

Diseño de sistemas empotrados

y de tiempo real

Control de velocidad de un vehículo (SCVV)

Adrian Rubio Hernandez

Belén Ariño Bolinches

[**INTRODUCCIÓN 3**](#_831cugomnnd9)

[**REQUISITOS 3**](#_vs28387dgu14)

[Objetivos del Sistema 3](#_wzx55zi0tf1n)

[Componentes involucrados 3](#_isds0n3745vf)

[Entradas del sistema 4](#_dci1q25eax1e)

[Salidas del sistema 4](#_tb4t9msqjwas)

[Requisitos Funcionales 5](#_b0iaomizjbvl)

[Requisitos no Funcionales 5](#_j9gfl7o7blga)

[**ESPECIFICACIONES 6**](#_42ak0js0vyie)

[Dimensiones y Peso 6](#_wzr5llvzpa4y)

[Consumo Energético 6](#_lleumgd3b353)

[Controles externos 6](#_g3t8qnsxj6nn)

[Controles internos 7](#_4zv1qj2gav1x)

[**ARQUITECTURA 8**](#_gbaaaknr4yq8)

[Diseño general 8](#_73f3tmyh670d)

[Diseño Hardware 8](#_ggzeed3gwjre)

[Diseño Software 9](#_s6996kolnfh9)

[**IMPLEMENTACIÓN 9**](#_x8x1yqdc900m)

[Código 10](#_dicrw670m5lz)

[Parte 1 10](#_rbs2p748mpy3)

[Parte 2 setup() 11](#_hkl72q62dig0)

[Parte 3 controlTask(void \*pvParameters) 12](#_yeggifnr6tdb)

[Parte 4 servoControlTask(int potValue) 12](#_lqweimwptm5e)

[Parte 5 setMotorSpeed(int speed) 13](#_pxbpmzjt985z)

[Parte 6 setMotorSpeedSimple(int speed) 14](#_6dy7nizex5sv)

[Parte 7 displayControlTask(void \*pvParameters) 15](#_ez5blc48axfl)

[Parte 8 contarRevolucion() 16](#_o15stgry5ouj)

# 

# INTRODUCCIÓN

Este documento establece los requisitos para el desarrollo de un sistema empotrado destinado al control de velocidad de un vehículo simulado, utilizando un motor instalado en un disco.

El sistema estará compuesto por un sensor de velocidad basado en un láser, un motor eléctrico, un controlador y funcionalidades adicionales para activación, desactivación, ajuste de velocidad y almacenamiento de velocidades preprogramadas. Además, se incluirá una función de "velocidad de crucero" para mantener una velocidad constante cuando el usuario no esté aplicando presión al pedal del acelerador, y se registrará el consumo de energía y la velocidad del motor para su posterior análisis.

# REQUISITOS

## 2.1 Objetivos del Sistema

El objetivo principal del sistema es mantener constante la velocidad de giro del motor, independientemente de la presión que establezca el servo.

## 2.2 Componentes involucrados

* Sensor de Velocidad: Se implementará un sensor de velocidad basado en una célula fotoeléctrica que detecta la luz ambiente.
* Motor Eléctrico: Se utilizará un motor eléctrico cuya fuerza generada dependerá de la energía suministrada. La velocidad del motor será controlada por el potenciómetro si el modo crucero no está activado y por el sistema, cuando si esté activado, para así mantener la velocidad deseada del vehículo.
* Controlador: Se desarrollará un controlador para ajustar la velocidad del motor y mantener la velocidad del vehículo constante, independientemente de las fluctuaciones en la carga.
* Display: Mostrar velocidad actual y si el modo crucero está activado o no.
* Servo motor: El sistema tendrá un servomotor que generará una presión determinada al disco dependiendo del valor del potenciómetro.
* Componentes Adicionales: Incluimos un botón para la activación/desactivación y otros dos para el ajuste de la velocidad del sistema. Además un potenciómetro de aceleración para regular la velocidad de giro del dispositivo y otro potenciómetro para regular la posición del servo.

## 

## 2.3 Entradas del sistema

* Señales del sensor de velocidad: El sistema recibirá señales del sensor de velocidad, el cual estará basado en una fotocélula que medirá el tiempo que tarda en detectar el haz de luz después de pasar por un agujero en el disco del motor. Estas señales proporcionarán información precisa sobre la velocidad de giro del vehículo, permitiendo al sistema realizar ajustes necesarios para mantener una velocidad constante.
* Entradas de los botones de control (activación/desactivación, ajuste de velocidad): El sistema contará con botones de control que permitirán al usuario activar o desactivar el control de velocidad, así como ajustar la velocidad de giro del vehículo de manera incremental. Estas entradas serán cruciales para proporcionar al conductor un control directo sobre el sistema, permitiéndole adaptar la velocidad según las condiciones de conducción.
* Ángulo del potenciómetro del motor: El sistema también recibirá información sobre la posición o ángulo en la que se encuentra el potenciómetro del motor. Esta entrada será fundamental para determinar la intención del conductor en cuanto a la aceleración o desaceleración del vehículo. El sistema utilizará esta información sólamente cuando el modo crucero está desactivado.
* Ángulo del potenciómetro del servo: El sistema recibe el valor de la posición o ángulo en la que se encuentra el potenciómetro del servo. Esta entrada será fundamental para determinar el ángulo de giro del servo y por tanto la presión que ejerce sobre el disco.

El sistema utilizará información tanto cuando el modo crucero está activado como cuando no lo está.

* Pulsación de los botones: El sistema reconoce cuando se ha pulsado alguno de los botones y actúa en consecuencia de la función del mismo, consta de 3 botones el cual 1 funciona de manera que capta la velocidad en la que gira el motor y pone el modo crucero a esa velocidad, los otros 2 botones realizan lo mismo pero a una velocidad ya programada, si se vuelve a pulsar el mismo botón se quitara el modo crucero y si se pulsa un botón diferente cuando alguno ya está activado se cambiará a ese modo.

## 

## 2.4 Salidas del sistema

* Señales de control al motor eléctrico: El sistema enviará señales de control al motor eléctrico para ajustar su velocidad según las entradas recibidas del sensor de velocidad, los botones de control y la presión del pedal del acelerador. Estas señales serán generadas por el controlador y se transmitirán al motor eléctrico para mantener una velocidad de giro constante del vehículo, compensando cualquier cambio en la carga o las condiciones de conducción.
* Presión generada por el servomotor:Además, el sistema estará conectado al servomotor, el cual generará una presión determinada sobre el disco del motor. Esta presión influirá en la velocidad de giro del vehículo, y el sistema responderá a los cambios en la presión generada por el servomotor para mantener la velocidad deseada. La información de esta entrada será crucial para realizar ajustes precisos y mantener una velocidad constante independientemente de las condiciones de carga.
* Información mostrada en el display (velocidad actual, estado del sistema): El display del sistema mostrará información relevante para el usuario, incluyendo la velocidad actual del vehículo y el estado del sistema (activado/desactivado). Esta información permitirá al conductor monitorear el rendimiento del sistema y ajustar su conducción en consecuencia. Además, se mostrarán mensajes de alerta o indicaciones cuando sea necesario, proporcionando una experiencia de conducción más segura y controlada.

## 2.5 Requisitos Funcionales

* Medición de la velocidad de giro del vehículo: Se refiere al proceso de obtener datos precisos sobre la velocidad actual del vehículo mediante un sensor de velocidad que emplea una fotocélula para detectar el movimiento del disco del motor.
* Control de velocidad del motor utilizando un potenciómetro: Permite al usuario ajustar la velocidad de rotación del disco en función del valor que establece en el potenciómetro.
* Control de presión del servo utilizando un potenciómetro: Permite controlar el ángulo de posición del servo mediante la posición que el usuario establece en el potenciómetro. De esta forma al aumentar o reducir el ángulo se proporciona más o menos presión al disco.
* Control de la velocidad del motor utilizando un controlador: Se implementará un controlador para mantener constante la velocidad de giro del vehículo. El controlador deberá ajustar la velocidad del motor de manera precisa y rápida para compensar cualquier cambio en la carga del vehículo.
* Activación y desactivación del sistema de control de velocidad: Permite al conductor activar o desactivar el sistema de control de velocidad según sus necesidades y preferencias, proporcionando flexibilidad y control durante la conducción.

## 2.6 Requisitos no Funcionales

* Tiempo de respuesta del sistema: El sistema debe ajustar la velocidad del motor en un tiempo menor a 100 ms para garantizar una respuesta rápida a los cambios en la carga del vehículo.
* Precisión del sensor de velocidad: La medición de la velocidad debe tener una precisión de al menos ±0.1 km/h.
* Interfaz de usuario: El control del sistema debe ser intuitivo y accesible para el usuario.
* Ergonomía: Los controles físicos, como el potenciómetro o los botones, deben ser fáciles de alcanzar y emplear.
* Facilidad de diagnóstico: El sistema debe proporcionar diagnósticos y mensajes de error claros que faciliten la identificación y reparación de problemas.
* Consumo de energía: El sistema debe minimizar el consumo de energía, especialmente cuando el control de velocidad crucero no está activado.
* Costo de implementación: El costo de desarrollo e implementación del sistema debe ser razonable y competitivo en el mercado automotriz.

# 

# ESPECIFICACIONES

## 3.1 Dimensiones y Peso

Las dimensiones físicas del sistema dependen en gran medida de los componentes utilizados y del diseño específico de montaje de la simulación del vehículo. Sin embargo, se buscará un diseño compacto para minimizar el espacio ocupado. Considerando los componentes mencionados, como el motor eléctrico, el sensor de velocidad, el controlador PID y el display, así como los componentes adicionales como los botones de control y el servomotor, se estima que el sistema tenga dimensiones aproximadas de 20 cm de largo, 20 cm de ancho y 15 cm de alto. El peso total del sistema puede rondar los 0,35 kilogramos, aunque esto puede variar según los materiales utilizados en la fabricación y la inclusión de otros componentes opcionales.

## 3.2 Consumo Energético

El consumo energético del sistema dependerá de varios factores, incluyendo la carga del vehículo y la velocidad de giro requerida. Se espera que el sistema sea eficiente en términos de consumo de energía, gracias al control preciso proporcionado por el controlador PID. Sin embargo, el consumo específico variará según las condiciones de conducción. Se estima que el consumo energético promedio esté en el rango de 1-2 vatios durante el funcionamiento normal del sistema.

## 3.3 Controles externos

El sistema estará equipado con varios controles para proporcionar al “conductor” un manejo intuitivo y conveniente del control de velocidad.

Estos controles incluyen:

* Un potenciómetro para regular la velocidad del motor (cuando no está habilitado el modo crucero).
* Un potenciómetro para regular con precisión la presión que ejerce el servo al disco en todo momento.
* Un botón para encender el modo crucero: Al ser pulsado por primera vez establecerá la velocidad en ese instante, como velocidad de crucero.
* Un botón para apagar el modo crucero: Desactiva la velocidad de crucero, y la ajusta al valor establecido por el potenciómetro.
* Un botón para establecer una velocidad de crucero por defecto: Si se pulsa este botón y el modo crucero está activado, se establecerá la velocidad a una velocidad preestablecida.
* Un display LCD en el que el usuario puede ver información sobre el estado del sistema, como la velocidad actual del sistema, si el modo de crucero está activado o no y la velocidad de crucero establecida.

## 3.4 Controles internos

Para poder realizar un sistema de control adecuado, se realiza mediante un sistema operativo en tiempo real, de forma que varios hilos ejecutan diferentes funciones para optimizar la ejecución del código. En este sistema contamos con dos hilos:

* Controlador de motor y servo: Realiza un bucle infinito en el que comprueba si el modo crucero está activado o no y en función de ello actúa en consecuencia ajustando la velocidad en función del potenciómetro o llamando al controlador de velocidad. Una vez ajustada la velocidad del motor procede a comprobar el valor del servo y modificarlo en caso de que haya variado.
* Controlador de display y obtener velocidad: Este otro controlador permite obtener la velocidad del disco, saber si se ha pulsado cualquiera de los botones y muestra la información necesaria del sistema por el display.

# 

# ARQUITECTURA

## 4.1 Diseño general



## 4.2 Diseño Hardware



## 

## 4.3 Diseño Software



# IMPLEMENTACIÓN

## 5.1 Funcionamiento del sistema

Este código de Arduino controla un motor y un servomotor usando potenciómetros para ajustar la velocidad y el ángulo, respectivamente, y muestra la información en una pantalla LCD. Utiliza botones para alternar entre diferentes modos de operación, como el control manual y el modo crucero. El sistema se gestiona mediante tareas de FreeRTOS, donde una tarea controla el motor y el servomotor basándose en la entrada de los potenciómetros y los modos seleccionados, mientras que otra tarea gestiona la pantalla LCD y los botones. Además, se utiliza un fotointerruptor para medir las revoluciones del motor y ajustar la velocidad automáticamente.

## 5.2 Inclusión de bibliotecas y definición de variables

#include <Servo.h>

#include <Arduino\_FreeRTOS.h>

#include <LiquidCrystal\_I2C.h>

#define configTOTAL\_HEAP\_SIZE ((size\_t)(1024))

const int potenciometroPinMotor = A0; // Pin analógico conectado al potenciómetro

const int potenciometroPinServo = A2;

const int fotoPin = 2;

const int pinServo = 3;

const int pinButOnOff = 12;

const int pinButOn1 = 13;

const int pinButOn2 = A1;

const int motorPinStart = 4;

const int motorPinEnd = 11;

volatile bool ButOn1 = false;

volatile bool ButOn2 = false;

volatile int revoluciones = 0;

volatile int revoluciones2 = 0;

volatile bool ModeAutoEnabled = false;

volatile float rpm = 0;

volatile float rpm2 = 0;

unsigned long tiempoAnterior = 0;

unsigned long tiempoAnterior2 = 0;

volatile int lastButtonSpeed = 0;

volatile int velocidades1\_2 = 0;

volatile int velocidad\_variable=0;

TaskHandle\_t displayControlTaskHandle;

TaskHandle\_t ControlTaskHandle;

// Objetos de las bibliotecas

Servo miServo;

LiquidCrystal\_I2C lcd(0x27, 16, 2); // Dirección 0x27, pantalla de 16x2

En esta parte del codigo de incluyen las bibliotecas necesarias para el uso de la pantalla, servomotor y los correspondientes hilos que se crean para el correcto funcionamiento del programa.

## 5.3 Setup

void setup() {

miServo.attach(pinServo);

pinMode(potenciometroPinMotor, INPUT);

pinMode(potenciometroPinServo, INPUT);

pinMode(pinButOnOff, INPUT\_PULLUP);

pinMode(pinButOn1, INPUT\_PULLUP);

pinMode(pinButOn2, INPUT\_PULLUP);

pinMode(fotoPin, INPUT\_PULLUP);

attachInterrupt(digitalPinToInterrupt(fotoPin), contarRevolucion, FALLING);

for (int pin = motorPinStart; pin <= motorPinEnd; ++pin) {

pinMode(pin, OUTPUT);

}

Serial.begin(9600);

lcd.init(); // Inicializa la pantalla LCD

lcd.backlight(); // Enciende la retroiluminación

xTaskCreate(controlTask, "Motor y Servo Control", 128, NULL, 1, &ControlTaskHandle);

xTaskCreate(displayControlTask, "Display Control", 128, NULL, 1, &displayControlTaskHandle);

Serial.println("Creamos los hilos!");

vTaskStartScheduler();

Serial.println("Scheduler no iniciado!");

}

En el setup se configura los pines y el servo, se inicializa la pantalla LCD y se crean dos tareas para el control del motor/servo y la pantalla.

## 5.4 Tarea de control

void controlTask(void \*pvParameters) {

while (true) {

int valorPotenciometroServo = analogRead(potenciometroPinServo);

int valorPotenciometroMotor = analogRead(potenciometroPinMotor);

if (ModeAutoEnabled) {

int velocidadMotor = lastButtonSpeed;

setMotorSpeed(velocidadMotor);

}else if(ButOn1){

int velocidadMotor = velocidades1\_2;

setMotorSpeed(velocidadMotor);

}else if(ButOn2){

int velocidadMotor = velocidades1\_2;

setMotorSpeed(velocidadMotor);

} else {

int velocidadMotor = map(valorPotenciometroMotor, 0, 1023, 0, 255);

lastButtonSpeed = velocidadMotor;

setMotorSpeedSimple(velocidadMotor);

}

servoControlTask(valorPotenciometroServo);

}

}

En esta función se controla el motor y el servo en función del modo y el valor del potenciómetro. Para el motor varía entre el valor del potenciómetro o los 3 diferentes modos del modo crucero.

## 5.5 Control del Servomotor

void servoControlTask(int potValue) {

//int angulo = map(potValue, 0, 1023, 0, 180);

int angulo = map(potValue, 0, 1023, 140, 180);

if (abs(angulo - anngulo\_prev) > 2) {

miServo.write(angulo);

anngulo\_prev = angulo;

}

}

Esta función controla el funcionamiento del servomotor a través del potenciometro.

## 5.6 Ajuste de velocidad del motor en modo automático

void setMotorSpeed(int speed) {

int searched\_rpm = speed \* 10; //velocidad a la que quremos llegar

if (millis() - tiempoAnterior2 >= 1000) {

rpm2 = (revoluciones2 / 2.0) \* 60.0;

revoluciones2 = 0;

tiempoAnterior2 = millis();

if (rpm2 < searched\_rpm ) {

Serial.println("Aumentamos velocidad");

if (searched\_rpm- rpm2 > 100) {

velocidad\_variable = velocidad\_variable + 100;

}else{

velocidad\_variable = velocidad\_variable + 10;

}

Serial.println("velocidad\_variable: " + String(velocidad\_variable));

Serial.println("searched\_rpm: " + String(searched\_rpm));

Serial.println("rpm2: " + String(rpm2));

setMotorSpeedSimple(velocidad\_variable/10);

} else if (rpm2 > searched\_rpm ) {

Serial.println("Reducimos velocidad");

if (rpm2-searched\_rpm > 100) {

velocidad\_variable = velocidad\_variable - 100;

}else{

velocidad\_variable = velocidad\_variable - 10;

}

Serial.println("velocidad\_variable: " + String(velocidad\_variable));

Serial.println("searched\_rpm: " + String(searched\_rpm));

Serial.println("rpm2: " + String(rpm2));

setMotorSpeedSimple(velocidad\_variable/10);

} else {

Serial.println("Control Conseguido");

}

}

}

En esta función si el modo crucero está activado ajusta la velocidad del motor comparando el RPM actual con el RPM deseado.

## 5.7 Ajuste de velocidad del motor en modo por defecto

void setMotorSpeedSimple(int speed) {

int binaryArray[8];

for (int i = 0; i < 8; i++) {

binaryArray[i] = (speed >> i) & 1;

}

int pos = 0;

for (int pin = motorPinStart; pin <= motorPinEnd; ++pin) {

digitalWrite(pin, binaryArray[pos] == 1 ? HIGH : LOW);

pos++;

}

}

Esta función configura los pines del motor según la velocidad deseada en formato binario.

### 

## 5.8 Control de la Pantalla

void displayControlTask(void \*pvParameters) {

while (true) {

bool buttonOnOff = digitalRead(pinButOnOff) == LOW; // Botón presionado

bool button1 = digitalRead(pinButOn1) == LOW; // Botón presionado

bool button2 = digitalRead(pinButOn2) == LOW; // Botón presionado

if (buttonOnOff) {

ModeAutoEnabled = !ModeAutoEnabled; // Cambiar el modo automático

velocidad\_variable=rpm;

ButOn1 = false;

ButOn2 = false;

vTaskDelay(pdMS\_TO\_TICKS(200)); // Debounce delay

}else if(button1){

ButOn1 = !ButOn1;

velocidades1\_2 = 20;

ModeAutoEnabled = false;

ButOn2 = false;

vTaskDelay(pdMS\_TO\_TICKS(200)); // Debounce delay

}else if(button2){

ButOn2 = !ButOn2;

velocidades1\_2 = 100;

ModeAutoEnabled = false;

ButOn1 = false;

vTaskDelay(pdMS\_TO\_TICKS(200)); // Debounce delay

}

if (millis() - tiempoAnterior >= 1000) {

rpm = (revoluciones / 2.0) \* 60.0;

revoluciones = 0;

lcd.setCursor(0, 0);

lcd.print("RPM: ");

lcd.print(rpm);

lcd.print(" "); // Añade espacios en blanco para borrar dígitos previos

tiempoAnterior = millis();

}

lcd.setCursor(0, 1);

if (ModeAutoEnabled) {

lcd.print("Cruise speed ON ");

} else {

lcd.print("Cruise speed OFF");

}

vTaskDelay(pdMS\_TO\_TICKS(100)); // Retardo entre verificaciones del control remoto

}

}

Esta tarea controla el contenido de la pantalla LCD y lee el estado de los botones para cambiar modos y mostrar información llamando a las correspondientes funciones.

## 5.9 Contador de Revoluciones

void contarRevolucion() {

revoluciones++;

revoluciones2++;

}

Incrementa los contadores de revoluciones cada vez que se detecta una interrupción en el pin del fotointerruptor para poder obtener la velocidad del motor.

## 5.10 Proceso de implementación

El proyecto comenzó con el desarrollo individual del código para el motor y el servomotor, sin problemas iniciales. Sin embargo cuando se juntaron ambas funcionalidades se detectó que el servomotor se movía de manera descontrolada debido a los valores del potenciómetro, lo que se controló realizando una comprobación de la variación del movimiento. Posteriormente, se integró un sensor de luz para medir las revoluciones por minuto (RPM) del motor, sin complicaciones. La inclusión del código para la pantalla LCD presentó desafíos debido a incompatibilidades entre librerías, pero finalmente se logró mostrar correctamente las RPM.

Un contratiempo significativo surgió cuando el Arduino Nano dejó de funcionar, obligando a cambiar a un Arduino Uno, tras realizar el cambio de todos los componentes uno a uno el sistema volvió a operar correctamente.

*Durante el proceso implementamos un control remoto con infrarrojos para gestionar el modo crucero y establecer velocidades por defecto. Permitiendo encender el modo crucero con el botón de encender/apagar, y al pulsar cualquiera de los botones de numerados del 0 al 9 se establecía la velocidad proporcional a este índice en el motor.*

void remoteControll(uint32\_t decodedRawData) {

switch (decodedRawData) {

case apagar\_encender:

if (ModeAuto) {

ModeAuto = false;

} else {

lastButtonSpeed=motorSpeed;

setMotorSpeed(lastButtonSpeed);

ModeAuto = true;

}

break;

case boton\_cero:

lastButtonSpeed = 0;

setMotorSpeed(lastButtonSpeed);

break;

case boton\_uno:

lastButtonSpeed = 28;

setMotorSpeed(lastButtonSpeed);

break;

//Resto de botones

case boton\_ocho:

lastButtonSpeed = 226;

setMotorSpeed(lastButtonSpeed);

break;

case boton\_nueve:

lastButtonSpeed = 255;

setMotorSpeed(lastButtonSpeed);

break;

default:

break;

}

}

A continuación, debido a que como un solo hilo ejecutaba todas las tareas, las acciones que realizaba el sistema, no se realizaban en tiempo real. Se decidió realizar la implementación del código empleando FreeRTOS, de forma que un hilo se encargara de la gestión del motor y del servo y otro hilo, de la comprobación del estado de los botones y los datos que se muestran por el display.

Al integrar la biblioteca de FreeRTOS, surgieron conflictos con la biblioteca “IRremote.h” empleada para gestionar el control remoto, plo que llevó a eliminar esta funcionalidad y reemplazarla con tres botones físicos: un botón para poner el modo de crucero en la velocidad que en ese momento se encuentra el motor y otros dos botones los cuales se ponen a una velocidad programada por defecto.