UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

ESCUELA DE INGENIERÍA EN MECATRÓNICA

TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN MECATRÓNICA

TEMA:

"SISTEMA DE MONITOREO EN TIEMPO REAL DE VARIABLES DE FUNCIONAMIENTO PARA BICICLETAS MOTORIZADAS"

AUTOR: SANTIAGO GABRIEL ERAZO SOLANO

DIRECTOR: CARLOS XAVIER ROSERO CHANDI

IBARRA-ECUADOR

JULIO 2019

UNIVERSIDAD TÉCNICA DEL NORTE
BIBLIOTECA UNIVERSITARIA
AUTORIZACIÓN DE USO Y PUBLICACIÓN
A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

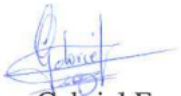
En cumplimiento del Art.144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a dispocisión la siguiente información:

DATOS DEL AUTOR			
CÉDULA DE IDENTIDAD:	0504181884		
APELLIDOS Y NOMBRES:	ERAZO SOLANO SANTIAGO GABRIEL		
DIRECCIÓN:	ANDRADE MARÍN		
EMAIL:	sgerazos@utn.edu.ec - santiago6893@gmail.com		
TELÉFONO FIJO:	062530242	**TELÉFONO MÓVIL:**	0995440171
DATOS DE LA OBRA			
TÍTULO:	"SISTEMA DE MONITOREO EN TIEMPO REAL DE VARIABLES DE FUNCIONAMIENTO PARA BICICLETAS MOTORIZADAS"		
AUTOR:	SANTIAGO GABRIEL ERAZO SOLANO		
FECHA (AAAA-MM-DD):	2019-07-15		
SÓLO PARA TRABAJOS DE GRADO			
PROGRAMA:	PREGRADO		
TÍTULO POR EL QUE OPTA:	INGENIERO EN MECATRÓNICA		
ASESOR/DIRECTOR:	CARLOS XAVIER ROSERO CHANDI		

2. CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló sin violar derechos de autor de terceros, por lo tanto la obra es original, y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 15 días del mes de julio de 2019.



Santiago Gabriel Erazo Solano
C.I.: 0504181884

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CERTIFICACIÓN

En calidad de director del trabajo de grado “SISTEMA DE MONITOREO EN TIEMPO REAL DE VARIABLES DE FUNCIONAMIENTO PARA BICICLETAS MOTORIZADAS”, presentado por el egresado SANTIAGO GABRIEL ERAZO SOLANO, para optar por el título de Ingeniero en Mecatrónica, certifico que el mencionado proyecto fue realizado bajo mi dirección.

Ibarra, julio de 2019

Carlos Xavier Rosero
DIRECTOR DE TESIS

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

DECLARACIÓN

Yo, Santiago Gabriel Erazo Solano con cédula de identidad Nro. 0504181884, declaro bajo juramento que el trabajo aquí escrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Universidad Técnica del Norte - Ibarra, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normativa institucional vigente.

Ibarra, julio de 2019

Santiago Gabriel Erazo Solano
C.I.: 0504181884

Agradecimiento

Quiero agradecer a mis padres por demostrarme que con trabajo y esfuerzo constante las metas propuestas llegan a verse materializadas, a mis hermanas y hermanos por brindarme la motivación y el apoyo en cada momento que así lo requería, y a todos esos amigos por las experiencias vividas en este transcurso de formación académica.

Al los docentes de la Carrera de Ingeniería en Mecatrónica por impartir su conocimiento en las aulas de clase, y al Grupo de Investigación en Sistemas Inteligentes por la guía brindada en el desarrollo de este trabajo.

Gabriel

Dedicatoria

Este trabajo va dedicado a mi madre, quien con paciencia y amor ha sabido acompañarme durante toda una vida y especialmente durante el proceso de formación profesional.

Gabriel

Resumen

Los sistemas embebidos se construyen generalmente sobre una arquitectura monoprocesador y adicionalmente suelen emplearse sensores y actuadores que le permiten una interacción continua con su entorno. Este tipo de sistemas se encuentra orientado a satisfacer los requerimientos de una aplicación en concreto debido a la limitación de recursos que estos presentan. Entre los campos de aplicación de los sistemas embebidos se encuentra el sector automotriz para el monitoreo y control de las variables de funcionamiento de diferentes tipos de vehículo. En el presente trabajo se desarrolló un dispositivo de monitoreo en tiempo real para ciclomotores que permite mostrar la información al usuario sobre el estado de diferentes parámetros de funcionamiento del vehículo tales como temperatura, régimen de giro del motor, velocidad y tiempo de funcionamiento; todo esto con la finalidad de hacer un uso medido del motor y evitar fallos o una rotura del mismo por sobreexigencia durante la marcha.

Abstract

Embedded systems are usually built on a monoprocessor architecture and additionally, sensors and actuators are usually used that allow continuous interaction with their environment. This type of systems is oriented to satisfy the requirements of a specific application due to the limited resources they present. Among the fields of application of embedded systems is the automotive sector for monitoring and controlling the operating variables of different types of vehicles. In the present work, a real-time monitoring device for mopeds was developed that allows the user to be informed about the status of different vehicle operating parameters such as temperature, engine speed, speed and operating time; all this in order to make a measured use of the engine and avoid failures or a breakage due to over-demand on the run.

Índice general

Índice de figuras

Índice de cuadros

Lista de Programas

Capítulo 1

Introducción

Este trabajo de grado ha sido realizado con el *Grupo de Investigación en Sistemas Inteligentes de la Universidad Técnica del Norte (GISI-UTN).*

1.1. Problema

En las dos últimas décadas se ha podido apreciar la reintroducción de las bicicletas motorizadas en espacios urbanos dado que muchos usuarios se han visto atraídos en obtener este tipo de transporte, ya sea mediante la compra de alguno de sus modelos comerciales o a través de kits para adaptación en bicicletas de uso convencional [1]. Según Juan Carlos Montalvo, gerente de *Bobber Motorcycles* en la ciudad de Ibarra, este tipo de bicicletas ha tenido gran acogida en la ciudad antes mencionada debido a que los usuarios las consideran como una alternativa de transporte amigable con el ambiente, que posee un reducido consumo de combustible y por tanto representa una menor inversión por recorrido con respecto a un vehículo particular. Este medio alternativo de movilidad brinda mayores prestaciones al usuario con respecto a las bicicletas convencionales: menor agotamiento físico, mayor confort y reducción del tiempo de recorrido. Sin embargo, como cualquier otro vehículo, requiere ser inspeccionada periódicamente con el fin de determinar su estado, mantener el motor con buen rendimiento y prolongar su ciclo de funcionamiento. Gran parte de estas inspecciones pueden verse simplificadas mediante la im-

plementación de un sistema de monitoreo de las variables propias de una bicicleta motorizada: velocidad a la que se desplaza, horas de funcionamiento, revoluciones y temperatura del motor. Actualmente, solo existen en el mercado sistemas de monitoreo para bicicletas con motor eléctrico, mas no para aquellas con motor de combustión interna, dejando a estas últimas sin un medio para monitorear su estado.

Surge entonces un inconveniente. Al carecer de un medio de monitoreo, el usuario desconoce si el motor se encuentra trabajando dentro del régimen de funcionamiento recomendado, o si la bicicleta se desplaza a velocidades permitidas dentro de espacios urbanos. Para dar una solución a esto, se dispone en el mercado de sistemas que podrían ser acoplados a las bicicletas motorizadas, sin embargo, su costo es relativamente elevado debido a su baja disponibilidad en el país. Además, ninguno de estos integra todas las funciones en un mismo dispositivo, encareciéndolos aún más y limitando su implementación.

1.2. Objetivos

1.2.1. Objetivo general

Desarrollar un sistema de monitoreo en tiempo real utilizando hardware abierto para el monitoreo de variables de funcionamiento en bicicletas motorizadas.

1.2.2. Objetivos específicos

- Establecer la estrategia de monitoreo en función de las variables correspondientes y la capacidad computacional del hardware.
- Desarrollar el sistema usando una plataforma de hardware abierto que incorpore elementos de bajo costo y de fácil adquisición.
- Acoplar el sistema de monitoreo a una bicicleta motorizada para someterlo a condiciones

reales de funcionamiento.

1.3. Justificación

El uso de sistemas en tiempo real en el transporte ayuda a asistir el vuelo de aviones, organizar el arribo de trenes y controlar el motor o los frenos en un automóvil. Observar la información del estado del vehículo implica una mayor confiabilidad sobre su funcionamiento e incluso un avance en la seguridad para el usuario. Bajo este concepto, a lo largo de los años se han ido mejorando los sistemas de información que proveen al conductor la información necesaria a la hora de conducir un vehículo.

El desarrollo del presente trabajo permitirá al usuario realizar un uso medido de los recursos del motor y de la bicicleta en sí. Además, el sistema propuesto podría ser extendido, no solo sirviendo para la medición de variables en una bicicleta motorizada, sino a la aplicación en sistemas industriales en donde la ejecución de tareas en tiempo real es esencial para ciertos procesos.

1.4. Alcance

El sistema será desarrollado en una plataforma de hardware abierto, utilizando un solo procesador que mantenga diferentes tiempos de moitoreo para cada variable. El sistema será puesto a prueba en una bicicleta BOBBER con motor de combustión interna para su respectiva validación. Los sensores y demás elementos acoplados deberán ser de bajo costo y de alta disponibilidad en el mercado.

1.5. Estructura del documento

Este documento se encuentra estructurado por cinco capítulos y un apéndice. El Capítulo 1 muestra la presente introducción, en donde se detalla el problema, obejtivos, justificación, alcance y estructura del documento.

En el Capítulo 2 se hace una recopilación de trabajos relacionados y se detalla el sustento teórico utilizado como marco de referencia para el desarrollo del sistema propuesto.

El Capítulo 3 describe el proceso de co-diseño de las arquitecturas de hardware y software del sistema a desarrollar.

En el Capítulo 4 muestra la implementación del sistema de monitoreo desarrollado, abordando la metodología mostrada previamente en el Capítulo 3. Además se presentan los resultados obtenidos a partir de las pruebas realizadas en condiciones reales de funcionamiento.

Finalmente, en el Capítulo 5 se proporciona las conclusiones y recomendaciones de este desarrollo y se señalan posibles líneas de trabajo futuro.

El Apéndice recopila el código implementado sobre el microcontrolador, el diagrama electrónico del sistema y los planos mecánicos de las estructuras elaboradas.

Capítulo 2

Revisión Literaria

Los motores de combustión interna están sometidos a diferentes condiciones de servicio tales como su temperatura de trabajo, régimen de giro, presión en las cámaras de lubricación y combustrión, vibraciones, entre otras. Varias de estas condiciones deben mantenerse dentro de unos límites especificados con la finalidad de no ocasionar fallos por sobreexigencia del motor. Resulta entonces necesario disponer de un mecanismo que permita monitorear su estado e informe oportunamente al usuario, quien ejecutará las acciones pertinentes. En este capítulo se resumen las alternativas desarrolladas dentro del entorno comercial y de investigación referente a sistemas de monitoreo para motocicletas, además de los aspectos fundamentales a considerar en entornos multitarea.

2.1. Antecedentes

Estudios preliminares [2]-[4] describen sistemas de monitoreo y control que brindan retroalimentación al conductor sobre el estado de funcionamiento del vehículo y/o motocicleta. Por ejemplo: en [2] se desarrolló un sistema odométrico/temporizador que informa al ususario sobre su desempeño en competencias de enduro. Una aproximación más elaborada puede encontrarse en [3] donde el sistema integra funcionalidades de odómetro y velocímetro en un solo microprocesador, además de varias características adicionales de configuración. En [4] se detallan

las partes constitutivas de una unidad de control electrónico para motocicletas, siendo de especial interés para el presente estudio el bloque sensorizado que se emplea para el monitoreo del motor. Posteriores modificaciones sobre estas patentes constituyeron el mejoramiento de las mismas, que junto al desarrollo de nuevas tecnologías, se adaptarían a mayores requerimientos en los distintos tipos de vehículo. Así, en [5] se logró obtener un entorno visual multifunción basado en un sistema monoprocesador. Tal sistema es capaz de adquirir, procesar y mostrar los resultados de variables medidas como: velocidad, régimen de giro, kilometraje, distancia y tiempo de recorrido, presión del aceite, además de otras funcionalidades. De manera similar, el dispositivo desarrollado en [6] incorpora varias de las funcionalidades anteriormente descritas y agrega la medición de una variable adicional siendo esta la temperatura del refrigerante.

Por su parte, los trabajos de investigación disponibles se han enfocado primordialmente al mejoramiento de las técnicas de monitoreo convencionales, adoptando tecnologías inalámbricas para la transmisión y almacenamiento remoto de la información. Tal es el caso presentado en [7], en el cual se describe el proceso de diseño e implementación de un sistema de monitoreo en línea que registra la información obtenida a través de los sensores impantados en el vehículo. Su propósito es el de incrementar la utilidad de la información organizándola, resumiéndola y analizándola. En [8], Bingamil et al. definen las variables a medir y las técnicas de procesamiento de datos a los cuales deben recurrirse para la detección de fallos en motores de combustión interna. El resultado una clasificación de las técnicas de procesamiento más adecuadas conforme a la variable medida.

2.1.1. Patentes desarrolladas

2.1.1.1. Odómetro/Velocímetro Basado en Microcontrolador

El sistema propuesto en [3] posibilita la configuración de parámetros de velocidad como: límite permitido, consigna de alerta, y regulación autónoma de la misma. El proceso de acondicionamiento de las señales de entrada al microcontrolador consta de un filtro pasa bajas, un

supresor de voltajes inversos, un circuito de acoplamiento a DC, entre otros, tal como se muestra en la Fig 2.1 La señal tratada es entonces receptada y procesada para finalmente obtener el valor de frecuencia de la misma. Dicho valor es posteriormente transformado por un conversor de frecuencia a voltaje que controla la posición del indicador del velocímetro. Luego este mismo voltage es retransformado a una señal de distinta frecuencia para el control de un odómetro rotativo.

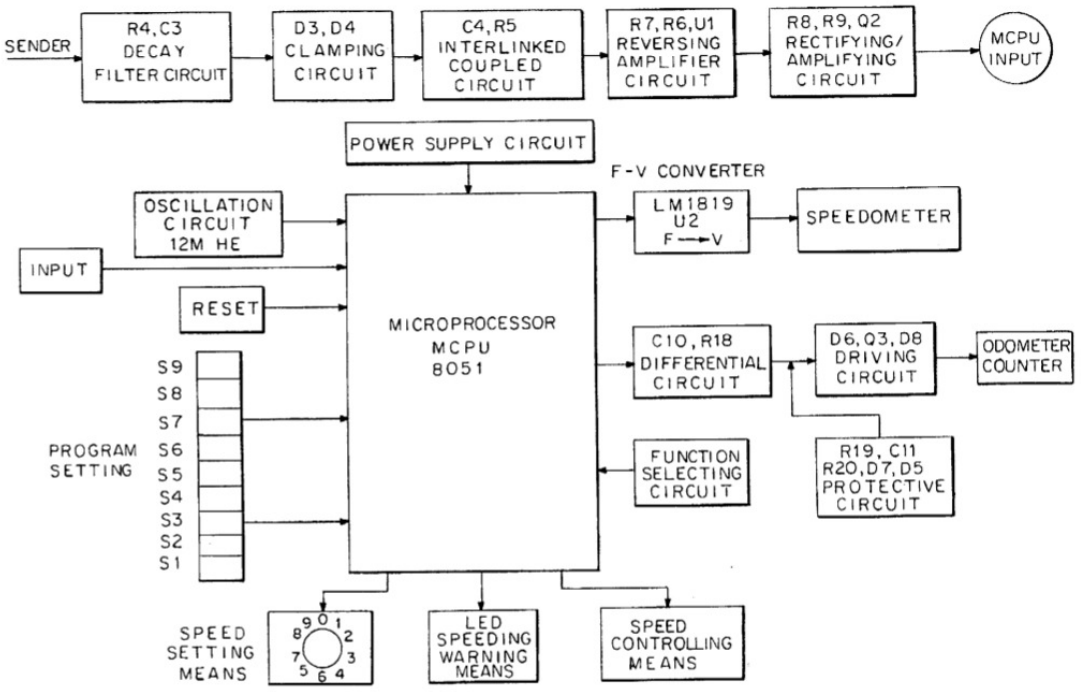
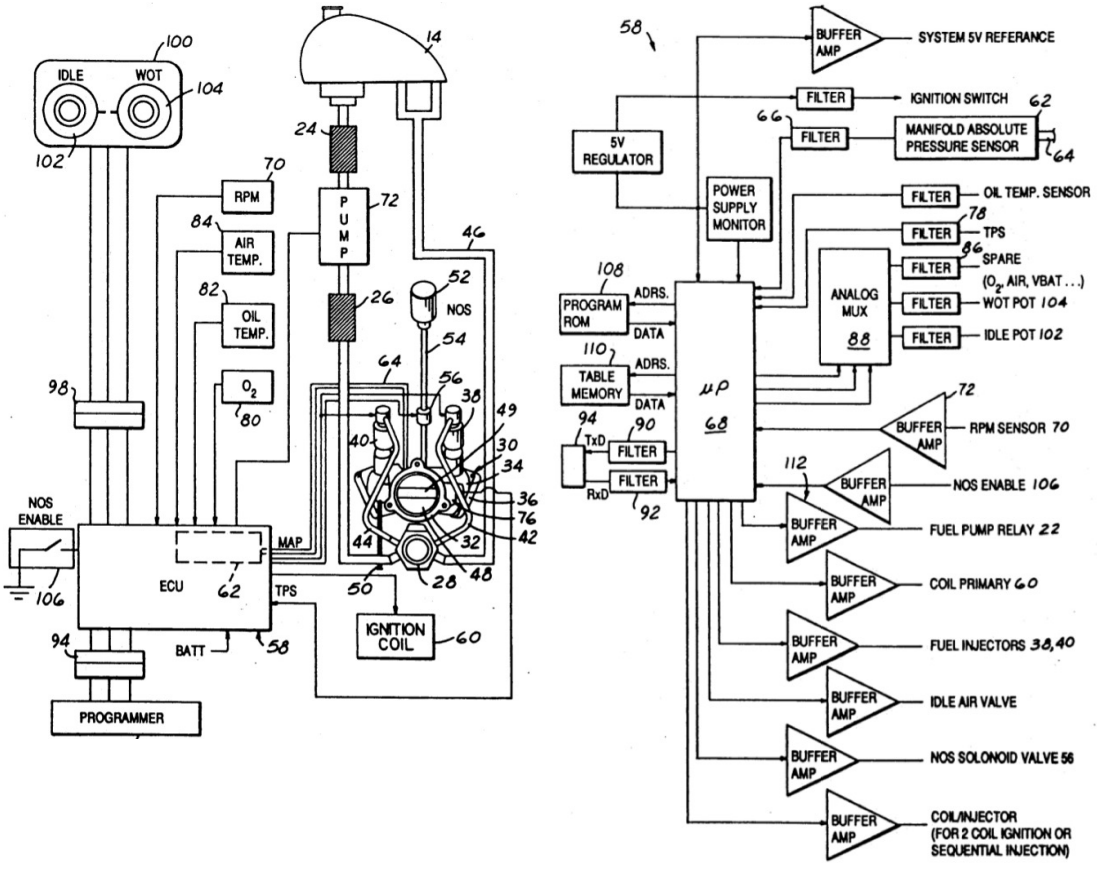


Figura 2.1: Diagrama de bloques del dispositivo odómetro/velocímetro propuesto en [3]

2.1.1.2. Sistema de Gestión del Motor para Motocicletas

En [4] el sistema está comprendido por la unidad de control electrónico (ECU)[1] que recibe como entrada las señales de varios sensores destinados a la medición de las condiciones de funcionamiento del motor, y provee las señales de control respectivas que permiten operar los inyectores y válvulas de combustible, y la bobina de encendido. Para ello utiliza un sensor de presión (ubicado en el colector de admisión) acoplado mediante filtro de señal hacia el microcontrolador. De la misma forma un sensor de revoluciones[2] provee una señal correspondiente a

la velocidad del motor a través de un circuito amplificador hasta una de las entradas del sistema.



(a) (b)

Figura 2.2: Descripción del sistema propuesto en [4]. (a) Conexiones, (b) diagrama de bloques.

2.1.1.3. Sistema Multifuncional de Medición en Pantalla para Motocicletas

Este sistema consta de un tacómetro y velocímetro guiados por motores a pasos [5]. La descripción del dispositivo establece como unidad de control al microprocesador PIC16C73A20 de Microchip. Aquí, tanto la velocidad de desplazamiento como revoluciones del motor se miden a través de sensores de efecto hall capaces de proporcionar una señal de onda cuadrada de

[1]Engine Control Unit.

[2]El sensor de revoluciones está conformado por un sensor de efecto hall que responde al giro del árbol de levas del motor para proporcionar una señal pulsante.

frecuencia variable a la unidad de control del vehículo. Finalmente, la información mostrada en la LCD conmuta entre los modos: odómetro, tiempo y distancia de viaje, nivel de combustible y nivel de voltaje.

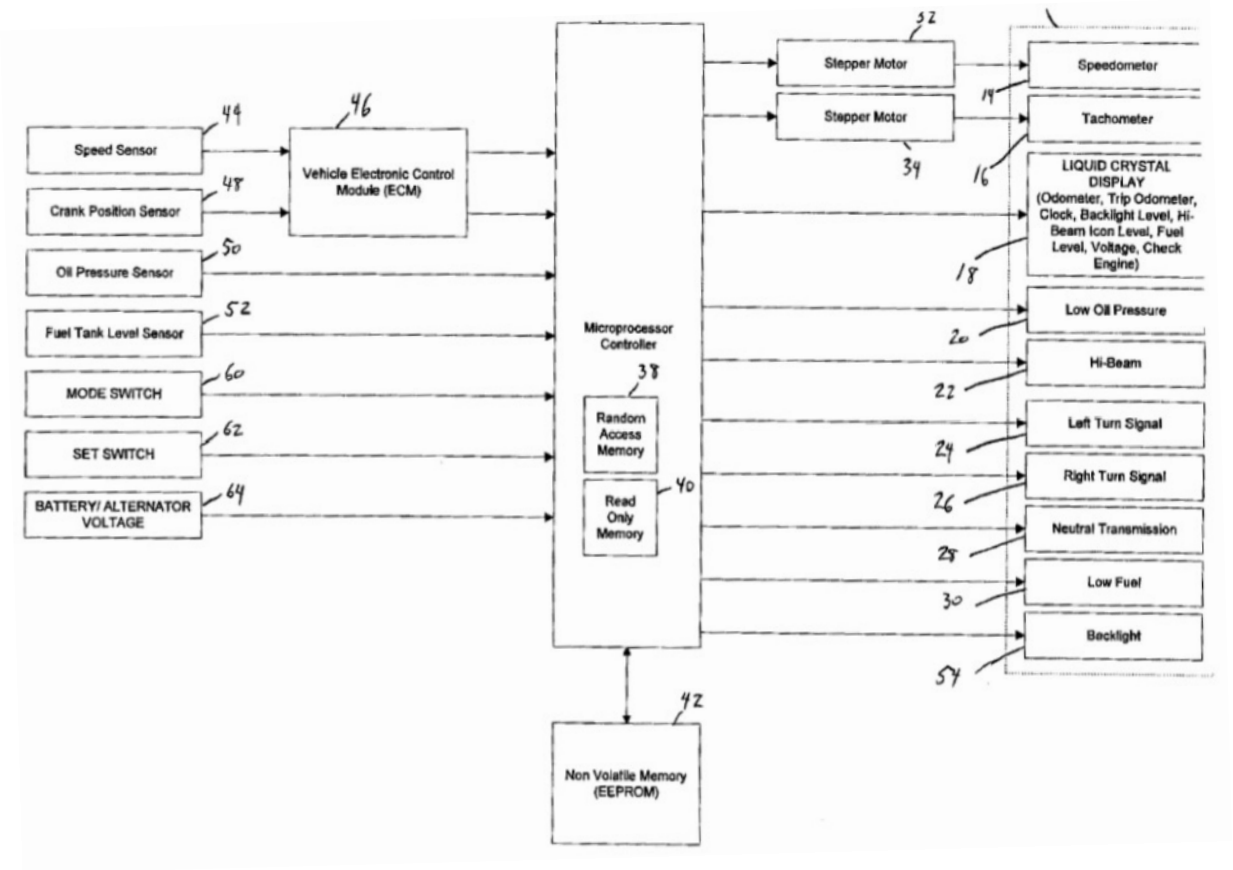


Figura 2.3: Diagrama de bloques del sistema propuesto en [5] .

2.1.1.4. Sistema Odométrico para Vehículos y Metodología de Medición

Este dispositivo se encuentra conectado a una interfaz sensorizada conformada por un sensor de velocidad que genera una señal pulsante conforme a la rotación de un eje, sensores de nivel de combustible, temperatura del aceite, régimen de giro del motor, entre otros adicionales. Los medidores: odómetro, velocímetro, distancia y tiempo de viaje, y termómetro son visualizados a través de pantallas LCD de 7 segmentos. La unidad de control comprende de una CPU,

memorias (RAM, ROM, EEPROM), un circuito divisor de pulso, y el controlador para LCD.

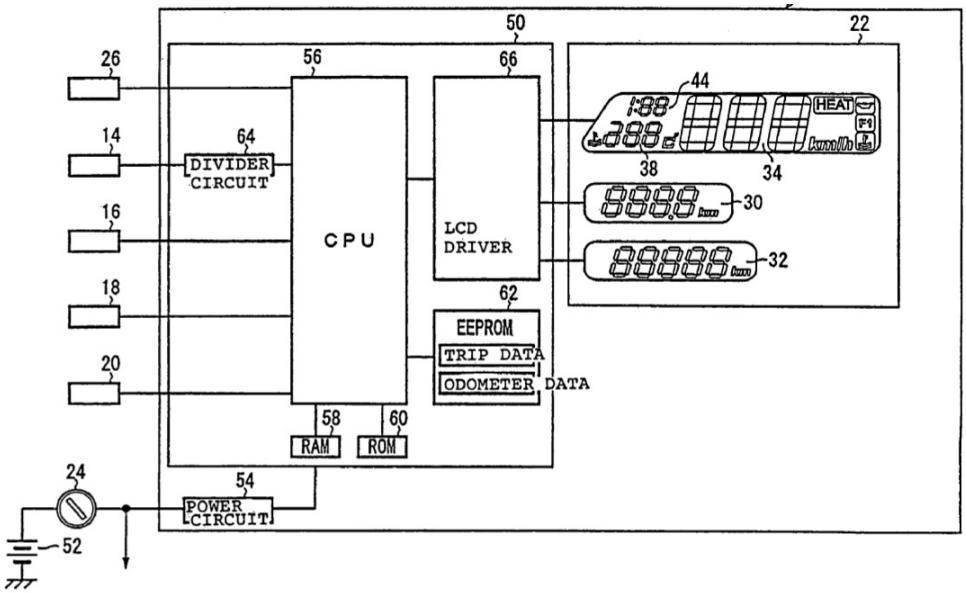


Figura 2.4: Diagrama de bloques del sistema propuesto en [6].

2.1.2. Sistemas de monitoreo comerciales

Es posible encontrar varios dispositivos comerciales que permiten monitorear el estado de servicio del ciclomotor. Muchas de ellas son de uso universal, por lo que no están destinadas únicamente al monitoreo de ciclomotores sino que también pueden ser acoplados sobre cualquier medio que utilice un motor de combustión de características similares. Sin embargo, dicha versatilidad ocasiona una limitación en la cantidad de variables a medir, por ejemplo un sistema que permita monitorear el régimen de giro y temperatura del motor puede ser aplicado tanto en un generador a gasolina como en una motocicleta, más en este último debe considerarse la medición de la velocidad de desplazamiento, variable que resulta innecesaria de medir en un generador.

Dado que el presente estudio contempla la medición de cuatro variables de funcionamiento dentro de un mismo sistema, se ha resumido una tabla comparativa con algunos de los dispo-

sitivos comerciales que más se ajustan a los requerimientos para el monitoreo del ciclomotor. Debido a la ausencia de distribuidores locales de este tipo de sistemas, la información resumida en la Tabla 2.1 corresponde a la otorgada por proveedores en linea para Ecuador.

Tabla 2.1: Dispositivos de monitoreo comerciales

Referencia	Marca	Medición[1]					Precio +imp.	Requerimientos		Precio total
		T	R	V	H	O		Sens.	Bat.	
SR-HM035LTD	SEARON	x	x		x		$80.06	$7.25	$0.00	$87.25
732-ET1	TrailTech	x	x				$67.53	$7.25	$0.00	$74.72
RPM1204BHA	DIGITEN		x	x			$44.06	$0.00	$16.00	$60.00
CB-11014001	CarBole		x		x		$48.06	$0.00	$0.00	$48.00
43483-17184	Samdo		x	x		x	$67.57	$0.00	$16.00	$83.51
RL-HM005L	RunLeader		x		x	x	$55.06	$0.00	$0.00	$55.00
SA034-1	Sdootauto		x	x		x	$56.06	$0.00	$16.00	$72.00
B07JZ93XD4	Keenso		x	x		x	$72.06	$0.00	$0.00	$72.00
BA010001	Koso		x	x		x	$352.91	$0.00	$16.00	$368.85
910-00129	Eling	x	x	x	x	x	$205.06	$7.25	$16.00	$228.25

De estos datos presentados es necesario aclarar los siguientes aspectos: el impuesto agregado a cada producto corresponde a un valor de $25.07 por concepto de tributación en aduana; varios de estos sistemas no incluyen sensores, por lo que deben adquirirse por separado; para aquellos que no incluyen baterías (generalmente celdas de tipo botón o baterías AAA) es necesario adquirir algún medio de energización a 12 V. Con esta información se puede consatar que la solución comercial que más se ajusta a los requerimientos del caso es el dispositivo de la marca Eling por un costo total de USD 228,25.

[1] Variables medidas por el dispositivo:
T = temperatura
R = régimen de giro
V = velocidad de desplazamiento
H = horas de operatividad del motor
O = otros

2.2. Sistemas de tiempo real

En su publicación, Nuria define a los sistemas de tiempo real como:

> Los sistemas de tiempo real (STR) son sistemas de computación que interactúan repetidamente con su entorno físico y responden a los estímulos que reciben del mismo dentro de un plazo determinado. Para que el funcionamiento sea correcto no basta con que las acciones sean correctas, sino que tienen que ejecutarse dentro de un intervalo de tiempo especificado [9].

2.2.1. Hard real-time y soft real-time

Las restricciones temporales son las más comunes en los sistemas de tiempo real, destacándose de entre estos parámetros el plazo de ejecución de una tarea o deadline [9]. De acuerdo a los efectos producidos en el sistemas por la pérdida de un plazo, los STR pueden categorizarse como estrictos (hard real-time systems) y no estrictos (soft real-time systems). En un STR estricto el tiempo de respuesta es crítico, lo cual implica que si una tarea no se ejecuta dentro del plazo establecido las consecuencias pueden resultar catastróficas para el sistema. Por otra parte, para un STR no estricto el tiempo de respuesta es importante mas no crítico, por lo que la pérdida ocasional de los plazos no genera mayores inconvenientes dentro del funcionamiento general del sistema. Jiménez, et al. [10] concluyen que:

> Los sistemas de tiempo real estrictos imponen un tiempo de respuesta determinista, permitiendo un comportamiento en caso de sobrecarga predecible, aunque supone generalmente trabajar con un volumen de datos reducido. Por el contrario un sistema de tiempo real no estricto no actúa de forma tan determinista, presentando algún tipo de degradación cuando se producen sobracargas. A cambio permiten trabajar con un mayor volumen de datos.

2.2.2. Sistemas controlados por tiempo o por eventos

El mismo estudio [10], muestra que los sistemas de tiempo real pueden clasificarse de acuerdo al uso de eventos o de ciclos de reloj como elemento de disparo para la ejecución de tareas. Pueden distinguirse entonces dos arquitecturas: una controlada por eventos o interrupciones (event-driven) y otra controlada por tiempo (time-driven).

En una arquitectura controlada por eventos, la ejecución de una tarea se efectúa solo ante la la ocurrencia de un determinado evento externo que es atendido por el mecanismo de interrupciones del computador. Por otra parte, una arquitectura controlada por tiempo reacciona ante los elementos de temporización del sistema, dando paso a la ejecución de una tarea cada vez que se alcanza una consigna de reloj predefinida.

A pesar que las características de estas arquitecturas difieren una de otra, resulta habitual que las dos puedan coexistir dentro de un mismo sistema.

2.2.3. Planificación cooperativa o Timeline Scheduling

Un planificador cooperativo está constituido por tablas de planificación o planes de ejecución, y estos a su vez están estructurados por el conjunto de tareas del sistema y sus respectivas restricciones temporales. De esta forma, con base en los tiempos de ejecución y los períodos de las tareas es posible diseñar un plan de ejecución fijo [9]. Dicho plan de ejecución está definido mediante un ciclo principal conocido como ciclo mayor y es el mínimo común múltiplo de los períodos. Un ciclo mayor está conformado por uno o más ciclos menores en los cuales se alojan las actividades de las diferentes tareas. Generalmente la duración de un ciclo menor debe ser igual al máximo común divisor de los períodos [11], sin embargo en [12] Baker y Shaw argumentan: “resulta interesante apreciar que el ciclo menor no es necesariamente tan corto como lo sea el máximo común divisor de los períodos...”, dejando al criterio del diseñador establecer una temporización adecuada.

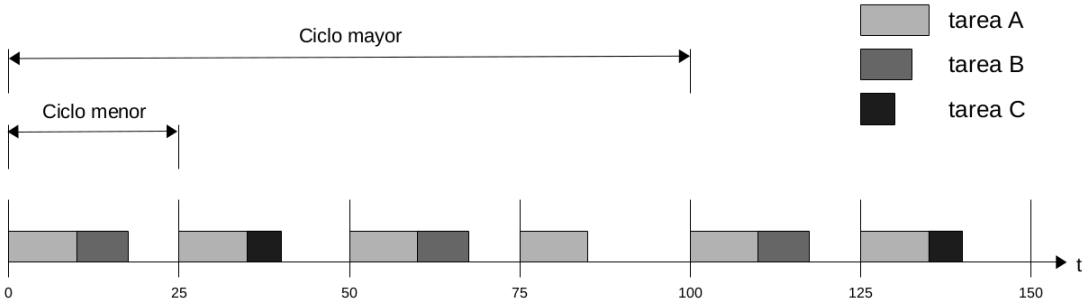


Figura 2.5: Organización de tareas mediante planificación cooperativa [11]

Como ejemplo, en la Fig. 2.5 se esquematiza la organización de las tareas mediante planificación cooperativa en donde las tareas A, B y C deben ejecutarse con períodos de 25, 50 y 100 ms respectivamente. Se puede apreciar entonces que el intervalo de tiempo óptimo para el ciclo menor es de 25 ms que es el máximo común divisor de los períodos [15]. De esta forma la solución supondría ejecutar la tarea A en cada intervalo, la tarea B cada dos intervalos, y la tarea C cada cuatro intervalos.

Para garantizar a priori la planificabilidad del algoritmo sobre un procesador en particular, bastará con conocer los tiempos de ejecución de las tareas para el caso más desfavorable y verificar que las suma de los tiempos de ejecución sea menor o igual a la duración del ciclo menor [11].

2.3. Sistemas multitarea

Diaz [16] define a los sistemas multitarea como sistemas orientados a la gestión de tareas que permiten repartir el uso de los recursos de procesado (CPU) disponibles entre las diferentes tareas que el sistema debe realizar.

2.3.1. Estados de las tareas

Las tareas son generealmente estructuras en bucle que pueden tomar distintos estados: bloqueda, disponible o en ejecución [16]. La Fig. 2.6 muestra un diagrama de los estados típicos de las tareas y posteriormente se brinda una descripción de las mismas.

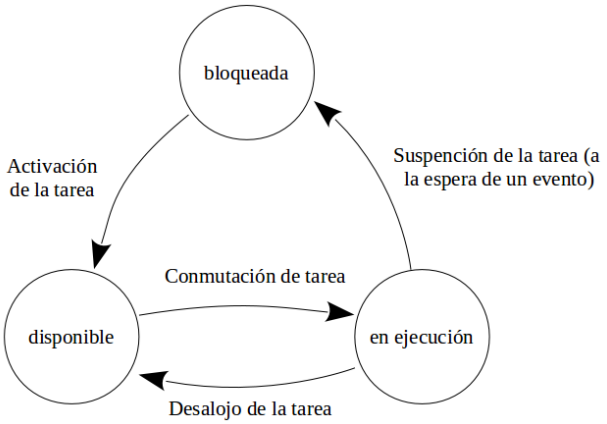


Figura 2.6: Estados de las tareas [17]

- En ejecución (running): es el estado en el cual la tarea tiene acceso al procesador (CPU) y está siendo ejecutada.
- Disponible (ready): la tarea se encuentra lista para su ejecución, pero esta debe esperar a que el planificador decida darle acceso a la CPU.
- Bloqueada (waiting): se encuentran en este estado las tareas que requieran de la ocurrencia de un evento para cambiar a estado ready e iniciar o continuar con su ejecución normal.

2.3.2. Sincronización de tareas

2.3.2.1. Semáforos

Son variables de estado que conforman un mecanismo de comunicación entre tareas para gestionar el acceso a un recurso compartido. Existen dos tipos de semáforo de acuerdo a la

función que estos desempeñen:

- Semáforo binario: es una variable que puede tomar únicamente dos estados, abierto (1) o cerrado (0). Su funcionamiento se basa en permitir o denegar la ejecución de una tarea dependiendo del estado de su semáforo (una tarea con semáforo abierto pasa a estar en ejecución, mientras que si este se encuentra cerrado la tarea se mantiene a la espera hasta que su semáforo cambie de estado).

- Semáforo de exclusión mutua: este mecanismo de sincronización impide que dos tareas accedan a un mismo recurso compartido simultáneamente. De esta forma, la tarea que tenga acceso al recurso hará uso de él y ninguna otra tarea podrá acceder al mismo hasta que dicho recurso haya sido devuelto.

2.3.2.2. Flags de evento

Un flag de evento es un bit que se activa tras la ocurrencia de un suceso determinado. Dicho bit puede ser modificado por una tarea o una ISR conforme la aplicación lo requiera. Este mecanismo es especialmente útil cuando se desea activar una tarea solo si se han cumplido uno o múltiples eventos necesarios para pasarla a un estado de ejecución.

Una plataforma embebida usualmente se construye a base de una única CPU sobre la cual deben programarse todas las tareas. Esto implica que solo una tarea puede tener acceso a los recursos de procesamiento en un momento determinado, y por tanto solo una se ejecutará en un dicho instante [14]. Tales restricciones temporales son solventadas abordando el paradigma de ejecución tiempo real, en donde además de garantizar los tiempos de respuesta, permite estructurar cualquier aplicación como un conjunto de tareas que interactúan entre sí [13].

2.4. Panorama

Los dispositivos de monitoreo y control vistos han demostrado excelentes resultados en el procesamiento de múltiples variables sobre arquitecturas monoprocesador, a pesar de las limitaciones de recursos que estas presentan. No obstante la adquisición de este tipo de sistemas podría resultar frustrante debido a la escasa disponibilidad de los mismos en el mercado local y al engorroso proceso de espera y tramitación al importar desde otro país. Partiendo de la información detallada en este capítulo se propone desarrollar un sistema de monitoreo que integra todas la funcionalidades requeridas para el caso, por solamente una fracción del costo de un dispositivo comercial de características similares.

2.4.1. Sistema propuesto

El sistema está físicamente constituido por tres bloques principales: una interfaz sensorizada, un módulo de adquisición y procesamiento (microcontrolador), y un módulo de visualización.

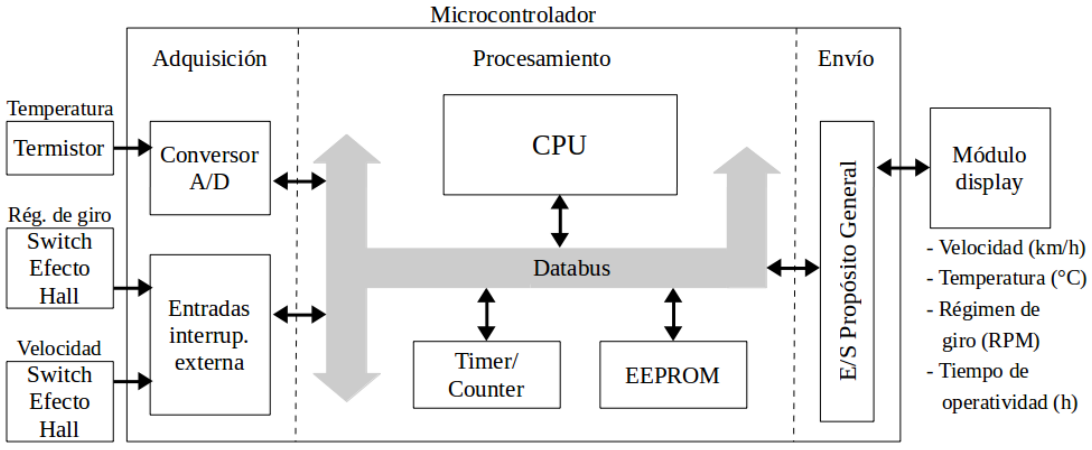


Figura 2.7: Diagrama de bloques del sistema propuesto

En el Capítulo 3 se detalla los criterios de selección del hardware y el funcionamiento de los diversos bloques del sistema.

Capítulo 3

Metodología

En el presente capítulo se describe el proceso de co-diseño de las arquitecturas de hardware y software del sistema desarrollado.

3.1. Descripción del sistema

3.1.1. Especificación de requisitos

El dispositivo debe cumplir con varios requisitos propios de la aplicación, los mismos que deben ser cubiertos tanto a nivel del harware como del software que componen el sistema. Tales requisitos se clasifican en aquellos de tipo funcional (describen que ha de hacer el sistema), y los de tipo no funcional (asociados a otras restricciones a considerar en el diseño). A continuación se describen los requisitos funcionales y no funcionales del sistema desarrollado.

3.1.1.1. Requisitos funcionales

R1. Cada variable (temperatura, régimen de giro, velocidad) debe ser monitoreada a intervalos regulares de tiempo. Esta acción se lleva a cabo mediante la estructuración de las tareas con una temporización adecuada.

R2. La actualización del tiempo operativo del motor debe realizarse cada vez que el sistema detecte un paro con respecto al régimen de giro. El tiempo operativo del motor es acumulativo y debe poder recuperarse tal información tras un corte de energía o un reinicio. Para esta operación se hará uso de un espacio de memoria no volátil.

R3. El sistema debe mostrar en todo momento la información por medio de una pantalla LCD, en la cual puedan apreciarse todas las variables monitoreadas de manera comprensible y cómoda para el usuario.

3.1.1.2. Requisitos no funcionales

R4. Tras la ocurrencia de un fallo por parte de una tarea, el sistema podrá degradar parcialmente su funcionalidad, sin embargo este seguirá en marcha con las tareas que se encuentren disponibles.

R5. Los sensores a implementarse deben poder tolerar las condiciones de trabajo a las que estos serán sometidos. Para ello se han resumido dichas condiciones en la Subsección 3.1.2 para una selección apropiada de los elementos.

R6. El sistema debe distribuise estratégicamente en distintas zonas del ciclomotor con estructuras que permitan fijarlo adecuadamente y lo protejan de agentes externos.

R7. La tasa de actualización de la información en pantalla debe efectuarse de tal forma que resulte cómoda ante la visión del usuario

3.1.2. Características del entorno de trabajo

El sistema se encuentra físicamente distribuido en distintas zonas del ciclomotor, por lo que varios de sus componentes están sometidos a diferentes condiciones de trabajo. La Tabla 3.1 resume los parámetros con los que se ha sido seleccionado cada elemento.

Tabla 3.1: Condiciones del entorno de trabajo

Elemento	Condiciones del entorno	
	Temperatura	Exposición
Sensor de temperatura	+25 a +150 °C	aceite
Sensor de revoluciones	+25 a +80 °C	aire, polvo, salpicaduras
Sensor de velocidad	+25 °C	aire, polvo, salpicaduras
Display	+25 °C	aire, luz solar
Microcontrolador	+25 °C	aire

3.1.3. Requisitos del microcontrolador

Con base en el hardware propuesto de la Fig. 2.7, se muestra un resumen de los requisitos mínimos que debe cumplir el microcontrolador de manera que este se ajuste a las necesidades de la aplicación.

Tabla 3.2: Requisitos del microcontrolador y comparación entre modelos ATmega

Requisitos		Microcontrolador		
Recurso	Cant.	ATmega 328P	ATmega 32U4	ATmega 2560
Módulo ADC	1	1	1	1
Entradas a interrupción externa	2	2	5	8
Timers	1	3	4	6
E/S propósito general	13	23	26	86
EEPROM (bytes)	512	1024	1024	4096
Flash (KB)	20	32	32	256
RAM (KB)	1	2	2,5	8

Por disponibilidad y conveniencia se ha elegido trabajar con la placa Arduino Nano ya que esta incorpora el ATmega 328P que cumple con los requisitos necesarios, como se muestra en

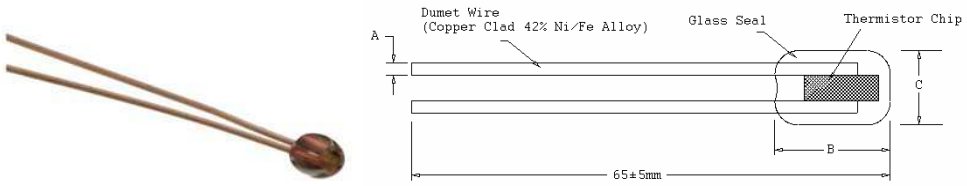
la Tabla 3.2.

3.2. Especificaciones del hardware

3.2.1. Caracterización de los sensores

3.2.1.1. Sensor de temperatura

El G100K4000 es un termistor de coeficiente negativo (NTC) que consta de un chip herméticamente sellado mediante encapsulación de vidrio. Está diseñado para trabajar en ambientes húmedos o con presencia de fluidos, especialmente agua y aceites. Opera en un rango de temperatura que va desde -40 °C hasta +250 °C [19].



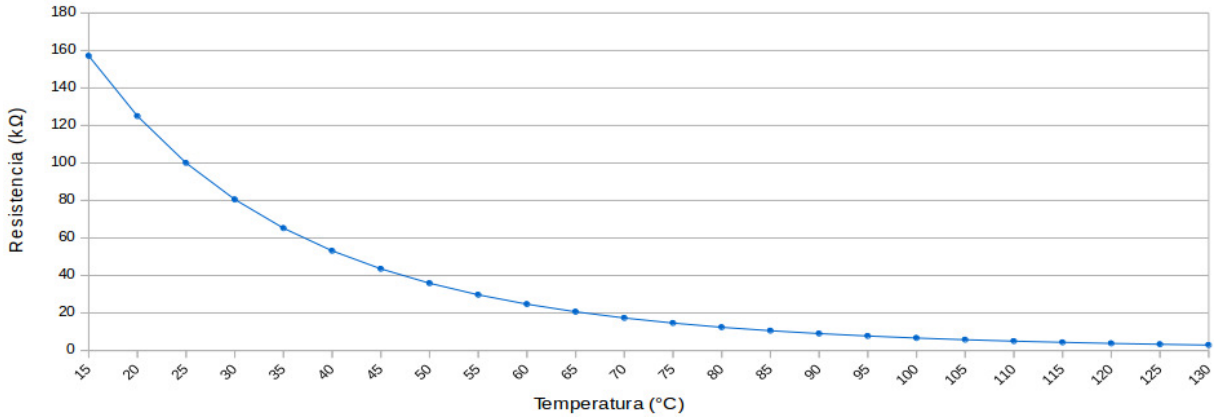
(a) (b)

Figura 3.1: Descripción del sensor G100K4000. (a) Esquema físico, (b) dimensiones.

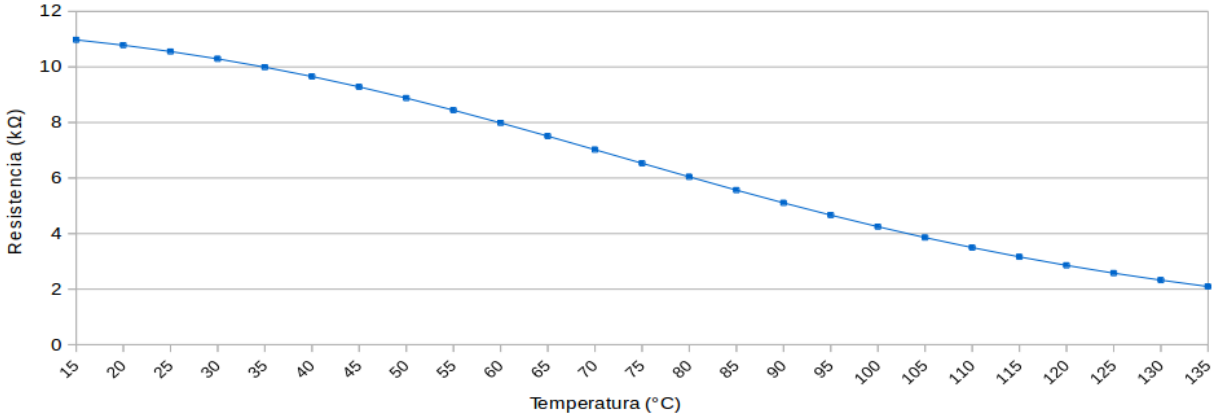
Diagrama Resistencia vs. Temperatura El coeficiente de relación entre resistencia y temperatura se representa a través de un diagrama R_T/T, en donde la curva describe la respuesta resistiva del sensor ante una variación en la temperatura. La Figura 3.2a muestra el diagrama característico del sensor especificado, mientras que la Tabla 3.3 recopila los valores R_T y T en el rango de temperaturas de +15 °C a +130 °C. En [19], se encuentran detallados los valores de resistencia para un rango más amplio de temperaturas.

Tabla 3.3: Valores resistencia-temperatura característicos del sensor G100K4000

$T(^{\circ}C)$	$R_T(k\Omega)$	$T(^{\circ}C)$	$R_T(k\Omega)$
15	157,1	85	10,53
25	100,0	95	7,712
35	65,23	105	5,730
45	43,54	115	4,317
55	29,68	125	3,295
65	20,64	135	2,546
75	14,62		



(a)



(b)

Figura 3.2: Diagramas R_T/T. a) Curva característica del sensor G100K4000, b) Respuesta linealizada del mismo sensor.

Linealización de la señal obtenida Para obtener una aproximación lineal en la salida del sensor, bastará con añadir una resistencia en paralelo a dicho elemento [20]. El cálculo de R se realiza mediante la ecuación (3.1), en donde R_{T1}, R_{T2}, y R_{T3} corresponden al valor de la respuesta resistiva del sensor a tres temperaturas "equidistantes" (T1 = 15 °C, T2 = 75 °C, T3 = 135 °C).

$$R = \frac{R_{T2}(R_{T1} + R_{T3}) - 2R_{T1}R_{T3}}{R_{T1} + R_{T3} - 2R_{T2}} \tag{3.1}$$

Y tras reemplazar con los valores presentados en la Tabla 3.3 se obtiene

$$R = \frac{14,62(157,1 + 2,546) - 2(157,1 \times 2,546)}{157,1 + 2,546 - 2(14,62)} = 11,76k\Omega$$

El valor de R es redondeado a 12 $k\Omega$, resultado con el cual es posible calcular los valores de resistencia R_{eq} y voltaje V_s en la salida. Estos nuevos valores son recopilados en la Tabla (3.4).

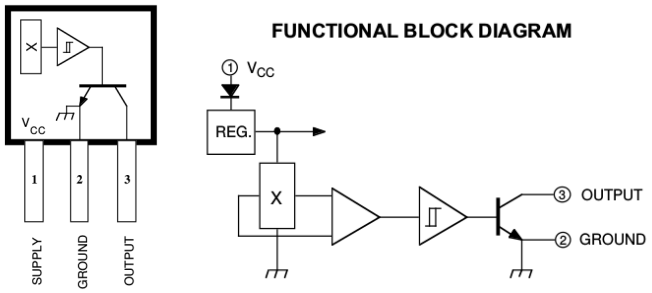
Tabla 3.4: Valores linealizados en la salida.

$T(°C)$	$R_{eq}(k\Omega)$	$V_s(V)$	$T(°C)$	$R_{eq}(k\Omega)$	$V_s(V)$
15	10,98	2,616	85	5,564	1,788
25	10,56	2,567	95	4,664	1,590
35	9,992	2,499	105	3,857	1,392
45	9,284	2,407	115	3,161	1,201
55	8,443	2,289	125	2,576	1,024
65	7,508	2,144	135	2,094	0,866
75	6,530	1,975			

3.2.1.2. Sensor de velocidad angular

El sensor de efecto Hall A3144 es un sensor todo/nada que conmuta en presencia de un campo magnético determinado. Está diseñado para operar en un amplio rango de temperaturas que pueden alcanzar hasta 150 °C. Funciona en modo de conmutación unipolar, permitiendo establecer la polaridad magnética con la que se desea trabajar. Su circuitería interna incorpora

un regulador de voltaje (voltaje de operación desde 4,5 a 24 V), un disparador Schmitt trigger y una salida en colector abierto [21].



(a) (b)

Figura 3.3: Descripción del sensor A3144 [21]. (a) Configuración de sus pines, (b) Diagrama funcional.

Modo de conmutación La salida del dispositivo cambia a estado bajo cuando la intensidad del campo magnético a medir excede un umbral de detección. Si por el contrario el campo magnético se reduce por debajo de este umbral, la salida cambia a estado alto.

Señal obtenida Los cambios de estado generados en la salida del sensor proporcionan una señal de onda cuadrada, cuya frecuencia servirá como parámetro para determinar la velocidad angular en los componentes mecánicos respectivos (motor y rueda).

3.2.2. Adquisición de datos

La información que el sistema debe adquirir de los sensores es obtenida por medio de un conversor analógico a digital y un módulo de ateción a interrupciones externas. El conversor A/D es asignado para el muestreo de la señal del termistor, mientras que el módulo de interrupciones externas a la medición de la frecuencia de los sensores de velocidad angular.

3.2.2.1. Conversor A/D

Es un conversor analógico a digital por aproximaciones sucesivas con resolución de 10 bits y ocho canales (ADC0 - ADC7) conectados a un multiplexor interno. Cuenta con tres referencias de voltaje, una externa asociada a la entrada AREF, una interna de 1,1V, y AVcc que es la entrada de alimentación del convertidor y se conectada directamente al riel de energización (Vcc) del microcontrolador.

Selección del voltaje de referencia y el canal de adquisición Empleando AVcc como voltaje de referencia, el valor obtenido en el convertidor A/D viene determinado por la ecuación (3.2)

$$ADC = \frac{V_s \cdot 1023}{AVcc} \tag{3.2}$$

Siendo V_s el voltaje del sensor de temperatura. Dicho voltaje es adquirido a través del canal 5 (ADC5) del convertidor, ya que los otros canales se encuentran en pines compartidos del microcontrolador que han sido asignados como bits de control del display (ver Subsección 3.2.4).

Muestreo en modo Single Conversion Con la finalidad de generar una acción de muestreo controlado por tareas, resulta necesario controlar la tasa de muestreo del ADC mediante algún mecanismo de disparo. Este conversor puede configurarse en modo Single Conversion que permite tomar una sola muestra ante la activación de un flag en uno se sus registros. Para ello basta con escribir un 1 lógico al bit ADSC (ADC Start Conversion) en el registro ADCSRA. Este bit permanece en estado alto mientras una conversión esté en progreso, y cambia automáticamente a estado bajo cuando la conversión haya terminado. Esta funcionalidad le permite al sistema adquirir muestras de la señal del sensor a intervalos regulares de tiempo según se establezca en la programación.

3.2.2.2. Entradas de atención a interrupciones externas

La utilizacion de entradas a interrupcion externa comprende un método rápido y efectivo para la detetección de eventos asociados a los cambios de estado de una señal digital. El ATmega 328P posee dos de ellas que se encuentran implementadas en los pines INT0 e INT1 y forman parte de los puertos de E/S del microcontrolador, sin embargo pueden ser configurados para asociarlos a cualquiera de los pines PCINT23..0. Por medio de estos pines es posible generar el disparo de una subrutina de interrupción ante la ocurrencia de eventos externos tales como cambios de estado en la señal de sensores.

Medición de ancho de pulso Esta técnica está basada en la detección de flancos que permiten determinar el período y ciclo de trabajo de una señal digital [18]. Bastará entonces con ejecutar una secuencia de instrucciones dentro de la rutina de interrupción: 1) detectar un flanco de bajada en la señal del sensor, 2) guardar instante del primer flanco (inicio del período), 3) detectar un nuevo flaco de bajada consecutivo al anterior, 4) guardar instante del segundo flanco (fin del período).

La tarea de cálculo del tiempo transcurrido entre flancos de bajada se efectúa fuera de la rutina de interrupción, por cuestiones inherentes a la programación de las mismas.

3.2.3. Gestión de recursos de procesamiento

3.2.3.1. Timer/Counter o temporizador

El timer es el componente de hardware encargado de generar una referencia de tiempo válida a la que se denomina tick del sistema. Este proceso interrumpe a la CPU a una razón de tiempo fija, en donde en cada tick, una variable entera que representa el tiempo interno del sistema es incrementada y se reestablece siempre durante la inicialización del dispositivo. El valor que debe establecerse como tick depende de la aplicación en concreto, pudiendo tomar un valor dentro de un rango típico de 1 a 50 milisegundos.

3.2.3.2. CPU

Es la unidad encargada de gestionar los recursos del microcontrolador y dar paso a la ejecución de las tareas tras alcanzar una consigna de tiempo establecida. Para ello la CPU debe acceder a las memorias (memoria de programa, memoria dinámica y memoria no volátil), realizar cálculos, controlar todos sus periféricos y manejar las interrupciones tanto de hardware como de software del sistema.

3.2.3.3. Memoria EEPROM

Es una memoria de almacenamiento no volátil destinada a la retención permanente de información (generalmente asociada al estado de variables o el valor de un contador), la misma que se mantendrá disponible tras un reinicio del dispositivo. Dicho espacio de memoria es accesible por la CPU y puede ser leida o sobrescrita según se disponga en la programación del sistema, sin embargo se debe tener en cuenta que el proceso de escritura se encuentra limitado hasta un cierto número de ciclos. Para este sector de memoria en el ATmega 328P se garantiza su buen funcionamiento hasta 100000 ciclos de escritura.

3.2.4. Envío de resultados por puerto paralelo

Tras la obtención de resultados en la etapa de procesamiento, la información correspondiente al estado del motor es organizada en paquetes de 8 bits y enviada hasta el controlador del módulo de visualización. Este proceso de empaquetamiento y envío se realiza por medio de la librería de la LCD, que ademas se encarga de la configuracion de los puertos C y D del microcontrolador como un puerto paralelo bidireccional y el puerto B como puerto de control.

3.2.4.1. Puertos de E/S

Todos los puertos del microcontrolador pueden utilizarse como entradas o salidas digitales de propósito general, cada una de ellas con resistencias pull-up internas que pueden ser configuradas independientemente. Esta característica hace que cada pin sea capaz de controlar pantallas

LED o LCD de forma directa sin necesidad de hardware adicional. El ATmega 328P posee tres puertos paralelo de E/S designados con las letras B, C, y D. La configuración precargada en la librería del display establece a los bits 4-7 del puerto D y 0-3 del puerto B (PB0, PB1, PB2, PB3, PD4, PD5, PD6, PD7) como el byte de información a transferir, y a los bits 0-4 del puerto C (PC0-PC4) como bits de control.

3.2.5. Descripción del módulo display

El módulo 128H064A consta de una pantalla gráfica LCD monocroma de matriz de pixeles con controlador incorporado. Este módulo se encuentra provisto de 20 pines, de los cuales: 8 conforman el bus de comunicación, 5 son dedicados al control del módulo, 3 para energización, 1 para reinicio y 3 para la regulación del contraste y luz de fondo [22]. En la Tabla 3.5 se encuentra la información general del módulo de visualización.

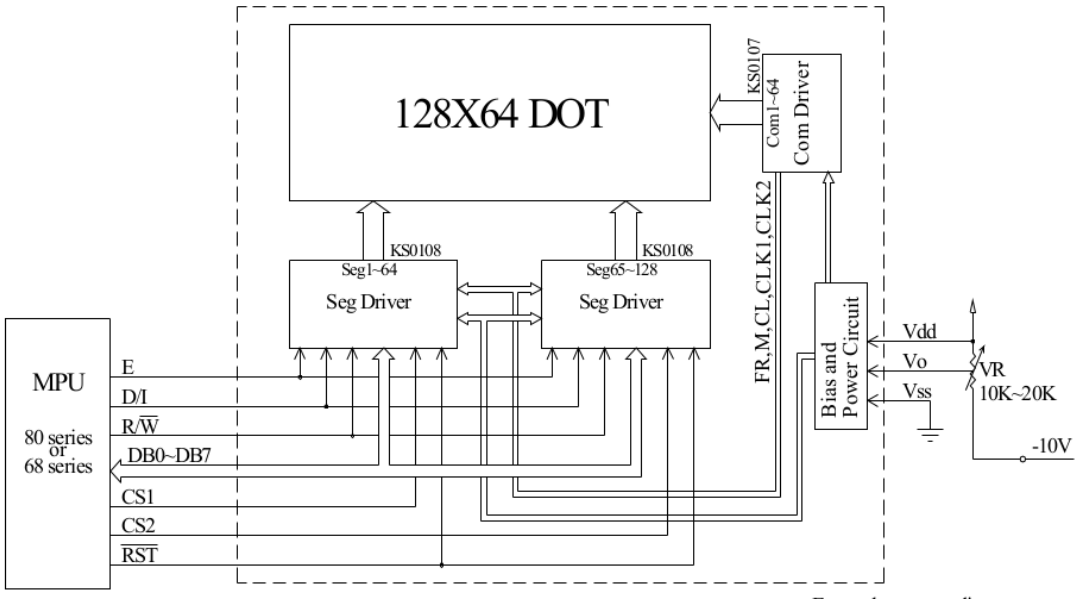


Figura 3.4: Diagrama funcional del módulo GLCD 128H064A

Tabla 3.5: Características generales del módulo GLCD 128H064A

Ítem	Descripción
Voltaje de operación	5 V
Formato	128x64 pixels
Tipo	Gráfico
Controlador	KS 0108
Dimensiones	93x70 mm
Área de visualización	72x40 mm

3.3. Diseño del planificador

La gestión de recursos y ejecución de las tareas es llevada a cabo mediante un planificador cooperativo. La Fig. 3.6 muestra un esquema simplificado del planificador, en donde el proceso principal consta de un bucle infinito en el cual se ejecutan una tras otra las diferentes tareas de acuerdo a un tiempo previamente establecido. El proceso INTx corresponde a las rutinas de interrupción asociadas a los pines INT0 e INT1. Estas rutinas son capaces de interrumpir al sistema con la finalidad de actualizar los instantes de inicio y finalización de la señal de los sensores de efecto Hall.

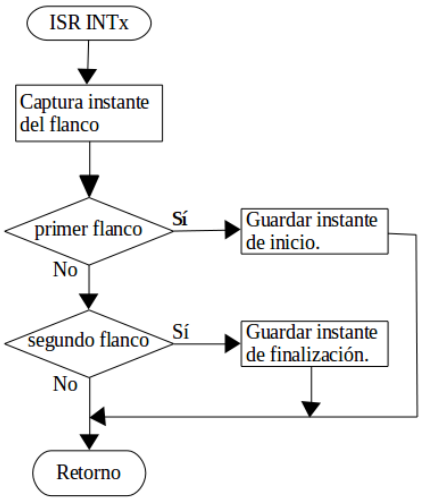


Figura 3.5: Rutina de interrupción para medición del período de las señales.

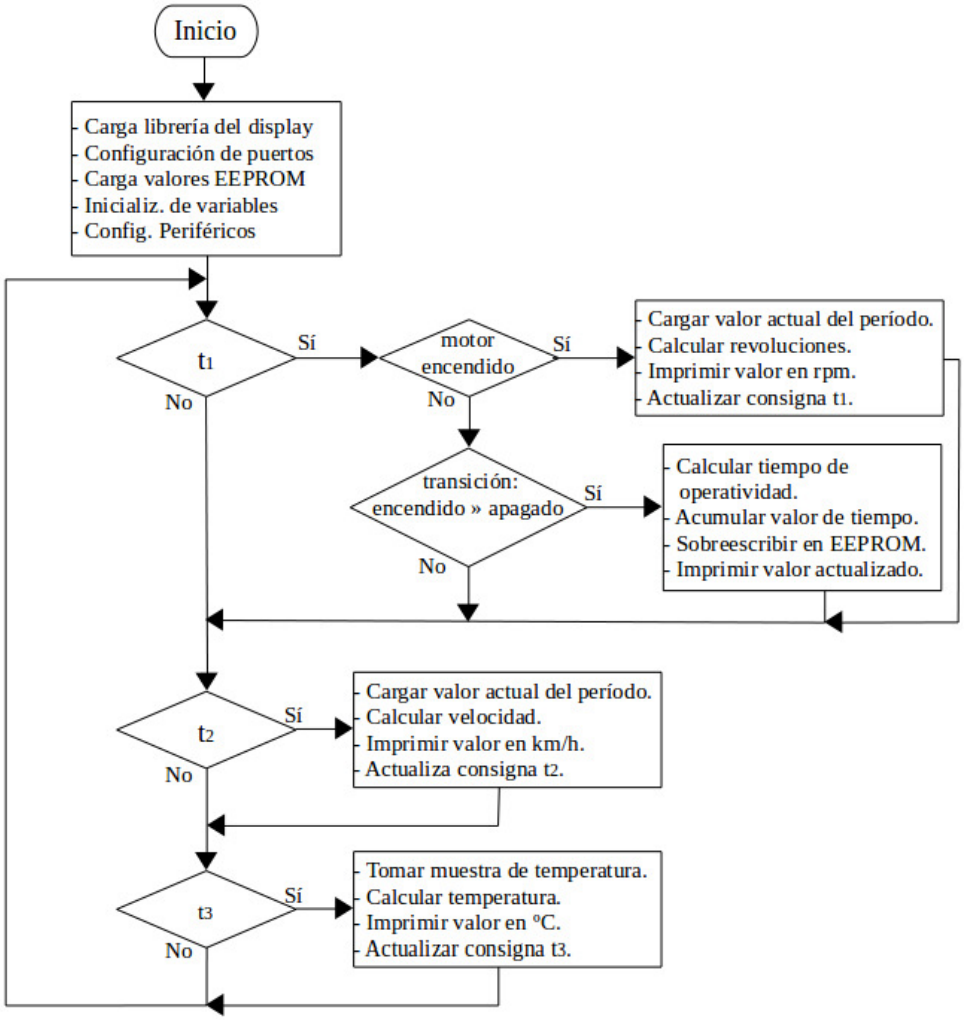


Figura 3.6: Planificación de las tareas mediante estructuración en bucle infinito.

3.3.1. Parámetros de planificación

Cada variable a medir es monitoreada a una razón de tiempo específica, obteniendo diferentes tasas de muestreo y refrescamiento de información en pantalla. Adicionalmente, pruebas preliminares han permitido registrar los tiempos de ejecución para el peor caso obteniéndose así la información de la Tabla 3.6.

Tabla 3.6: Parámetros de planificación de las tareas

τ_i	C_i (ms)	T_i (ms)
τ_1	14	500
τ_2	4	2000
τ_3	12	5000

A partir de los parámetros del sistema se tiene:

- Ciclo mayor o principal: $T_p = mcm(T_i) = mcm(\ 0{,}5,\ 2,\ 5) = 10$ s
- Ciclo menor o secundario: $T_s = MCD(T_i) = MCD(0{,}5,\ 2,\ 5) = 500$ ms

Conociendo esta información es posible describir la evolución de las tareas mediante una línea temporal, tal como se muestra en La Fig. 3.7.

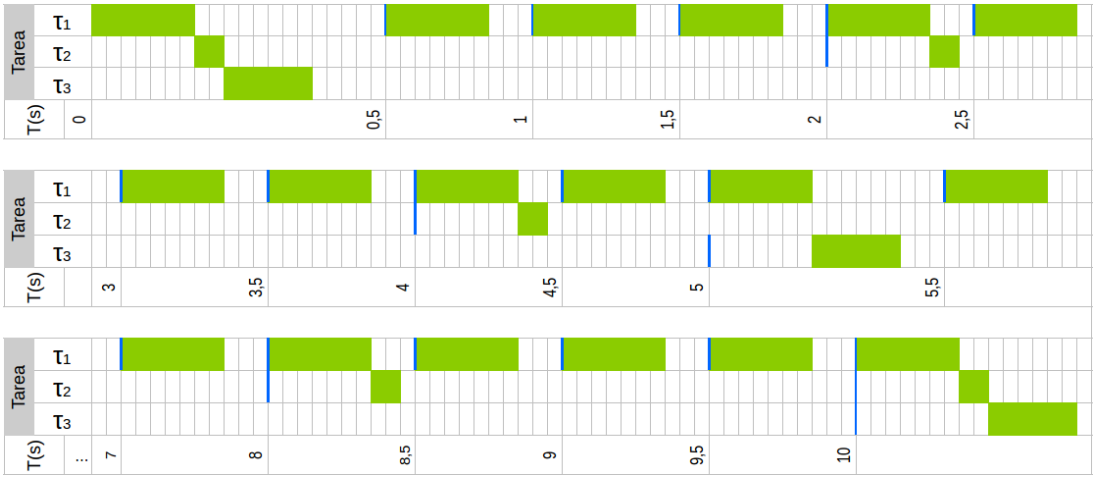


Figura 3.7: Línea temporal del sistema

Como es posible apreciar en la línea temporal del sistema, todas la tareas se ejecutan repetitivamente de acuerdo a su respectivo período. Dicha ejecución se efctúa sin interrupción por parte de las otras tareas hasta que esta finaliza y cede el procesador y los recursos a la siguiente en espera.

Capítulo 4

Implementación y pruebas

En este capítulo se describe la implementación del sistema de monitoreo desarrollado, abordando la metodología mostrada previamente en el Capítulo 3. Además se presentan los resultados obtenidos a partir de las pruebas realizadas en condiciones reales de funcionamiento.

4.1. Implementación sobre el microcontrolador

4.1.1. Base de tiempo del sistema

El temporizador del sistema ha de configurarse adecuadamente para generar una referencia de tiempo válida. Se ha determinado experimentalmente que con una resolución de 5 milisegundos se producen resultados satisfactorios para los fines de esta aplicación.

4.1.1.1. Configuración del timer

Para programar el temporizador con un período de interrupción igual a 5 ms, se ha de determinar previamente la consigna superior de conteo o desbordamiento (que se alojará en el registro OCRx del μC) mediante la ecuación (4.1)

$$OCRx = \frac{f_{CLK}}{N * f_{int}} - 1 = \frac{16 \times 10^6}{256 \times 200} - 1 = 312 \tag{4.1}$$

En donde f_{CLK} es la frecuencia de reloj del sistema, N el factor de preescalado del timer y f_{int} la frecuencia de interrupción requerida ($f_{int} = 1/0,005s$).

El Timer1 del Atmega328P posee una resolución de 16 bits que permtirá almacenar el valor obtenido en (4.1) y ejecutar el incremento respectivo del contador hasta la consigna de desbordamiento calculada de 312.

4.1.1.2. Interrupción en modo comparación

La configuración en modo CTC (Clear Time on Compare Match) utiliza el valor de consigna de desbordamiento en el registro OCRx para manipular la resolución del contador. El valor del contador TCNTx incrementa 1 unidad por cada ciclo de reloj del timer, hasta que se produce una coincidencia por comparación con OCRx, tras lo cual se genera una interrupción, el contador desborda y el proceso se repite nuevamente.

4.1.2. Estimación de las variables medidas

Para poder conocer el estado del motor con respecto a alguna de sus variables medidas, es necesario interpretar de manera adecuada los valores obtenidos en cada sensor. Dichos valores son procesados conjuntamente con un factor de conversión que permite obtener la información requerida por el usuario.

4.1.2.1. Temperatura del motor

Mediante interpolación lineal y empleando los valores de la Tabla 3.4, la ecuación para el cálculo de la temperatura en un instante dado se expresa como

$$T_{oil} = 183,7 - 0,0562V_s \tag{4.2}$$

Siendo T_{oil} la temperatura del aceite en la cámara de lubricación, y V_s el voltaje obtenido en la salida del sensor de temperatura.

4.1.2.2. Régimen de giro del motor

Para la estimación del régimen de giro se debe considerar que el ciclo de trabajo del motor en cuestión es de cuatro tiempos. De esta forma a cada revolución del motor le corresponden dos ciclos de la señal del sensor, o dicho de otra forma, $1rev = 0,5 f_{gm}$, siendo f_{gm} la frecuencia de la señal asociada al giro del motor.

$$R_{giro} = 1\,rev \left(\frac{1}{2} f_{gm}\right) \left(\frac{60\,s}{1\,min}\right) = \left(30 \frac{rev \cdot s}{min}\right) f_{gm} \tag{4.3}$$

4.1.2.3. Velocidad del ciclomotor

De manera similar al procedimiento anterior, el objetivo es transformar el valor de la frecuencia medida correspondiente al giro de la rueda (f_{gr}), en otro cuyas unidades son km/h a través de factores de conversión. Para ello se realizaron mediciones del radio aproximado del neumático trasero del ciclomotor, obteniéndose un valor promedio de 33 cm. Esto permite obtener una relación entre desplazamiento angular y desplazamiento rectilíneo correspondiente a la rueda, siendo $1rev = 2,07m$. Así, la estimación de la velocidad del ciclomotor (v_c) queda determinada por la ecuación (4.4).

$$v_c = f_{gr}(2,07\,m) \left(\frac{1\,km}{1000\,m}\right) \left(\frac{3600\,s}{1\,h}\right) = \left(7,45 \frac{km \cdot s}{h}\right) f_{gr} \tag{4.4}$$

4.1.3. Interfaz de visualización

Para brindar la información pertinente, el sistema posee una interfaz de visualización en la cual puede apreciarse el estado del motor con respecto a las variables monitoreadas. Dicha interfaz se encuentra dividida en zonas específicas que muestran los valores medidos de temperatura, régimen de giro, horas de operatividad del motor desde sus último mantenimiento, velocidad del ciclomotor y estado de la batería. en la Fig. 4.1 se muestra la interfaz desarrollada y seguidamente se describe su funcionamiento.

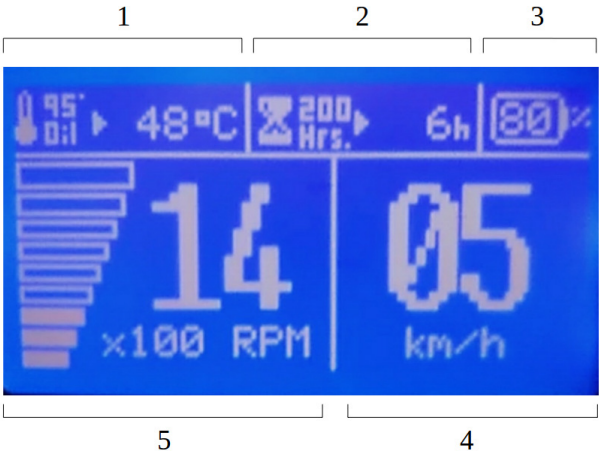


Figura 4.1: Interfaz de visualización del sistema. 1) Indicador de temperatura, 2) indicador de horas, 3) estado de la batería, 4) indicador de velocidad, y 5) indicador de revoluciones.

4.1.3.1. Descripción de la interfaz

La interfaz de visualización se encuentra conformada por los 6 indicadores siguientes:

- Indicador de temperatura. Muestra la temperatura del motor junto a una pequeña etiqueta que advierte la máxima temperatura admisible (95°C).
- Indicador de horas. Muestra el tiempo que el motor ha permanecido encendido desde su último mantenimiento general. La etiqueta “200Hrs” indica que tras acumularse 200 horas en el horómetro, el motor deberá someterse nuevamente a mantenimiento. Las horas se reestablecerán automáticamente luego de alcanzar dicha consigna.
- Estado de la batería. Advierte el nivel porcentual de voltaje suministrado al sistema.
- Indicador de velocidad. Muestra la velocidad a la que se desplaza el ciclomotor.
- Indicador de revoluciones. Muestra el régimen de giro del motor expresado por dos cifras significativas y un factor de multiplicación (x100). Un indicador de barras adicional brinda una apreciación más intuitiva del régimen de giro.

4.2. Acoplamiento del sistema con el ciclomotor

4.2.1. Conexionado del dispositivo

El hardware del sistema se encuentra interconectado a través de una placa electrónica específicamente desarrollada para la aplicación. La placa dispone de los conectores necesarios para el acoplamiento de los sensores y la fuente de alimentación, tal como se describe en la Fig. 4.2. Para información más detallada de las conexiones del sistema, refiérase al diagrama electrónico en el Apéndice .B.

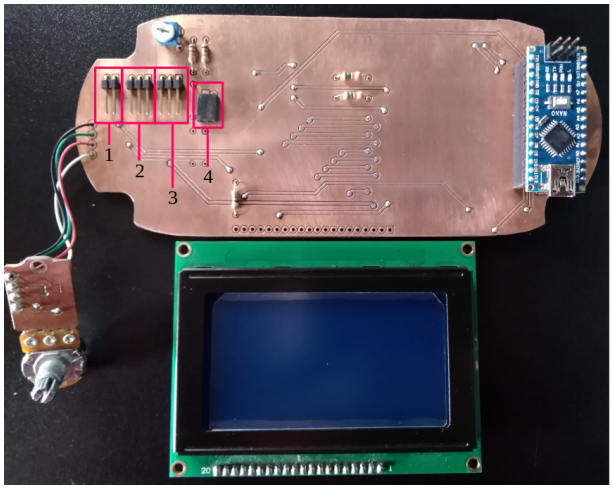


Figura 4.2: Conexiones del dispositivo: 1) entrada del sensor de temperatura, 2) entrada del sensor de revoluciones, 3) entrada del sensor de velocidad, y 4) entrada del voltaje de alimentación.

4.2.2. Estructuras de protección y soporte

Todos los componentes de hardware del sistema deben estar protegidos de agentes externos como agua y polvo, además de cuerpos extraños que puedieran afectar su correcto funcionamiento. Por tal razón se desarrollaron mediante impresión 3D, varias estructuras de protección y soporte para el acoplamiento del sistema con el ciclomotor. En el Apéndice .C se encuentra detallada la lista de materiales de dichas estructuras, así como los planos mecánicos respectivos.

4.2.3. Distribución física de los componentes

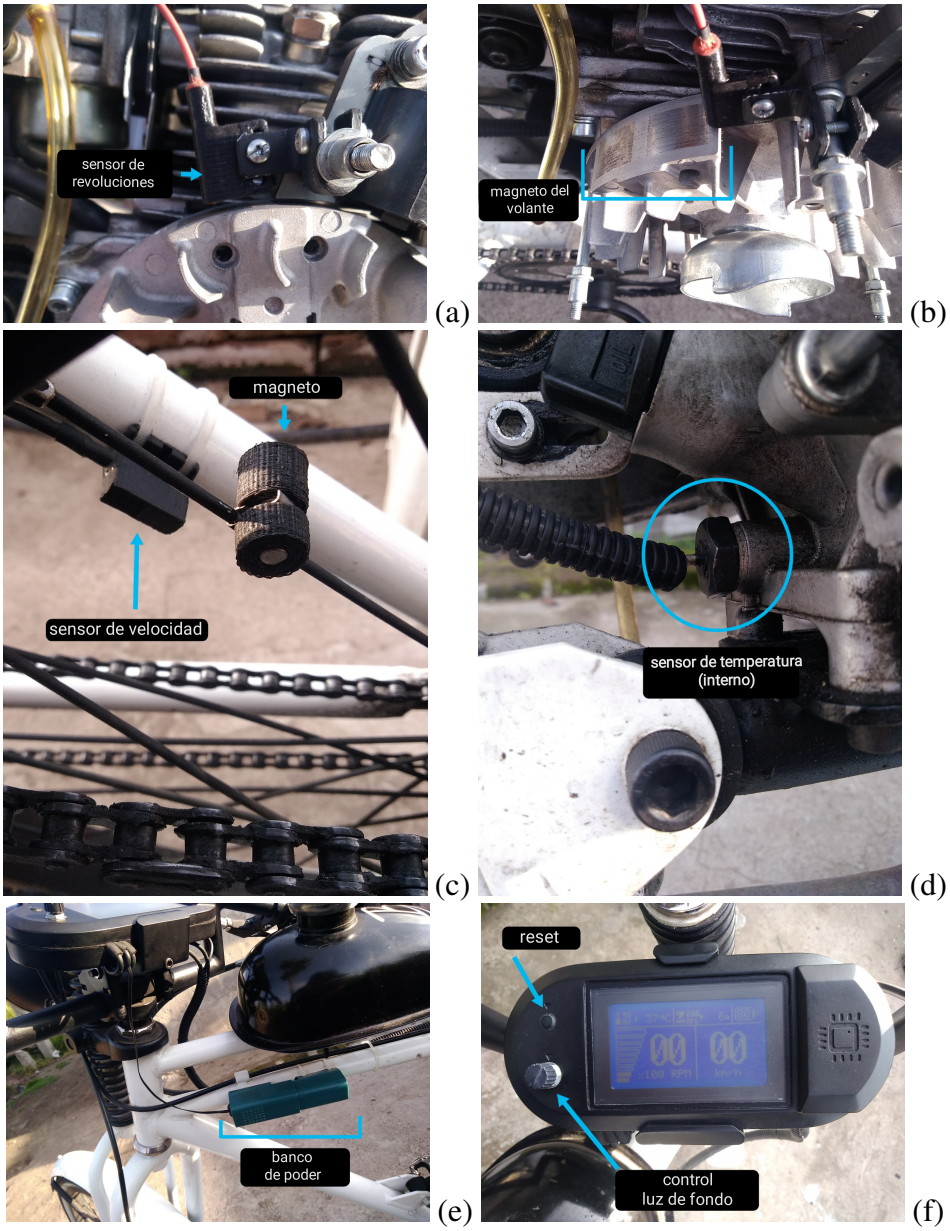


Figura 4.3: Distribución de componentes en el ciclomotor. (a) y (b) Sensor de revoluciones junto al volante de inercia, (c) sensor de velocidad junto al neumático trasero, (d) sensor de temperatura incorporado en la cámara del lubricante, (e) banco de poder bajo el tanque de combustible, (f) módulo de visualización y procesamiento, sobre la dirección.

4.3. Configuración de las pruebas y validación

Con la finalidad de garantizar su correcto funcionamiento, el dispositivo desarrollado ha sido sometido a pruebas de verificación y validación. En la etapa de verificación se realizaron pruebas unitarias en donde cada tarea es probada independientemente, además de las pruebas una vez integrado el sistema en su totalidad. Este procedimiento busca observar y corregir posibles errores durante el proceso de diseño tanto de hardware como de software. Para ello se ha hecho uso de instrumentos de medida dedicados como referencia en la medición de las variables, mientras que para determinar la correctividad temporal se empleó una salida digital del sistema que se activa al inicio y se desactiva al final de una tarea u otro proceso en observación empleando un osciloscopio digital. El tiempo de ejecución de las tareas y el margen de error en las medidas se detalla en la Sección 4.4.

Tabla 4.1: Validación del sistema mediante especificación de requisitos (ver Subsección 3.1.1).

Requisito	Cumple	Resultado
R1	Sí	Cada variable es monitoreada a intervalos regulares de tiempo, siendo estos 500 ms, 2 s, y 5 s.
R2	Sí	El dispositivo es capaz de detectar los instantes de arranque y parada del motor, pudiendo así determinar el tiempo de operatividad entre viajes y el temporizador de para su mantenimiento.
R3	Sí	En la interfaz de visualización es posible observar el estado de las variables de forma cómoda e intuitiva, usando indicadores gráficos y tamaños de fuente fácilmente apreciables desde la posición de conducción del usuario.
R4	Sí	Se garantiza el correcto funcionamiento del dispositivo para la presente aplicación. Ante un fallo por excesiva carga del procesador, únicamente se garantizan los resultados de velocidad y temperatura.
R5	Sí	Los sensores utilizados se ajustan adecuadamente a las condiciones de trabajo de la aplicación
R6	Sí	El sistema ha sido distribuido sobre el ciclomotor con estructuras de protección y soporte.
R7	Sí	La actualización de información en pantalla es adecuada y se realiza a una tasa similar a la de dispositivos comerciales.

Además de las pruebas de verificación, el sistema ha sido sometido a pruebas de validación que respaldan el cumplimiento de las especificaciones para el dispositivo. Una forma de realizar este tipo de pruebas es a partir de la especificación de requisitos detallada en la Subsección 3.1.1, debiendo comprobar que estos se cumplan de acuerdo a lo establecido durante el proceso de diseño. En la Tabla 4.1 se muestran los resultados del proceso de validación del sistema, los mismos que corroboran que el dispositivo desarrollado es adecuado para la aplicación.

4.4. Discusión de resultados

4.4.1. Tiempos de ejecución de las tareas

Para la realización de estas pruebas se consideraron los estados de ralentí y máxima aceleración del motor, los cuales dan lugar a las condiciones de carga computacional nominal y máxima respectivamente. Este comportamiento indica que la carga del procesador se encuentra estrechamente ligada al régimen de giro, debido a que la frecuencia de la señal del sensor de revoluciones varía conforme a la aceleración del motor. De esta forma, una mayor frecuencia en la señal del sensor genera un mayor tasa de interrupciones en el sistema. Los resultados obtenidos de esta prueba se resumen en la Tabla 4.2

Tabla 4.2: Carga computacional del procesador

Tarea	Período	Régimen ≈ 800 rpm		Régimen ≈ 4200 rpm	
		Tiempo decómputo	Utilización	Tiempo decómputo	Utilización
τ_i	T_i (ms)	C_i (ms)	U_i	C_i (ms)	U_i
τ_1	500	8	0,016	14	0,028
τ_2	2000	4	0,002	4	0,002
τ_3	5000	12	0,002	12	0,002
Utilización total del procesador U_p			0,032		0,132

Las pruebas realizadas mostraron que el algoritmo implementado produce resultados satisfactorios. Es decir que ante la mínima y máxima frecuencia de entrada del sistema (30 Hz en ralentí y 140 Hz en aceleración respectivamente) todas las tareas se ejecutan dentro de los

plazos establecidos y los valores medidos son correctos. Tal comportamiento se mantiene hasta frecuencias que bordean los 200 Hz (6000 rpm en el motor). Frecuencias mayores a esta degradan el desempeño del sistema alterando la funcionalidad de tacómetro y horómetro momentaneamente.

4.4.2. Margen de error en las variables medidas

El margen de error en las medidas ha sido cuantificado mediante comparación con otros dispositivos de medición dedicados, siendo estos un termómetro digital, un velocímetro, y un tacómetro. A Partir de este proceso de validadción se obtienen las características funcionales mostradas en la Tabla 4.3.

Tabla 4.3: Características del dispositivo

Variable	Rango	Resolución	Error absoluto	Unidades
Giro del motor	0 - 6000	100	$\pm$ 50	rpm
Velocidad	0 - 99	1	$\pm$ 0,5	km/h
Temperatura	15 - 135	1	$\pm$ 2	°C
Tiempo de operacion	0 - 200	1	$\pm$ 0,1	h

4.4.3. Autonomía

El sistema se energiza a través de un banco de poder con capacidad de 2600 mAh. Para estimar su autonomía se ha llevado un registro de consumo eléctrico global del sistema tomando en cuenta modos de espera y durante la marcha.

Tabla 4.4: Registro de consumo eléctrico del dispositivo

Temperatura (°C)	Consumo (mA)	
	Modo espera	En la marcha
20	34,0	34,3
70	34,1	34,4

Conforme a estos datos se evidencia una variacion de 100 μA entre el modo de espera y

sobre la marcha. Tal incremento se debe a que el ADC es activado únicamente si se ha detectado actividad en el motor. La duración promedio del banco de poder entre cargas ha sido estimada según la ecuación (4.5).

$$T_{Desc.} = \frac{Cap.\ bater\acute{i}a\ [mAh]}{Consumo\ promedio\ [mA]} \tag{4.5}$$

$$T_{Desc.} = \frac{2600}{34,4} = 75,58h$$

El resultado obtenido indica que el banco deberá ser recargado al menos cada 75 horas de uso para prevenir cortes de energía en el dispositivo.

4.4.4. Disponibilidad del hardware en el mercado local

Para garantizar el fácil desarrollo del sistema propuesto se requiere de una buena disponibilidad de los componentes de hardware a utilizar. La Tabla 4.5 muestra en detalle el precio y cantidad del material utilizado, el mismo adquirido sin dificultad en el mercado local.

Tabla 4.5: Lista del material electrónico utilizado

Elementos	Precio	Cant.	Total
Hall Effect switch A3144	$1,70	2	$3,40
Glass bead sensor G100K4000	$3,00	1	$3,00
Arduino Nano R3 (compatible)	$10,00	1	$10,00
Pantalla gráfica LCD 128x64	$12,00	1	$12,00
Banco de poder 2600 mAh	$7,00	1	$7,00
Capacitores	$0,15	2	$0,30
Resistores	$0,10	5	$0,50
Potenciómetro 10k	$0,50	2	$1,00
Pulsador digital	$0,20	1	$0,20
Consumibles y otros	-	-	$8,10
TOTAL			$45,50

Capítulo 5

Conclusiones, recomendacones y trabajo futuro

5.1. Conclusiones

Se desarrolló un sistema de monitoreo en tiempo real utilizando una plataforma de hardware abierto. Dicho sistema permite cuantificar y mostrar al usuario el estado de un ciclomotor con respecto a sus variables de funcionamiento, siendo estas su velocidad de desplazamiento, régimen de giro y temperatura del motor. Adicionalmente el sistema cuenta con un horómetro que indica el tiempo de operatividad del motor a partir de el último mantenimiento general realizado sobre el mismo.

Se estableció una estrategia de monitoreo que consiste en mantener diferentes tiempos de observación sobre cada variable de funcionamiento del ciclomotor. Dicha estrategia consistió en estructurar todos los procesos de monitorización como un sistema multitarea mediante un planificador cooperativo que permite gestionar el acceso de estos procesos a la CPU y a los ditintos recursos del microcontrolador.

Se utilizó la plataforma de hardware abierto Arduino junto a su propio entorno de desarrollo

Arduino IDE, más específicamente el modelo Nano R3 que se ajusta a las necesidades de la aplicación. Esta tarjeta de desarrollo ha mostrado excelentes resultados debido a su versatilidad y a los recursos presentes en el microcontrolador que esta incorpora. Tanto la tarjeta de desarrollo como los componentes de hardware utilizados para la elaboración del dispositivo pueden ser fácilmente adquiridos y se encuentran cotizados a la fecha a un precio de 45,50 dólares americanos en el mercado local.

El dispositivo elaborado fue acoplado a un ciclomotor y puesto a prueba en condiciones reales de funcionamiento. El desempeño del sistema durante estas pruebas mostró resultados satisfactorios, cumpliendo todos los requisitos establecidos en la etapa de diseño.

5.2. Recomendaciones

A pesar que el dispositivo demostró excelentes resultados sobre el ciclomotor empleado en las pruebas, se recomienda utilizarlo únicamente en ciclomotores cuyo régimen de giro del motor sea menor a 6000 rpm. Un régimen de giro mayor a este altera momentaneamente las funcionalidades de tacómetro y horómetro del sistema.

Para el manejo de un mayor número de interrupciones que deba soportar el sistema, es altamente recomendable implementar un algoritmo de planificación distinto ya que un planificador cooperativo presenta inconvenientes si la cantidad de interrupciones que este debe gestionar es elevada.

Al ser una plataforma de hardware abierto Arduino cuenta con varias versiones clonadas que poseen las mismas prestaciones que el modelo original a un precio reducido. Se recomienda evitar las versiones genéricas con el chip CH340, ya que este requiere la instalación de un controlador adicional en el ordenador para la comunicación con la tarjeta Arduino.

Durante las pruebas de funcionamiento el dispositivo evidenció un acoplamiento físicamente firme con ciclomotor, sin embargo su montaje puede llevar varios minutos dependiendo de la experiencia del usuario. Se recomienda mejorar el modelo y las dimensiones de las estructuras de protección y soporte de este dispositivo para reducir el tiempo que toma la instalación del mismo.

5.3. Trabajo futuro

El desarrollo de este sistema de monitoreo multivariable servirá como un precedente para la creación de nuevos sistemas embebidos con características similares en los cuales se requiera una arquitectura monoprocesador y dispositivos de bajo costo, volviéndose un trabajo fácilmente replicable y escalable.

Además del punto expuesto anteriormente, las variables medidas por el dispositivo desarrollado podrían ser registradas y almacenadas en un espacio de la memoria EEPROM del microcontrolador con la finalidad de crear un registro del funcionamiento y estado del ciclomotor que permitirían un análisis más minucioso sobre el rendimiento y el ciclo de vida de este tipo de vehículo.

Bibliografía

[1] T. Bartlett, *Motorized Bicycles: From Motorbikes to Mopeds to Ebikes*, Lekington, KY: Lulu.com, 2010.

[2] "Digital Pacing Indicator for Motorcycles", por K. Mackenroth. (1987, Sept. 22). *Patent 4695958* [En línea]. Disponible: https://patentimages.storage.googleapis.com/33/7b/65/021b850b7b597a/US4695958.pdf

[3] "Microprocessor-Controlled Speedometer/Odometer", por P. Liu. (1998, Feb. 3). *Patent 5714929* [En línea]. Disponible: https://patentimages.storage.googleapis.com/4f/b0/d7/94c03e2449e3a9/US5714929.pdf

[4] "Motorcycle Engine Management System", por J. D. Meaney. (1992, Dic. 29). *Patent 5174263*. Disponible: https://patentimages.storage.googleapis.com/4c/1b/14/474b4f616e56c1/US5174263.pdf

[5] "Multi-Function Display Meter System for a Motorcycle", por J. W. Hugget. (2002, Junio 18). *Patent US 6407663B1* [En línea]. Disponible: https://patentimages.storage.googleapis.com/05/a3/23/4eaf8af2b950d8/US6407663.pdf

[6] "Odometer System and Method for a Vehicle", por Y. Nakasawa *et al*. (2007, Mayo 1). *Patent US 7212107B2* [En línea]. Disponible: https://patentimages.storage.googleapis.com/be/60/f9/29777c66710017/US7212107.pdf

[7] N. H. Norma *et al*, “Software Design and Development for Automotive Online Monitoring System”, en *2011 IEEE 7th International Colloquium on Signal Processing and its Applications*, pp. 489-494.

[8] A. Bingamil, I. Alsyouf, A. Cheaitou, “Condition Monitoring Technologies, Parameters and Data Processing Techniques for Fault Detection in Internal Combustion Engines”, en *2017 International Conference on Electrical and Computing Technologies and Applications*, Sharjah, United Arab Emirates, 2017.

[9] A. Nuria Oliva, *Redes de Comunicación Industriales*, Madrid: Editorial UNED, 2013.

[10] L. M. Jiménez García, R. Puerto Manchón, *Sistemas Informáticos en Tiempo Real: Teoría y Aplicaciones*, Universidad Miguel Hernández, 2017.

[11] G. C. Buttazzo, *Hard Real Time Computing Systems*, Nueva York: Springer, 2011.

[12] T. P. Baker, A. Shaw, “The Cyclic Executive Model and Ada”, 1988

[13] A. Hangan, G. Sebestyen, “Cyclic Executive-based Method for Scheduling Hard Real-Time Transactions on Distributed Systems”, en *2011 IEEE 7th International Conference on Intelligent Computer Communication and Processing*, pp. 441-444.

[14] A. Burns, N. Hayes, M.F. Richardson, “Generating Feasible Cyclic Schedules”, en *Control Eng. Practice*, Vol. 3, No. 2, pp. 151-162.

[15] F.J. Aliaga García, I.M. Aliaga García, “Planificadores STR”, *Simulador de Planificadores de Sistemas en Tiempo Real*, http://www.uco.es/ el2pamuj/simuladorSTR/algoritmos.htm#

[16] A. Díaz Estrella *et al*, *Teoría y Diseño con Microcontroladores de Freescale*, España: McGraw Hill, 2008.

[17] G. Rivas, “Arquitectura de software ejecutivo en tiempo real multitareapara sistemas embebidos basada en máquinas de estados finitos”, en *1 st LACCEI International Symposium on Software Architecture and Patterns*, ciudad de Panamá, 2012.

[18] T.L. Floyd, *Fundamentos de Sistemas Digitales*, 9na ed. Madrid: Prentice Hall, 2006.

[19] G100K4000 Radial Glass Thermistor, Measurement Specialties, Malibú, CA, 2008.

[20] R. Pallás Areny,*Sensores y Acondicionadores de Señal*, 4ta ed. Barcelona: Marcombo, 2005.

[21] Sensitive Hall Effect Switches for High-Temperature Operations, Allegro Microsystems., Worcester, MA, 2005.

[22] 128 x 64 Graphic LCD, Vishay Intertechnologies. Shelton, CT, 2017

Apéndice

.A. Firmware

.A.1. Codigo implementado sobre microcontrolador

Programa 1: Código implementado en Arduino

```
#include <glcd.h>
#include <fonts/allFonts.h>
#include <EEPROM.h>

//  Timer interrupt configuration
unsigned int T_INTERRUPT = 0x138;    //0x7A11 -> 500 ms
                                     //0x1860 -> 100 ms
                                     //0x270  -> 10 ms
                                     //0x138  -> 5 ms

// Variables for external interrupts
volatile int COUNTER;
volatile unsigned long I_TIMES;
volatile boolean FIRST_INT0;
volatile boolean FIRST_INT1;
volatile boolean TRIGGD_INT0;
volatile boolean TRIGGD_INT1;
volatile unsigned long START_TIME_INT0;
volatile unsigned long START_TIME_INT1;
volatile unsigned long FINISH_TIME_INT0;
volatile unsigned long FINISH_TIME_INT1;

// Variables for timing
int TIME1;
int TIME2;
int TIME3;
int LAST_TIME1;
int LAST_TIME2;
int LAST_TIME3;

// ADC Variables
int VALIDATE;
int BATT;
```

```
void setup()
{
  // Configure content to be displayed
  GLCD.Init();
  SetInterface();
  // Disable interrupts
  SREG = (SREG & 0b01111111);
  // Clear ADC multiplexer register
  ADMUX = 0;
  // AVcc(~5V) reference voltage
  ADMUX |= (1 << REFS0);
  // set ADC prescaler to 128
  ADCSRA |= (1 << ADPS2)|(1 << ADPS1)|(1 << ADPS0);
  // Reset Timer 1
  TCCR1A = 0;
  //-set prescaler to 256 and
  //-select Clear Timer on Compare match (CTC) mode
  TCCR1B = 0b00001100;
  // set compare match at 624, since compare match register
  // CM = (16*10^6) / (100*256) - 1 = 624
  OCR1AH = T_INTERRUPT >> 8;
  OCR1AL = T_INTERRUPT;
  // zero 16-bit Timer/Counter1
  TCNT1 = 0;
  // enable interrupt for Output Compare Register A
  TIMSK1 = 0b00000010;
  // Enable interrupts
  SREG = 0b11111111;
  // Set up for interrupts
  prepareInterrupt0 ();
  prepareInterrupt1 ();
  // Start monitorizing
  MonitorBegin();
}

// Define text areas
gText rpmNum = gText(31, 22, 2, 1, fixednums15x31);
gText rpmTex = gText(19, 54, 8, 1, SystemFont5x7);
gText spdNum = gText(84, 22, 2, 1, fixednums15x31);
gText spdTex = gText(87, 54, 4, 1, SystemFont5x7);
gText Batt = gText(108, 4, 2, 1, SystemFont5x7);
gText C = gText(45, 5, 1, 1, SystemFont5x7);
gText Temp = gText(22, 5, 3, 1, SystemFont5x7);
```

```
gText OpTimeH = gText(80, 5, 3, 1, SystemFont5x7);
gText tripN = gText(78, 54, 3, 1, SystemFont5x7, INVERTED);

void MonitorBegin()
{
  // Loads device initial display state
  rpmNum.print("00");
  rpmTex.print("x100 RPM");
  spdNum.print("00");
  spdTex.print("km/h");
  Temp.Printf("   ");
  GLCD.DrawRect(41, 6, 2, 2);
  C.print("C");

  boolean MOTOR_START = false;
  boolean BIKE_STOP = false;
  boolean ADC_ENABLE = false;
  unsigned long TRIP_TIME;
  unsigned long TIME_M = EEPROM.read(0);
  unsigned long TIME_H = EEPROM.read(1);
  OpTimeH.Printf("%3d", TIME_H);
  unsigned long INIT;

  while (1)
  {
    // For each external interrupt, the system asks if a second state change
    // has ocurred which corresponds to the signal's one cycle completion.
//------------------------ For INT0: ------------------------
    if (TRIGGD_INT0)
    {
      // Time required to refresh RPMs text area
      TIME1 = I_TIMES - LAST_TIME1;

      // Each 500 ms
      if (TIME1 >= 100)
      {
        // Period and frequency of the signal:
        unsigned long elapsedTime0 = FINISH_TIME_INT0 - START_TIME_INT0;

        // Each tick takes 16 us
        // RPM conversion factor = 0.3
        int rpm = (1.0 / (((elapsedTime0) * 16e-6)) * 0.3);
        if (rpm < 8)
```

```
{
  rpm = 0;
}
rpmNum.Printf("%02d", rpm);

// Make a correspodence between RPM number and RPM indicator
if (rpm < 33)
{
  if (rpm <= 30)
  {
    if (rpm <= 27)
    {
      if (rpm <= 24)
      {
        if (rpm <= 21)
        {
          if (rpm <= 18)
          {
            if (rpm <= 15)
            {
              if (rpm <= 12)
              {
                if (rpm <= 9)
                {
                  GLCD.FillRect(1, 61, 8, 1);
                  GLCD.FillRect(1, 56, 10, 1, WHITE);
                  GLCD.FillRect(1, 51, 12, 1, WHITE);
                  GLCD.FillRect(1, 46, 14, 1, WHITE);
                  GLCD.FillRect(1, 41, 16, 1, WHITE);
                  GLCD.FillRect(1, 36, 18, 1, WHITE);
                  GLCD.FillRect(1, 31, 20, 1, WHITE);
                  GLCD.FillRect(1, 25, 22, 2, WHITE);
                  GLCD.FillRect(1, 18, 24, 3, WHITE);
                }
                else
                {
                  GLCD.FillRect(1, 61, 8, 1);
                  GLCD.FillRect(1, 56, 10, 1);
                  GLCD.FillRect(1, 51, 12, 1, WHITE);
                  GLCD.FillRect(1, 46, 14, 1, WHITE);
                  GLCD.FillRect(1, 41, 16, 1, WHITE);
                  GLCD.FillRect(1, 36, 18, 1, WHITE);
                  GLCD.FillRect(1, 31, 20, 1, WHITE);
```

```
            GLCD.FillRect(1, 25, 22, 2, WHITE);
            GLCD.FillRect(1, 18, 24, 3, WHITE);
          }
        }
        else
        {
          GLCD.FillRect(1, 61, 8, 1);
          GLCD.FillRect(1, 56, 10, 1);
          GLCD.FillRect(1, 51, 12, 1);
          GLCD.FillRect(1, 46, 14, 1, WHITE);
          GLCD.FillRect(1, 41, 16, 1, WHITE);
          GLCD.FillRect(1, 36, 18, 1, WHITE);
          GLCD.FillRect(1, 31, 20, 1, WHITE);
          GLCD.FillRect(1, 25, 22, 2, WHITE);
          GLCD.FillRect(1, 18, 24, 3, WHITE);
        }
      }
      else
      {
        GLCD.FillRect(1, 61, 8, 1);
        GLCD.FillRect(1, 56, 10, 1);
        GLCD.FillRect(1, 51, 12, 1);
        GLCD.FillRect(1, 46, 14, 1);
        GLCD.FillRect(1, 41, 16, 1, WHITE);
        GLCD.FillRect(1, 36, 18, 1, WHITE);
        GLCD.FillRect(1, 31, 20, 1, WHITE);
        GLCD.FillRect(1, 25, 22, 2, WHITE);
        GLCD.FillRect(1, 18, 24, 3, WHITE);
      }
    }
    else
    {
      GLCD.FillRect(1, 61, 8, 1);
      GLCD.FillRect(1, 56, 10, 1);
      GLCD.FillRect(1, 51, 12, 1);
      GLCD.FillRect(1, 46, 14, 1);
      GLCD.FillRect(1, 41, 16, 1);
      GLCD.FillRect(1, 36, 18, 1, WHITE);
      GLCD.FillRect(1, 31, 20, 1, WHITE);
      GLCD.FillRect(1, 25, 22, 2, WHITE);
      GLCD.FillRect(1, 18, 24, 3, WHITE);
    }
  }
```

```
        else
        {
          GLCD.FillRect(1, 61, 8, 1);
          GLCD.FillRect(1, 56, 10, 1);
          GLCD.FillRect(1, 51, 12, 1);
          GLCD.FillRect(1, 46, 14, 1);
          GLCD.FillRect(1, 41, 16, 1);
          GLCD.FillRect(1, 36, 18, 1);
          GLCD.FillRect(1, 31, 20, 1, WHITE);
          GLCD.FillRect(1, 25, 22, 2, WHITE);
          GLCD.FillRect(1, 18, 24, 3, WHITE);
        }
      }
      else
      {
        GLCD.FillRect(1, 61, 8, 1);
        GLCD.FillRect(1, 56, 10, 1);
        GLCD.FillRect(1, 51, 12, 1);
        GLCD.FillRect(1, 46, 14, 1);
        GLCD.FillRect(1, 41, 16, 1);
        GLCD.FillRect(1, 36, 18, 1);
        GLCD.FillRect(1, 31, 20, 1);
        GLCD.FillRect(1, 25, 22, 2, WHITE);
        GLCD.FillRect(1, 18, 24, 3, WHITE);
      }
    }
    else
    {
      GLCD.FillRect(1, 61, 8, 1);
      GLCD.FillRect(1, 56, 10, 1);
      GLCD.FillRect(1, 51, 12, 1);
      GLCD.FillRect(1, 46, 14, 1);
      GLCD.FillRect(1, 41, 16, 1);
      GLCD.FillRect(1, 36, 18, 1);
      GLCD.FillRect(1, 31, 20, 1);
      GLCD.FillRect(1, 25, 22, 2);
      GLCD.FillRect(1, 18, 24, 3, WHITE);
    }
  }
  else
  {
    GLCD.FillRect(1, 61, 8, 1);
    GLCD.FillRect(1, 56, 10, 1);
```

```
          GLCD.FillRect(1, 51, 12, 1);
          GLCD.FillRect(1, 46, 14, 1);
          GLCD.FillRect(1, 41, 16, 1);
          GLCD.FillRect(1, 36, 18, 1);
          GLCD.FillRect(1, 31, 20, 1);
          GLCD.FillRect(1, 25, 22, 2);
          GLCD.FillRect(1, 18, 24, 3);
        }
      }
      LAST_TIME1 += 100;

      if (rpm > 8)
      {
        while (!MOTOR_START)
        {
          INIT = I_TIMES;
          MOTOR_START = true;
          ADC_ENABLE = true;
        }
      }
    }
  }
  else
    {
      if (MOTOR_START)
      {
        unsigned long TIME5 = (I_TIMES * 313) + COUNTER;
        if ((TIME5 - START_TIME_INT0) > 31250)
        {
          if ((TIME5 - FINISH_TIME_INT0) > 31250)
          {
            rpmNum.print("00");
            TRIP_TIME = (I_TIMES - INIT) / 6000;
            TIME_M = TIME_M + TRIP_TIME;

            if (TIME_M >= 60)
            {
              TIME_H = TIME_H + (TIME_M / 60);
              TIME_M = TIME_M % 60;
              if (TIME_H >= 200)
              {
                TIME_H = 0;
              }
```

```
            EEPROM.write(1, TIME_H);
          }
          OpTimeH.Printf("%3d", TIME_H);
          EEPROM.write(0, TIME_M);
          MOTOR_START = false;
        }
      }
    }
  }

  // Time required to refresh Temperature text area
  TIME2 = I_TIMES - LAST_TIME2;

  // Each 5 s
  if (ADC_ENABLE && (TIME2 >= 1000))
  {
    long result = readVcc();
    ADMUX = (1 << REFS0)|(0 << MUX3)|(1 << MUX2)|(0 << MUX1)|(1 << MUX0);
    ADCSRA |= (1 << ADSC);
    while (bit_is_set(ADCSRA,ADSC));

    long adc_b1 = ADCL;
    adc_b1 |= ADCH<<8;

    long Vin = adc_b1*result/1023;
    int temp;

    if (Vin > 2289)
    {
      temp = 335 - 0.1223 * Vin;
    }
    else
    {
      temp = 183.7 - 0.0562 * Vin;
    }
    Temp.Printf("%3d", temp);
    LAST_TIME2 += 1000;
  }
//------------------------- For INT1: -------------------------
  if (TRIGGD_INT1)
  {
    unsigned long elapsedTime1 = FINISH_TIME_INT1 - START_TIME_INT1;
    int spd = (1.0 / (elapsedTime1 * 16e-6)) * 7.45;
```

```
  // Speed conversion factor = 7.45
  spdNum.Printf("%02d", spd);
  if (BIKE_STOP)
    BIKE_STOP = false;
}
else
{
  if (!BIKE_STOP)
  {
    unsigned long TIME4 = (I_TIMES * 313) + COUNTER;
    if ((TIME4 - START_TIME_INT1) > 465625)
    {
      if ((TIME4 - FINISH_TIME_INT1) > 465625)
      {
        spdNum.print("00");
        BIKE_STOP = true;
      }
    }
  }
}

if (TRIGGD_INT0)
  prepareInterrupt0 ();

if (TRIGGD_INT1)
  prepareInterrupt1 ();

TIME3 = I_TIMES - LAST_TIME3;

  // Each 2 seconds
  if (TIME3 >= 400)
  {
    long result = readVcc();
    if (result > 5165)
    {
      BATT = 99;
    }
    else
    {
      BATT = 80;
    }
    Batt.Printf("%02d", BATT);
```

```
        LAST_TIME3 += 400;

      }
  }
}

void isr0 ()
{
  unsigned int COUNTER = TCNT1;  // Quickly save it

  // Wait until we noticed second one
  if (TRIGGD_INT0)
    return;

  // Detects when the first falling edge has ocurred
  if (FIRST_INT0)
  {
    START_TIME_INT0 = (I_TIMES * 313) + COUNTER;
    FIRST_INT0 = false;
    return;
  }
  FINISH_TIME_INT0 = (I_TIMES * 313) + COUNTER;
  TRIGGD_INT0 = true;
  detachInterrupt(0);
}

void isr1 ()
{
  unsigned int COUNTER = TCNT1;

  if (TRIGGD_INT1)
    return;

  if (FIRST_INT1)
  {
    START_TIME_INT1 = (I_TIMES * 313) + COUNTER;
    FIRST_INT1 = false;
    return;
  }
  FINISH_TIME_INT1 = (I_TIMES * 313) + COUNTER;
  TRIGGD_INT1 = true;
  detachInterrupt(1);
}
```

```
// Timer compare match takes place every 5 ms
ISR(TIMER1_COMPA_vect)
{
  I_TIMES++;
}

void prepareInterrupt0()
{
  // Get ready for next time
  EIFR = _BV(INTF0);  // Clear flag for interrupt 0
  FIRST_INT0 = true;
  TRIGGD_INT0 = false;  // Re-arm for next time
  attachInterrupt(0, isr0, FALLING);
}

void prepareInterrupt1()
{
  // Get ready for next time
  EIFR = _BV(INTF1);  // Clear flag for interrupt 1
  FIRST_INT1 = true;
  TRIGGD_INT1 = false;
  attachInterrupt(1, isr1, FALLING);
}

long readVcc() {
  long result;
  // Read 1.1V reference against AVcc
  // ADC channel 5 selection
  ADMUX = _BV(REFS0) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1);
  do
  {
    VALIDATE++;
    ADCSRA |= _BV(ADSC); // Convert
    while (bit_is_set(ADCSRA,ADSC));
    result = ADCL;
    result |= ADCH<<8;
    // Back-calculate AVcc in mV
    result = 1151870L / result;  //(5.03*229/1023)*1023*1000 Calib. at ~1.13V
    } while (VALIDATE < 8);
    VALIDATE = 0;
    return result;
}
```

```
void SetInterface()
{
//————————RPM indicator————————————————————
  // RPM Levels
  GLCD.DrawRect(0, 60, 10, 3);
  GLCD.DrawRect(0, 55, 12, 3);
  GLCD.DrawRect(0, 50, 14, 3);
  GLCD.DrawRect(0, 45, 16, 3);
  GLCD.DrawRect(0, 40, 18, 3);
  GLCD.DrawRect(0, 35, 20, 3);
  GLCD.DrawRect(0, 30, 22, 3);
  GLCD.DrawRect(0, 24, 24, 4);
  GLCD.DrawRect(0, 17, 26, 5);

  // Delimiters
  GLCD.DrawHLine(0, 15, 127);
  GLCD.DrawVLine(71, 16, 47);
  GLCD.DrawVLine(52, 0, 14);
  GLCD.DrawVLine(103, 0, 14);

//————————Temperature indicator————————————————
  // Thermometer
  GLCD.FillCircle(2, 10, 2);
  GLCD.DrawVLine(1, 2, 5);
  GLCD.DrawVLine(3, 2, 5);
  GLCD.DrawVLine(2, 7, 1);
  GLCD.SetDot(2, 1, BLACK);
  // –9–
  GLCD.FillRect(7, 2, 2, 4);
  GLCD.SetDot(8, 3, WHITE);
  GLCD.DrawRect(7, 5, 1, 1, WHITE);
  // –5–
  GLCD.FillRect(11, 2, 2, 4);
  GLCD.DrawHLine(12, 3, 1, WHITE);
  GLCD.DrawHLine(11, 5, 1, WHITE);
  // –degree–
  GLCD.SetDot(15, 2, BLACK);
  // –O–
  GLCD.DrawRect(7, 8, 2, 4);
  // –i–
  GLCD.DrawVLine(11, 11, 1);
  GLCD.SetDot(11, 9, BLACK);
```

```
  //  -l-
  GLCD.DrawVLine(13, 8, 4);
  //  Arrow
  GLCD.DrawVLine(18, 6, 4);
  GLCD.DrawVLine(19, 7, 2);
  GLCD.SetDot(20, 8, BLACK);

//------------------Battery indicator----------------------
  GLCD.DrawRoundRect(106, 2, 15, 10, 2);
  GLCD.DrawVLine(122, 5, 4);
  GLCD.DrawLine(127, 5, 124, 8);
  GLCD.SetDot(127, 8, BLACK);
  GLCD.SetDot(124, 5, BLACK);

//------------------Hour-meter indicator-------------------
  //  Sand clock
  GLCD.FillRect(55, 3, 6, 2);
  GLCD.DrawVLine(56, 4, 1, WHITE);
  GLCD.DrawHLine(58, 6, 1);
  GLCD.SetDot(58, 7, BLACK);
  GLCD.DrawLine(56, 6, 57, 7);
  GLCD.DrawLine(59, 7, 60, 6);
  GLCD.DrawLine(55, 10, 57, 8);
  GLCD.DrawLine(59, 8, 61, 10);
  GLCD.FillRect(55, 11, 6, 1);
  GLCD.SetDot(58, 10, BLACK);

  //  -2-
  GLCD.FillRect(64, 3, 2, 4);
  GLCD.DrawHLine(64, 4, 1, WHITE);
  GLCD.DrawHLine(65, 6, 1, WHITE);
  //  -0-
  GLCD.DrawRect(68, 3, 2, 4);
  //  -0-
  GLCD.DrawRect(72, 3, 2, 4);
  //  -H-
  GLCD.FillRect(64, 9, 2, 4);
  GLCD.DrawVLine(65, 9, 1, WHITE);
  GLCD.DrawVLine(65, 12, 1, WHITE);
  //  -r-
  GLCD.DrawVLine(68, 10, 3);
  GLCD.SetDot(69, 10, BLACK);
  //  -s-
```

```
  GLCD.FillRect(71, 10, 1, 3);
  GLCD.SetDot(72, 11, WHITE);
  GLCD.SetDot(71, 12, WHITE);
  //  -.-
  GLCD.SetDot(74, 13, BLACK);
  //  Arrow
  GLCD.DrawVLine(76, 6, 4);
  GLCD.DrawVLine(77, 7, 2);
  GLCD.SetDot(78, 8, BLACK);
  //  -h-
  GLCD.DrawVLine(98, 7, 4);
  GLCD.SetDot(99, 9, BLACK);
  GLCD.DrawVLine(100, 9, 2);
}

void loop(){}
```

.B. Esquema electrónico

.C. Planos mecánicos