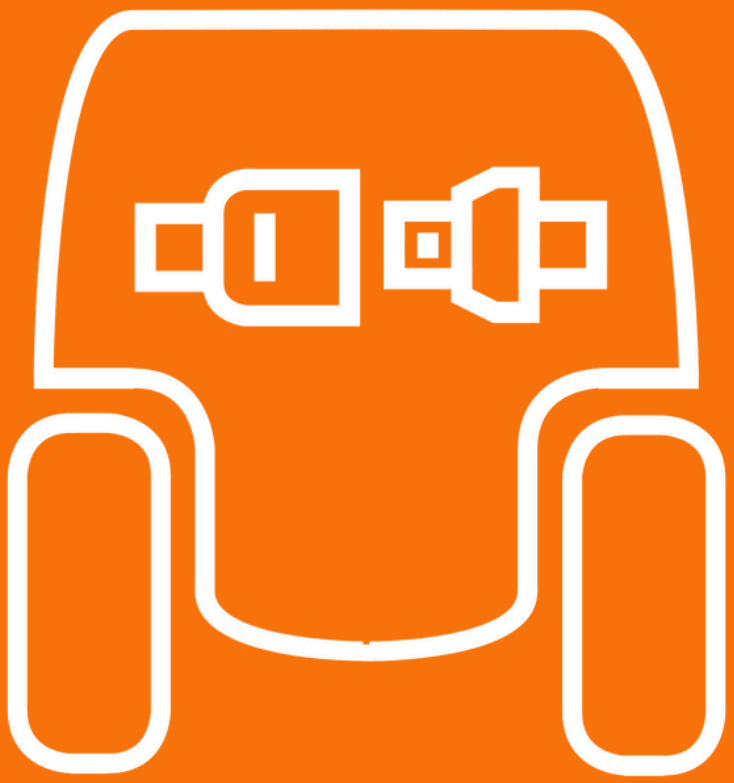
Munay 2024

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

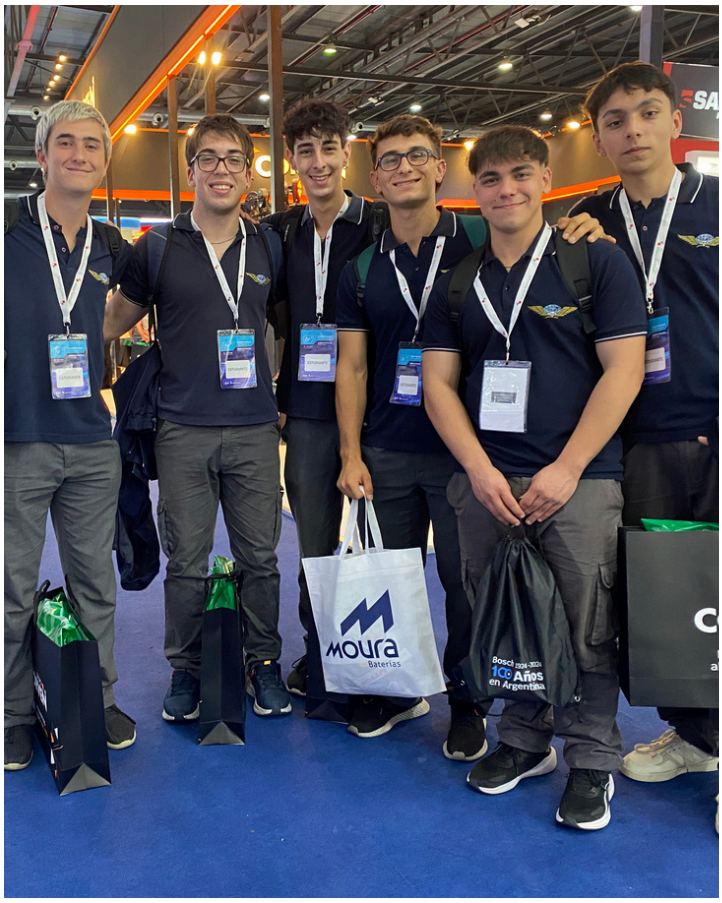
DUBAL, Agustín
DNI: 47.279.596 - Tel.: 11 2658 3058

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

Mediante este proyecto se planteó una serie de escalones (tres) de mejora sobre el auto eléctrico experimental “Munay”.
El proyecto presenta efectivizar aún más el trabajo realizado por el operador de este mismo. 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), decidimos simplificar a este una de esas dos tareas mediante ADAS (Advanced Driver Assistance Systems). De esta manera garantizamos una mayor efectividad en el empleo de ambas tareas.

Alcance

Munay está 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 ya que el vehículo se puede dividir en dos tecnologías; conducción semiautónoma y el reconocimiento de FOD´s mediante IA. Esto le permite al vehículo una gran versatilidad como producto, si se personaliza el segundo sistema mencionado dependiendo de la industria que lo requiera.

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.

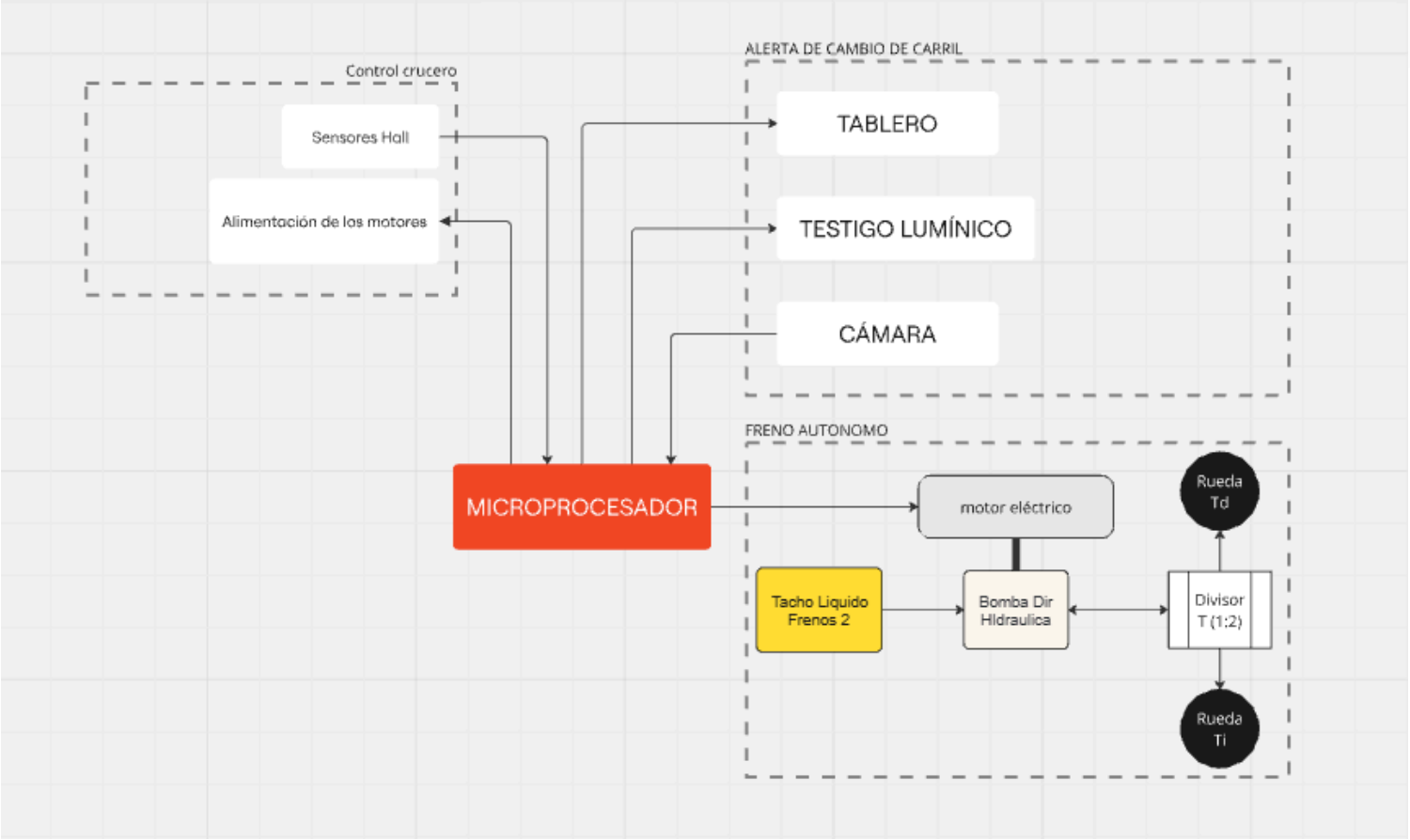
- Alerta de cambio de carril involuntario: mediante sensores infrarrojos situados en su lateral se mide y compara entre sí la distancia del vehículo respecto de las líneas de carril trazadas en el asfalto. Se comparan ambas distancias para saber si el vehículo se encuentra centrado en el carril (dando 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

Diagrama en bloques global



Software

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.

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);
```

```c
            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;
```

```c
        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;
}
```

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
```

```python
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
```

```python
    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
```


```python
        # 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()
```