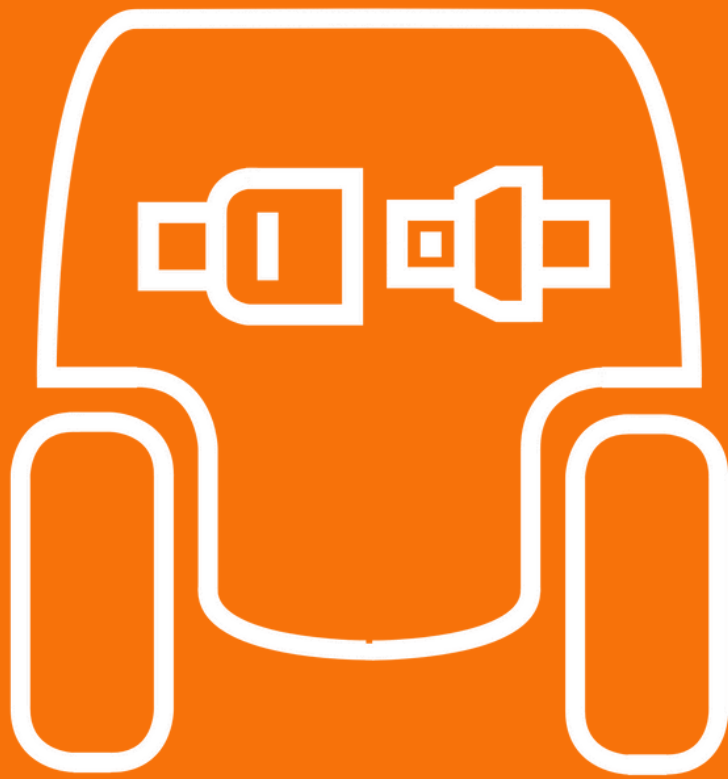


Munay 2024

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.



Descripción General

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

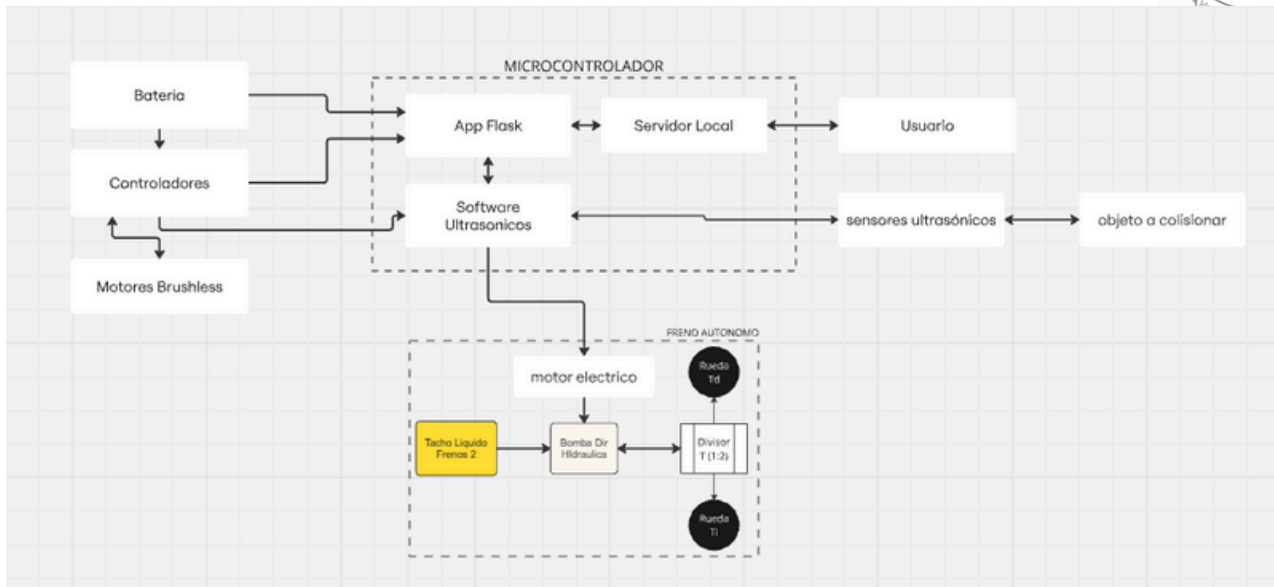
- **Alerta de riesgo de colisión frontal:** El sistema considera la velocidad del vehículo, la distancia entre su frente y un supuesto obstáculo y la velocidad de dicho obstáculo mediante un sensor ultrasónico, y avisa al conductor si se aproxima de manera peligrosa a este.
- **Alerta de cambio de carril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



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%.

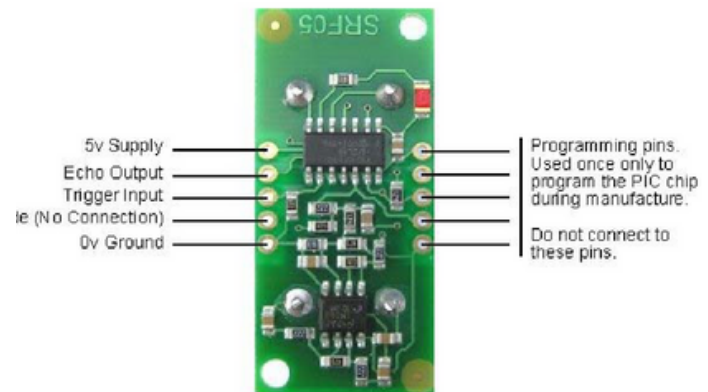
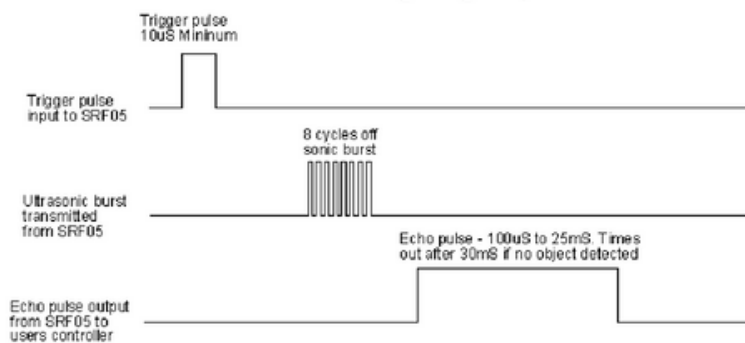
- CSS: utilizado para el estilo de la pagina
- HTML: utilizado para la estructura de la pagina web y la interfaz grafica del dashboard
- JavaScript: utilizado para el desarrollo de la parte dinámica de la pagina web y del dashboard (animaciones en la pagina web, actualizacion de datos de hora y parámetros 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

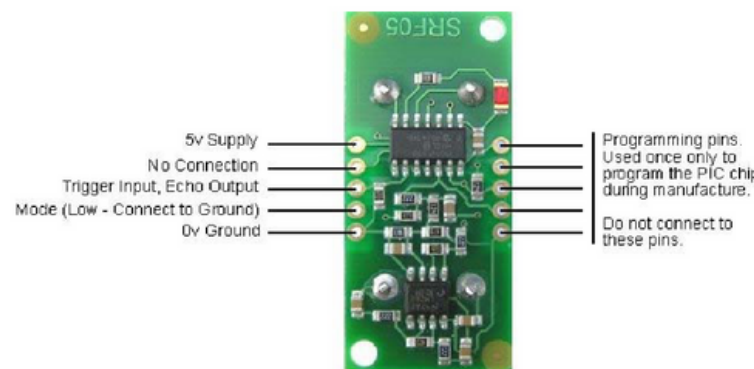
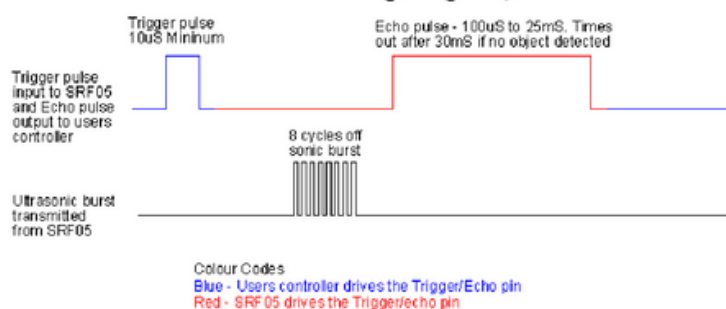
SRF05 Timing Diagram, Mode 1



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

Mode 2 - Single pin for both Trigger and Echo

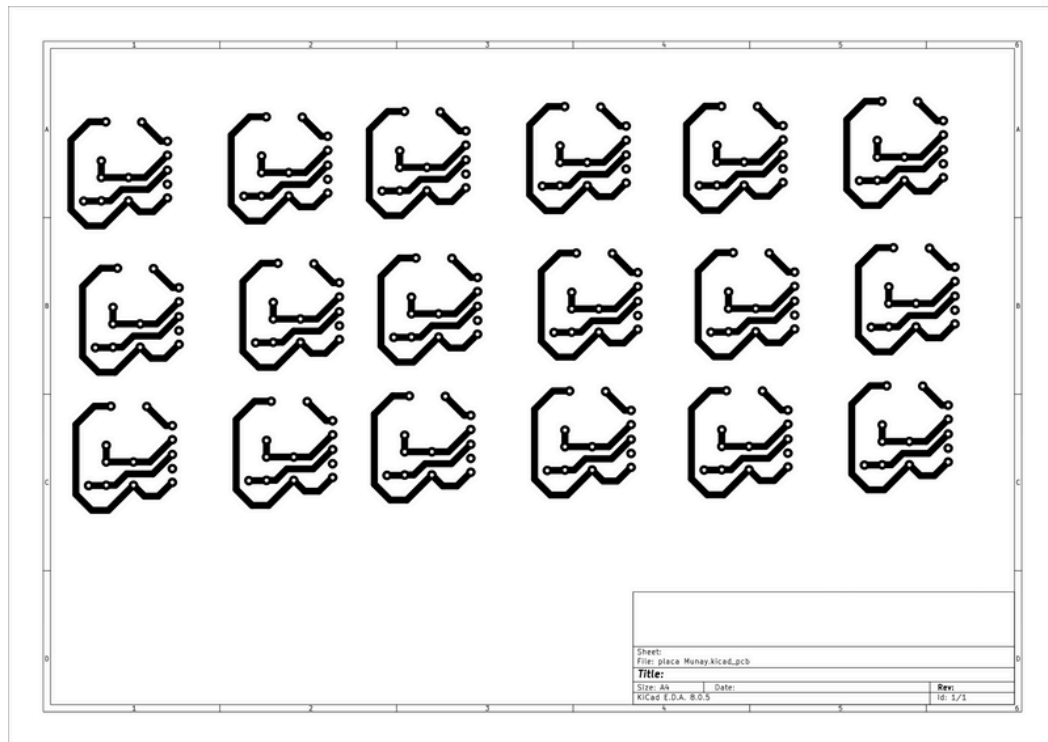
SRF05 Timing Diagram, Mode 2



Connections for single pin Trigger/Echo Mode



Diseño de PCB de los sensores ultrasonicos:



Código de calibración de Sensores Ultrasónicos

```

1  #include <stdio.h>
2  #include <pthread.h>
3  #include <unistd.h>
4  #include <pigpio.h>
5  #include "libs/hysrf05.h"
6
7
8
9  Sensor sensors[NUM_SENSORS] = {
10     {14, 15}, //Completa con los pines a los que se conecte el sensor 1
11     {18, 23}, //Completa con los pines a los que se conecte el sensor 2
12     {24, 25}, //Completa con los pines a los que se conecte el sensor 3
13     {8, 7}, //Completa con los pines a los que se conecte el sensor 4
14     {12, 16} //Completa con los pines a los que se conecte el sensor 5
15 };
16
17
18 void *measure_distances(void *arg) {
19     while (1) {
20         double min_distance = 9999;
21         int closest_sensor = -1;
22
23
24         for (int i = 0; i < NUM_SENSORS; i++) {
25             double distance = sensor_get_distance(&sensors[i]);
26             printf("Sensor %d mide: %.2f cm\n", i + 1, distance);
27

```



```

28         if (distance < min_distance) {
29             min_distance = distance;
30             closest_sensor = i;
31         }
32     }
33     sleep(1);
34 }
35 return NULL;
36 }
37
38 int main() {
39
40     if (gpioInitialise() < 0) {
41         printf("Error al iniciar pigpio!\n");
42         return 1;
43     }
44
45     for (int i = 0; i < NUM_SENSORS; i++) {
46         sensor_init(&sensors[i]);
47     }
48
49     pthread_t distance_thread;
50     if (pthread_create(&distance_thread, NULL, measure_distances, NULL) != 0) {
51         printf("Error al crear el hilo de medición\n");
52         return 1;
53     }
54 }

```

Desarrollo Dashboard

```

1  from flask import Flask, render_template, jsonify
2  import RPi.GPIO as GPIO
3  import time
4  speed=0
5
6  app = Flask(__name__)
7
8  try:
9      # Intentar importar módulos específicos que dependen del ADC
10     import board
11     import busio
12     import adafruit_ads1x15.ads1115 as ADS
13     from adafruit_ads1x15.analog_in import AnalogIn
14
15     # Intentar crear el objeto I2C bus y ADC
16     i2c = busio.I2C(board.SCL, board.SDA)
17     ads = ADS.ADS1115(i2c)
18
19     # Crear un canal de entrada analógica en el ADC
20     chan = AnalogIn(ads, ADS.P0)
21
22     # Definir la función int8 después de haber importado el módulo
23     def int8(channel):
24         global up_times, index, s2
25         s1 = time.time()
26         up_times[index] = (s1 - s2) * 100
27         index = (index + 1) % 3
28         s2 = s1
29     except ImportError:
30         print("Error: No se pudo importar el módulo del ADC. El ADC no estará disponible.")
31         chan = None # Configurar chan como None si el ADC no está disponible
32         encoder_reading_pin = None # Configurar encoder_reading_pin como None si el ADC no está disponible

```

```

33     except Exception as e:
34         print(f"Error Initializing ADC: {e}")
35         chan = None # Configurar chan como None si la inicialización falla
36         encoder_reading_pin = None # Configurar encoder_reading_pin como None si la inicialización falla
37
38     # Configuración de GPIO si encoder_reading_pin está definido
39     if encoder_reading_pin is not None:
40         GPIO.setmode(GPIO.BCM)
41
42     # Variables para el medidor de velocidad
43     encoder_reading_pin = 17
44     up_times = [0, 0, 0]
45     index = 0
46     s1 = time.time()
47     speed = 0
48
49     GPIO.setup(encoder_reading_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
50     GPIO.add_event_detect(encoder_reading_pin, GPIO.BOTH, callback=int8)
51
52
53     # Función para calcular el porcentaje de carga de la batería
54     def calculate_percentage_charge(adc_voltage):
55         if adc_voltage is None:
56             return 0 # Default to 0% if ADC is not initialized
57
58         # Define las variables de la batería
59         voltage_divider_ratio = 15
60         battery_voltage_min = 0.0 * 12 # 2.4V per cell for a 12-cell battery
61         battery_voltage_max = 4.0 * 12 # 4.2V per cell for a 12-cell battery
62
63         # Calcula el voltaje de la batería y el porcentaje de carga
64         battery_voltage = adc_voltage * voltage_divider_ratio
65
66         if battery_voltage < battery_voltage_min or battery_voltage > battery_voltage_max:
67             return 0 # Default to 0% if voltage is out of range

```



```
68 percentage_charge = ((battery_voltage - battery_voltage_min) / (battery_voltage_max - battery_voltage_min)) * 100
69
70
71 if percentage_charge < 0:
72     percentage_charge = 0
73 elif percentage_charge > 100:
74     percentage_charge = 100
75
76 return percentage_charge
77
78 # Variable global para el perfil de frenado
79 braking_profile = "dry" # Valor predeterminado
80
81 # Ruta para cambiar el perfil de frenado
82 @app.route('/set_braking_profile')
83 def set_braking_profile():
84     global braking_profile
85     profile = request.args.get('profile')
86
87     if profile in ["dry", "rain", "snow"]:
88         braking_profile = profile
89         return f'Perfil de frenado cambiado a: {braking_profile}'
90     else:
91         return "Perfil no válido", 400
92
93
94 # Endpoint para obtener el porcentaje de batería
95 @app.route('/get_battery_percentage', methods=["GET"])
96 def get_battery_percentage():
97     # Leer el voltaje del ADC
98     if chan is None:
99         adc_voltage = 0 # Default to 0 if ADC is not initialized
100     else:
101         adc_voltage = chan.voltage
102
103     # Calcular el porcentaje de carga
```

```
103     # Calcular el porcentaje de carga
104     percentage_charge = calculate_percentage_charge(adc_voltage)
105
106     # Devolver el porcentaje en formato JSON
107     return jsonify({"percentage": percentage_charge})
108
109 # Endpoint para obtener la velocidad actual
110 @app.route('/get_motor_speed', methods=["GET"])
111 def get_motor_speed():
112     global speed
113     return jsonify({"speed": speed})
114
115 # Renderizar la plantilla con los valores del porcentaje y la velocidad
116 @app.route("/")
117 def index():
118     return render_template("index.html", percentage=0, speed=0)
119
120 if __name__ == "__main__":
121     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



Batería de alimentación de los controladores:

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



Batería de alimentación de Periféricos:

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

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





Adaptación Mecánica

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 búsqueda 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 pequeños cambios, como un soporte adaptador para la instalación de un cinturón. Por otro lado, 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 alza cristales.

