FACULTAD DE INGENIERIA

DEPARTAMENTO DE ELECTRONICA

Proyecto de fin de carrera:

Implementación de lógica de control de UEC en motores de combustión Interna mediante una CIAA.

Alumno: Federico Marcelo Carnevale

Director:
Dra. Luciana De Micco
Co-Director:
Dr. Martin Caldera

II

Agradecimientos

Quiero agradecer a los profesores de la UNMDP, la Dra. Luciana De Micco del departamento de Electrónica y al Dr. Martin Caldera del departamento de Mecánica por el continuo asesoramiento y apoyo durante el desarrollo de todo el trabajo. También quiero hacer especial agradecimiento y mención al señor Luciano Di Tonto por su gran participación y ayuda durante los ensayos en el LABORATORIO DE MÁQUINAS TÉRMICAS de la UNMDP. También quiero hacer más que especial agradecimiento, enorme agradecimiento a mi toda mi familia por su apoyo incondicional durante el transcurso de toda la carrera, que permitieron y lograron que nunca baje los brazos y siga hasta llegar al momento de escribir esto, que permitieron que llegue este momento.

Resumen

El presente trabajo describe el desarrollo y la implementación de una unidad electrónica de control (UEC) orientada a la inyección de combustible para un motor de combustión interna de ignición por chispa. El objetivo es desarrollar una UEC nacional de programación abierta, para controlar el caudal de combustible y así lograr una relación estequiométrica entre el aire y el combustible de modo de obtener consumo óptimo y también disminuir las emisiones contaminantes de los motores. El proyecto se centra en el análisis del comportamiento de un motor de combustión interna desde el punto de vista del control de dosado de combustible y el desarrollo de la lógica del sistema utilizando la Computadora Industrial Abierta Argentina (CIAA) programada en OSEK RTOS, el sistema operativo que se utiliza en la industria automotriz.

El trabajo fue desarrollado en base a un único modelo de motor comercial en donde se hicieron las pruebas y los análisis necesarios para llegar a su fin.

Como banco de prueba se utilizo un motor de cuatro tiempos, marca **Renault** modelo clio de 1.149 litros 74 kW.

Índice general

Lista de simbolos y abreviaturas

Simbolo	Descripción	Unidades
P o P_{adm}	Presión de múltiple de admisión	Kpa
P_{esc}	Presión de múltiple de escape	Kpa
T_{adm}	Temperatura de aire de admisión	$°C$
T_{esc}	Temperatura de gases de escape	$°C$
t	Tiempo	S
λ	Factor lambda	
AFR	Relación aire combustible	
AFR_e	Relación aire combustible esperada	
m_a	Masa de aire	g
m_c	Masa de combustible	g
N	Torque	Nm
V	Cilindrada	Litros
R	Constante de gases	$\frac{Kj}{Kg}$
G_i	Ganancia de inyector	$\frac{g}{seg}$
t_i	Tiempo de inyección total	ms
t_{iff}	Tiempo de inyección feedforward	ms
t_{ifb}	Tiempo de inyección feedback	ms
n	Régimen del motor	RPM
η_{vol}	Rendimiento volumétrico	

Abreviatura	Termino
CIAA	Computadora industrial abierta argentina
GPIO	General Purpose Input/Output
HMI	Human Machine Interface
MCI	Motor de combustión Interna
OSEK	Traducción:Sistemas abiertos y sus interfaces para la electrónica en automóviles
PMI	Punto muerto inferior
PMS	Punto muerto superior
UEC	Unidad electrónica de control

Capítulo 1

Introducción

1.1. Origen del problema

La inyección de combustible es un sistema de alimentación de motores de combustión interna (MCI), que se utiliza para reemplazar al antiguo carburador. Hoy en día, está presente en prácticamente todos los automóviles debido a la obligación de reducir las emisiones contaminantes de escape y además para que sea eficiente el uso del catalizador de gases a través de un ajuste óptimo del factor lambda.

La UEC comanda desde el sistema de alimentación de combustible y formación de la mezcla hasta el sistema de encendido de chispa en los motores Otto, que es el que se encarga de desencadenar la combustión de la mezcla aire/combustible.

La inyección posee una aplicación de mando electrónico, por medio de un calculador o UEC, que utiliza la información de diversos sensores colocados sobre el motor para manejar las distintas fases de funcionamiento, siempre obedeciendo las solicitudes del conductor en primer lugar y las normas de anticontaminación en un segundo lugar. Se consigue una mejor dosificación del combustible.

1.2. Objetivos

1.2.1. Objetivo general

Desarrollar e implementar la lógica de una UEC de inyección de combustible nacional, de bajo costo, basada en una placa CIAA. La misma será de programación abierta, permitiendo así que los usuarios intervengan en las acciones de gestión que la misma realice bajo ciertas circunstancias de funcionamiento del motor.

1.2.2. Objetivos específicos

- Programar la CIAA mediante OSEK RTOS, para implementar un sistema de inyección y control de **A/C**. Adquirir los conocimientos necesarios para la programación de la CIAA. Capacitación en programación OSEK:
  - Entorno de desarrollo IDE Eclipse.
  - Introducción a OSEK.
  - Utilización de servicios POSIX.
- Diseño y programación del corazón de la UEC.
- Puesta en servicio. Ensayos de laboratorio.
- Conclusiones

1.3. Dificultades

Un MCI (motor de combustión interna) tiene características desde el punto de vista de un sistema dinámico, las cuales plantean una serie de problemas para el diseño del controlador de la mezcla **A/C**. Estas dificultades son enumeradas a continuación.

1. **Planta compleja**
   - El fenómeno de la formación de la mezcla de aire y combustible es muy complejo, gobernado por numerosas leyes físicas en las que intervienen muchas variables.
   - El orden del sistema es desconocido.
   - La dinámica depende del punto de funcionamiento del motor, representado por el grado de carga y régimen de giro. Pueden variar muy rápidamente en lo que se refiere a la carga de motor, y algo menos en régimen, por la propia inercia mecánica del conjunto motor más coche.
   - El propio ciclo termodinámico en el que se basa el funcionamiento de un MCI implica una discontinuidad en los procesos.
2. **Presencia de importantes retardos en el lazo de realimentación**
   - **Retardo por régimen:** Desde el momento en el que se inyecta el combustible, hay un retardo propio que depende de régimen de giro hasta que los gases de esa inyección son expulsados del cilindro al colector de admisión.

- **Retardo por transporte de gases:** Luego de la expulsión de los gases residuales del cilindro, es necesario que el colector de escape sea llenado por completo y así obtener información sobre el estado de la mezcla, para eso es necesario tener en cuenta el tiempo de transporte desde la valvula de escape hasta la ubicación del sensor de gases.

3. **Dinámica del combustible:** El combustible pulverizado por el inyector no va a parar directamente al cilindro, sino que siempre hay una porción de combustible líquido depositado sobre el colector de admisión, a modo de depósito. Parte de este líquido vuelve a evaporarse y entra al cilindro junto con la fracción de vapor , o bien lo hace directamente, pero con un retardo mayor.

1.4. Organización de la tesis

El presente trabajo contiene un total de 7 capítulos. En el Capítulo 2 se da una explicación detallada del funcionamiento de un motor de ciclo Otto, los ciclos térmicos en los que el combustible se transforma en trabajo mecánico, y por último se nombran los sensores y actuadores que intervienen en su funcionamiento. El Capítulo 3 trata sobre los modelos matemáticos que describen el comportamiento de las variables mecánicas, como el aire y el torque. En el Capítulo 4 se ahonda sobre los temas de control de combustión y desde los parámetros de entrada hasta los ciclos de realimentación. En el Capítulo 5 se da una introducción del funcionamiento de la CIAA tanto en software como en hardware. En el Capítulo 6 se hace la unión entre el motor, el hardware y el software propuestos. El Capítulo 7 muestra el comportamiento del control bajo distintos parámetros.

Capítulo 2

Funcionamiento de motor

2.1. Introducción

En este capítulo se hace un recorrido sobre el funcionamiento de un motor de combustión interna de encendido a chispa, comenzando por los ciclos térmicos y terminado por los sensores que intervienen en la gestión.

2.2. Funcionamiento de motor de ciclo Otto

El ciclo Otto es el ciclo termodinámico que se aplica en los motores de combustión interna de encendido provocado (motores de gasolina). Existen en funcionamiento dos tipos de motores que se rigen por el ciclo de Otto, por un lado están los motores de dos tiempos que tienen la característica en la que una vuelta de motor es igual a un ciclo completo, y por otro lado están los motores de cuatro tiempos los cuales dos vueltas de motor es equivalente a un ciclo de motor. [01, 2016].

La figura (2.1) muestra un diagrama simplificado de MCI, en el cual se muestra la entrada de aire, en la parte superior de la imagen se encuentra una válvula llamada **válvula mariposa** que cumple la función de limitar el aire que ingresa al múltiple de admisión y luego al cilindro.
El aire es mezclado con el combustible por medio de la inyección electrónica monopunto (un inyector alimenta todos cilindros) o multipunto (cada cilindro tiene su propio inyector), luego la mezcla es encendida por medio de una chispa de bujía y comienza un ciclo termodinámico. Por último, los gases de la combustión pasan por un convertidor catalítico o catalizador para el control y reducción de los gases nocivos expulsados al medio ambiente.

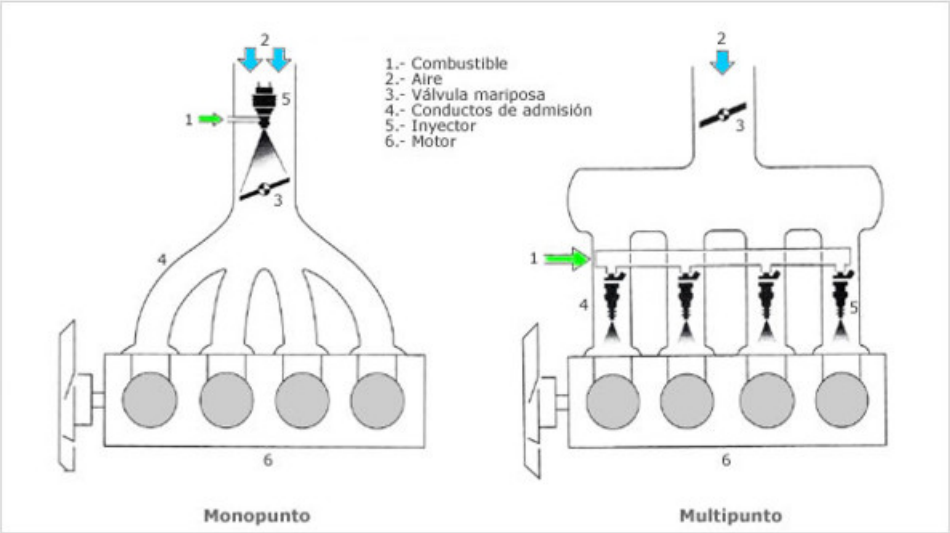

Figura 2.1: Diagrama motor con inyección monopunto-multipunto.

2.2.1. Ciclos termodinámicas

El ciclo Otto consta de seis procesos, dos de los cuales (E-A y A-E) no participan en el ciclo termodinámico del fluido operante pero son fundamentales para la renovación de la carga del mismo. La figura (2.2)muestra la evolución del ciclo termodinámico:

1. E-A: Admisión a presión constante (renovación de la carga).

2. A-B: Compresión de los gases e isoentrópica.

3. B-C: Combustión, aporte de calor a volumen constante. La presión se eleva rápidamente antes de comenzar el tiempo útil.

4. C-D: Fuerza, expansión isoentrópica o parte del ciclo que entrega trabajo.

5. D-A: Escape, cesión del calor residual al ambiente a volumen constante.

6. A-E: Escape, vaciado de la cámara a presión constante (renovación de la carga.)(isobárico).

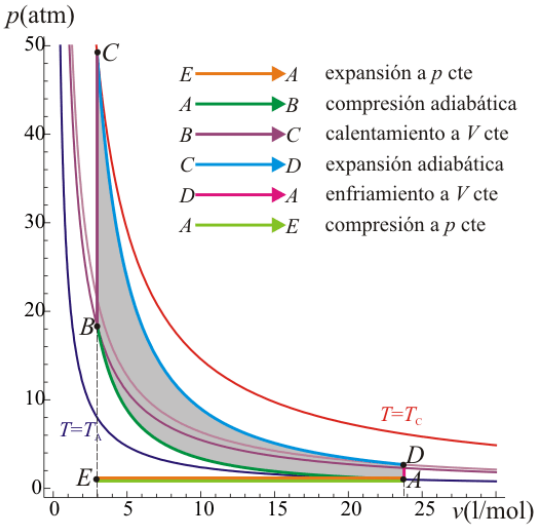

Figura 2.2: Diagrama P vs V de ciclo Otto.

2.2.2. Ciclo 4 tiempos

1. Durante la primera fase, el pistón se desplaza hasta el PMI (Punto Muerto Inferior) y la válvula de admisión permanece abierta, permitiendo que se aspire la mezcla de combustible y aire hacia dentro del cilindro (esto no significa que entre de forma gaseosa).

2. Durante la segunda fase las válvulas permanecen cerradas y el pistón se mueve hacia el PMS (Punto Muerto Superior), comprimiendo la mezcla de aire y combustible. Cuando el pistón llega al final de esta fase, una chispa en la bujía enciende la mezcla.

3. Durante la tercera fase, se produce la combustión de la mezcla, liberando energía que provoca la expansión de los gases y el movimiento del pistón hacia el PMI. Se produce la transformación de la energía química contenida en el combustible en energía mecánica trasmitida al pistón, que se trasmite a la biela, y la biela la trasmite al cigüeñal, de donde se toma para su utilización.

4. En la cuarta fase se abre la válvula de escape y el pistón se mueve hacia el PMS, expulsando los gases producidos durante la combustión y quedando preparado para empezar un nuevo ciclo (renovación de la carga).

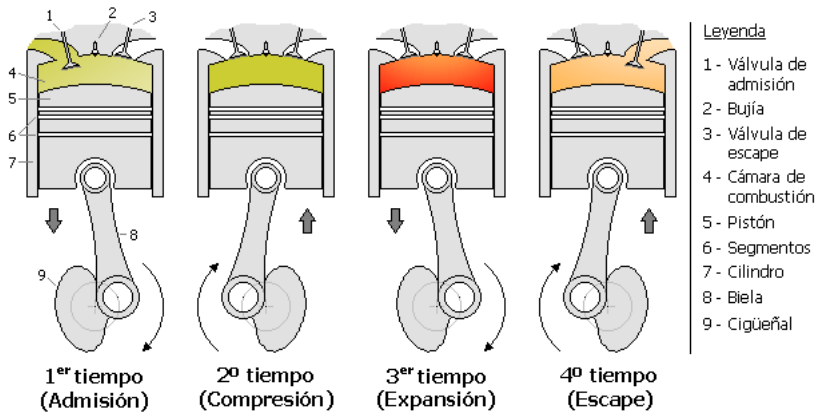

Figura 2.3: Diagrama ciclo 4 tiempos.

Normalmente, los motores constan de más de un cilindro. En el caso más habitual, que es el de cuatro cilindros en linea, las bielas van unidas a un cigüeñal común, de forma que los ciclos de trabajo están desfasados $180°$ (un ciclo es de $720°$ o de dos vueltas de cigüeñal) y se realiza en el orden 1-3-4-2, con el objetivo de conseguir ofrecer un par lo más regular posible, y un dinámica de cigüeñal equilibrada. La combustión en un motor es una reacción exotérmica, en la que parte de ese calor de reacción se transforma en trabajo mecánico. Para ello es necesario lograr una mezcla de un combustible (la gasolina, formada por un conjunto de hidrocarburos) y un carburante (el oxigeno, contenido en el aire atmosférico) de tal calidad que proporcione una combustión correcta. Independientemente de la tecnología utilizada y del objetivo, esta mezcla ha de cumplir una serie de requerimientos:

- Que sea **combustible**, por lo cual la mezcla debe contener todos los reactivos.
- Que sea **gaseosa en el momento del encendido**, por lo que el combustible debe estar perfectamente vaporizado dentro de la cámara de combustión.
- Que sea **homogénea**. Esto implica que la mezcla que llega a la cámara de combustión debe tener iguales características en todos los puntos del espacio. Como consecuencia, debe repartirse por igual entre todos los cilindros.
- Que sea correctamente **dosificada**. En este punto es donde entra en tema el control de inyección por medio de la UEC.

A lo largo del trabajo se da por supuesto las tres primeras condiciones y queda por entender e implementar el término “correctamente dosificada”, esto se va a tratar a continuación.

Se entiende por **dosado** o **relación aire-combustible** (air–fuel ratio: **AFR**) al cociente entre la masa de aire y la masa de combustible existente en la mezcla. En los combustibles comerciales utilizados en automoción, el valor para el cual la relación de combustión es teóricamente completa sin que sobre ninguno de los reactivos, se sitúa entre 14 y 15. Esto es lo que se denomina dosado **estequiométrico**. Si se normaliza el dosado respecto al dosado estequiométrico dividiendolo por la relación deseada ($AFR_e = 14{,}7$), se obtiene la variable denominada con la letra griega **lambda** (λ). En relación al valor estequimétrico, se define como **mezcla rica** a aquella que tiene un exceso de combustible ($\lambda < 1$), mientras que una **mezcla pobre** tiene exceso de aire, o lo que es lo mismo, una falta de combustible ($\lambda > 1$)

$$AFR = \frac{m_a}{m_c} \tag{2.1}$$

$$\lambda = \frac{AFR}{AFR_e} = \frac{1}{AFR_e} \times \frac{m_a}{m_c} \tag{2.2}$$

El dosado requerido por un motor dependerá del objetivo buscado, siempre y cuando se sitúe dentro de los límites de la mezcla $\lambda \cong [0,5 \div 1,5]$.

El dosado de la mezcla tiene gran influencia en las emisiones contaminantes, la combustión completa de hidrocarburo debería proporcionar dióxido de carbono (CO_2) y agua (H_2O). Lo que sucede es que, o bien la combustión no es completa, o bien resulta que las altas presiones y temperaturas reinantes en la cámara de combustión hacen que estos productos reaccionen entre sí, dando lugar a otros productos contaminantes resultante de estas reacciones secundarias. El contenido total de contaminantes en los gases resultantes es aproximadamente el $1\,\%$ del volumen de los gases totales emitidos, y en un motor Otto, se agrupan en tres categorías:

- **Monóxido de carbono (CO):** Aparece como consecuencia de una oxidación parcial. Es altamente tóxico, porque impide la aportación de oxigeno por la sangre a los tejidos del cuerpo.

- **Hidrocarburos (HC):**Proceden de la propia gasolina y del aceite sin quemar o quemado parcialmente. Son muy irritantes para los seres vivos.

- **Óxidos de nitrógeno ($\mathbf{NO_X}$):** Consiste en una mezcla de diferentes óxidos de nitrógeno, producida por la oxidación del nitrógeno atmosférico como consecuencia de las altas temperaturas y presiones existentes en la cámara de combustión, lo cual favorece ese tipo de reacción.

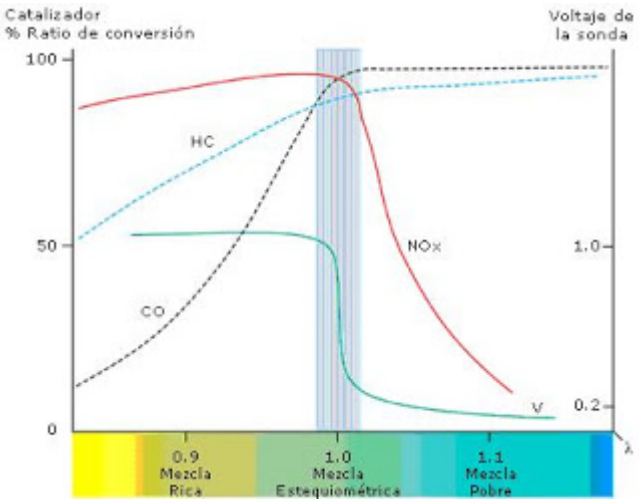

Figura 2.4: Eficiencia de conversión del catalizador en función de λ

En la figura (2.4) se muestra la eficiencia de conversión de cada uno de los gases (NO_x,HC y CO) dentro del catalizador de escape en función al factor λ y se observa que el punto de mayor eficiencia se da cuando $\lambda = 1$. Si se trabaja con $\lambda < 1$ la eficiencia de conversión de los Hidrocarburos y del monóxido de carbono bajan considerablemente. Por otro lado, si se trabaja con $\lambda > 1$ la eficiencia de conversión de los óxidos de nitrógeno cae bruscamente.
La **eficiencia de conversión** de un contaminante se define como la diferencia entre las concentraciones de entrada y de salida del catalizador, respecto a la concentración de entrada al mismo.
La tolerancia en el valor de λ que se admite para el rendimiento mínimo de conversión del 80 % entre los tres contaminantes simultáneamente se denomina **ventana**, y es un valor muy pequeño, centrado justamente en el valor estequiométrico. El tamaño de la ventana suele ser del orden del $\pm 0,3\,\%$, aunque varía según el tipo de catalizador y su estado. El desgaste por uso y envejecimiento del catalizador hace que la ventana sea mas estrecha que la de un catalizador nuevo.
El catalizador tiene partículas de **Ce**, las cuales le dan la capacidad de almacenar oxígeno químicamente en forma de $\mathbf{CeO_2}$. Esto significa que ligeras excursiones fuera de la ventana no sólo no son perjudiciales, sino incluso beneficiosas, siempre y cuando estas sean del lado rico y pobre alternativamente [Falk and Mooney, 1980] y [Katashiba et al., 1991]. Cuando la mezcla rica llega al catalizador, se libera el oxígeno que previamente, por efecto de la mezcla pobre, se había almacenado, el cual se utiliza para la reacción de oxidación.

2.3. Gestión de motor

La UEC requiere de señales de entrada y actuadores para poder gestionar y calcular los valores de salida a controlar. En la figura(2.5) se representa un diagrama en donde se listan todas las señales que necesita una UEC como entrada, y todos los actuadores que controla.

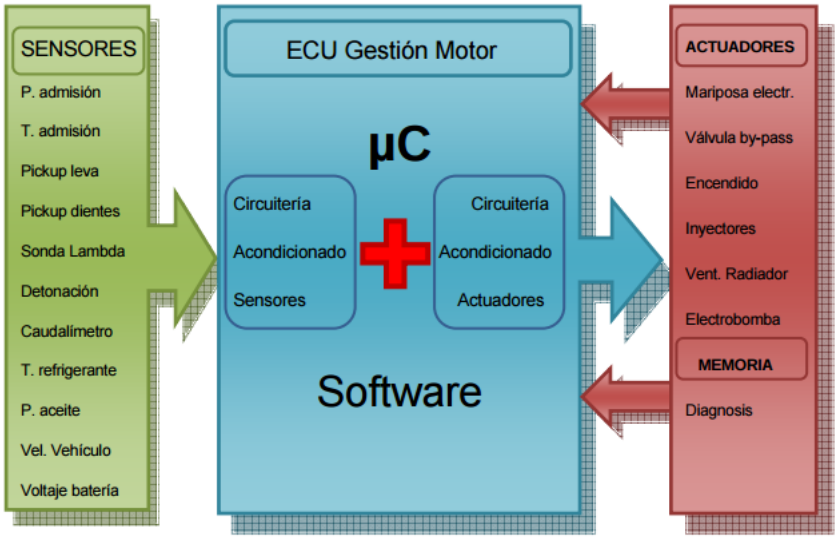

Figura 2.5: Arquitectura generica de UEC para gentión de motor.

2.3.1. Sensores

Sensor pickup dientes

También llamado sensor **CKP**, se utiliza para medir la velocidad de giro de motor (RPM) y también la posición del cigüeñal para así poder sincronizar la inyección y el encendido en la correcta posición de pistón. Existen dos tipos de sensores, por un lado está el sensor inductivo que es de imán permanente, por lo que no necesita alimentación. Por otro lado está el sensor de reluctancia variable, el cual necesita alimentación. En el presente trabajo se utiliza el sensor del primer tipo.
El sensor inductivo está colocado en la rueda fónica solidaria al volante del motor y el cable de salida de la señal esta apantallado para evitar interferencias electromagnéticas. La rueda fónica tiene $\mathbf{60 - 2 = 58}$ dientes. El hueco por dientes faltantes es especialmente grande y su función es de referencia y está asignado a

una posición definida del cigüeñal. Sirve para la sincronización de la unidad de control, en la figura (2.6) se muestra la disposición del sensor sobre la rueda fónica.

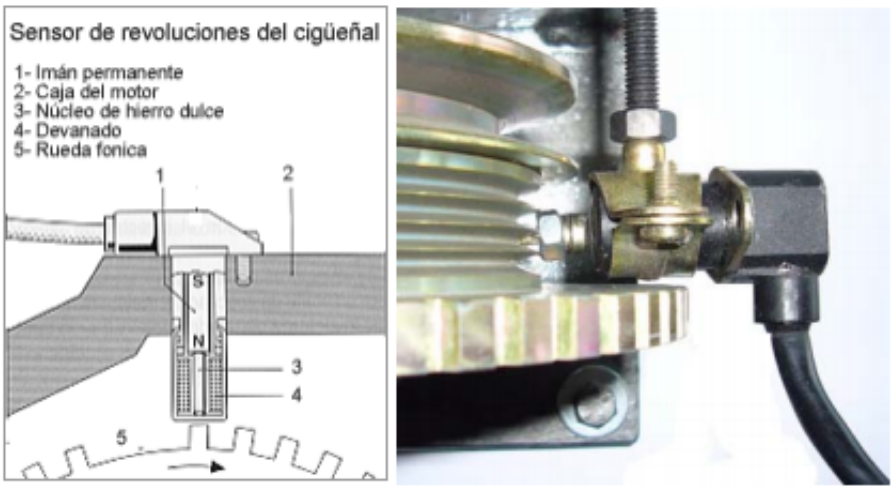

Figura 2.6: Sensor inductivo

La señal de salida obtenida por este tipo de sensor tiene una amplitud variable que depende de las revoluciones del motor, a mayor revolución mayor amplitud de señal y mayor frecuencia. En la figura(2.7) se representa la señal de salida en régimen de mínimas revoluciones (ralentí).

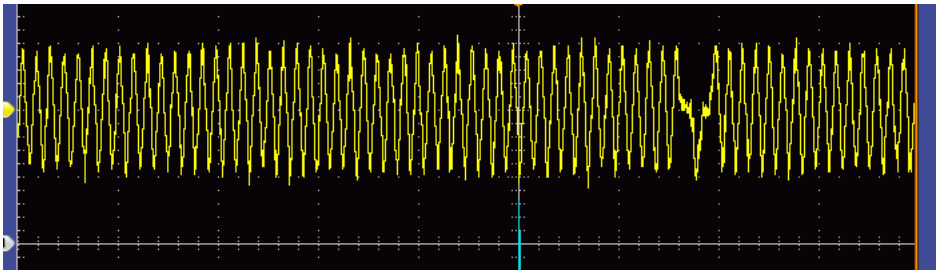

Figura 2.7: Señal de sensor inductivo

Sensor de presión de admisión (MAP)

El sensor de presión llamado en ingles Manifold Absolute Pressure (MAP), mide la presión del aire en la admisión. Es el primer sensor micromecanizado utilizado en la automoción. Está compuesto por un chip de silicio con dos partes,

un transductor de presión (membrana) y la electrónica de acondicionamiento. La membrana del sensor tiene resistencias que varían cuando se las somete a un esfuerzo. La señal de salida del puente, es del orden de los 100 mV, se la hace pasar por un amplificador de ganancia elevada y la señal de salida es del tipo analógica con rango de $0,5V a 4,5V$ como se muestra en la figura(2.8).

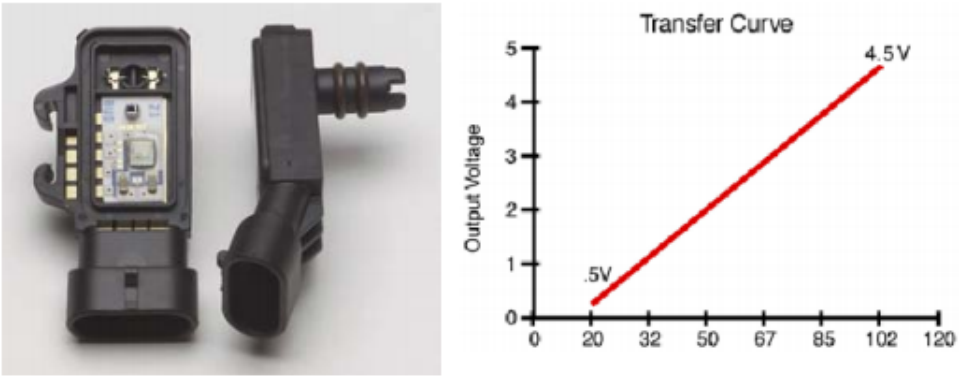

Figura 2.8: Sensor MAP (izquierda) Curva de transferencia del sensor (derecha) Tensión [V] vs Presión [Kpa]

Sensor de temperatura

Los sensores de temperatura se utilizan para medir la temperatura del refrigerante, aceite, aire de admisión, etc. En el caso del Renault Clio, posee dos sensores resistivos llamados **termistores**, y se utilizan para medir la temperatura del liquido refrigerante del motor y del aire de admisión. Estos sensores son del tipo **NTC** llamado así por sus siglas en ingles Negative Temperature Coefficient (Fig.2.9). Este sensor varía su resistencia de forma no lineal, inversamente a la variación de la temperatura, y de ahí sale el nombre de coeficiente negativo.

Figura 2.9: Termistor tipo NTC

Sensor de gases de escape

El sensor de **gases de escape**, o **sonda lambda** (λ) realiza la medición de los gases de escape, es el componente principal a la hora de cerrar el lazo de control.

Su salida se la compara con el valor de referencia de la UEC, en base a esto se ajusta el caudal de combustible con el fin de obtener una combustión completa.

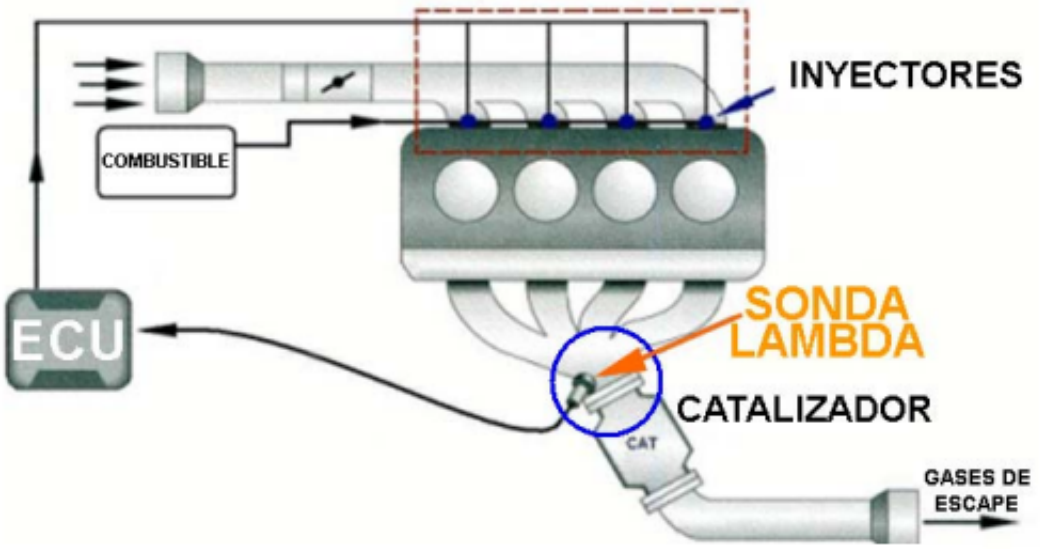

Figura 2.10: Ubicación de la sonda lambda en el lazo cerrado

En la fig.2.10 se observa la localización de dicha sonda y como la UEC toma la señal y controla los inyectores en base a la medición.

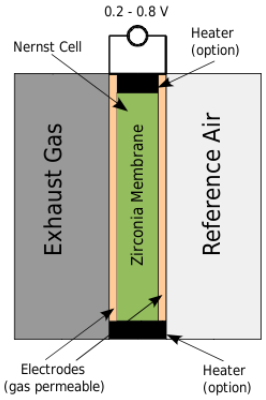

Figura 2.11: Composición de la sonda EGO

La sonda lambda capta los excesos o defectos de oxígeno de los gases de escape, dicha sonda está constituida por una parte **cerámica** y unos electrodos de **circonio** o **titanio**, figura(2.11). Los gases de escape están en contacto con la sonda y esta toma información de la proporción de dioxígeno residual tras la combustión.

En temperaturas superiores a $300°C$ la cerámica de la sonda, compuesta de dióxido de circonio y de itrio, se vuelve conductora de iones negativos de oxígeno. La diferencia de concentración da lugar a una difusión de iones del gas evacuado. Los átomos de oxígeno pueden moverse en la cerámica como iones de carga negativa doble. Los electrones necesarios para la ionización de los átomos de oxígeno son suministrados por los electrodos, que son conductores electrónicos. De esta forma puede tomarse el voltaje de la sonda entre los electrodos de platino situados dentro y fuera. Esta información se transmite por medio del cableado a la ECU del motor.
Existen básicamente dos tipos de sonda lambda, las de escalón o binarias y las sondas proporcionales. Las de escalón llamadas **EGO** (Exhaust Gas Oxigen sensor) ,figura(2.12), proporcionan un valor de lambda del tipo binaria o signo del error ,figura:(2.13). Para $\lambda > 1$ (mezcla pobre, falta de combustible o exceso de aire) toma un valor entre 0 y 200 mV. Para $\lambda < 1$ (mezcla rica), exceso de combustible o falta de aire) toma un valor entre 800 y 1000 mV.

Figura 2.12: Sonda lambda EGO

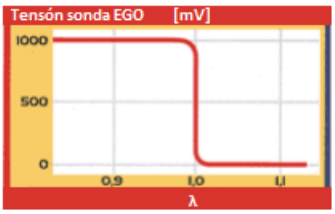

Figura 2.13: Curva de sonda EGO

La sonda proporcional **UEGO** (Universal EGO) o también llamado sensor de **"banda ancha"** se caracteriza por dar una señal analógica, como su nombre lo indica, proporcional al contenido de O_2 de los gases de escape. Este tipo de sonda es muy similar a las EGO pero también incorpora electrónica auxiliar en la que se utiliza una bomba de gas electroquímico, figura(2.14. Un circuito electrónico que contiene una retroalimentación de bucle controla la corriente de la bomba de gas

para mantener la salida de la constante de celda electroquímica, de manera que la corriente de la bomba indica directamente el contenido de oxígeno del gas de escape.

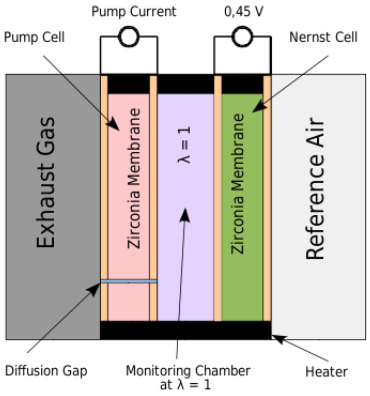

Figura 2.14: Composición de sonda UEGO

2.3.2. Actuadores

Acelerador electrónico y Mariposa Motorizada

El acelerador electrónico es un dispositivo que reemplaza la antigua conexión mecánica que existía entre el pedal del acelerador y la mariposa del colector de admisión en los vehículos equipados con motores de gasolina. Quedando sustituida por una conexión eléctrica a través de la central electrónica UEC.

En la figura(2.15) se observa como el accionar del pedal de aceleración modifica la tensión del potenciómetro de éste y así le indica a la UEC la necesidad de aceleración o desaceleración por parte del usuario. En base a la medición de posición de pedal, la UEC es la encargada de modificar la posición de la mariposa en forma controlada. Por motivos de seguridad el pedal de aceleración y la mariposa motorizada tienen dos potenciómetros cada uno para indicarle a la UEC la posición de los mismos evitando errores. La mariposa motorizada puede estar controlada por un motor paso a paso o por un motor de corriente continua (CC). En el caso del motor de CC éste puede ser de un único sentido de giro y por lo tanto la mariposa retorna a su posición de reposo mediante un muelle. De esta forma, no es necesario la utilización de un puente de transistores para que el motor pueda girar en ambas direcciones.

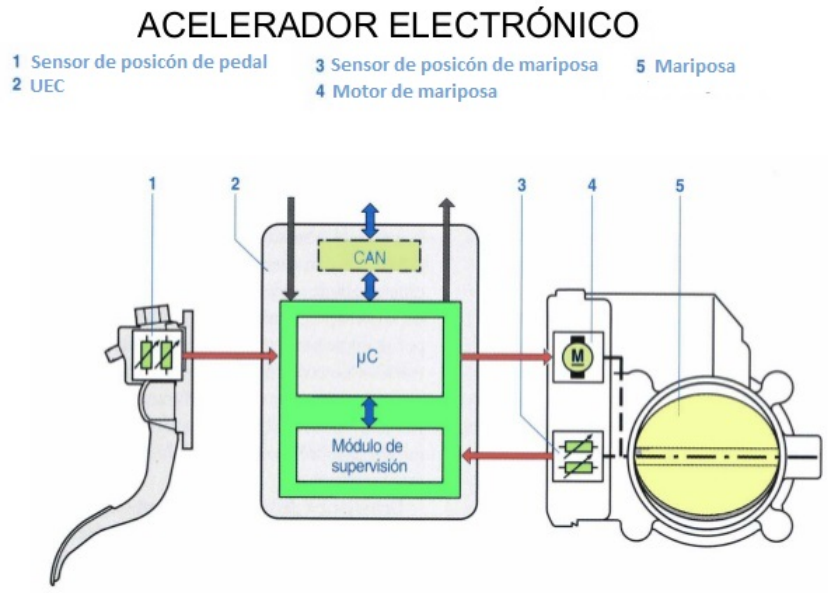

Figura 2.15: Diagrama de sistema de acelearador electrónico

Inyectores

El inyector cumple la función de introducir una determinada cantidad de combustible en el colector de admisión, sobre la válvula de admisión de la cámara combustión en forma pulverizada, distribuyéndolo lo más homogéneamente posible dentro del aire contenido en la cámara.
El funcionamiento básico de un inyector consiste en un solenoide que al hacerle pasar una determinada cantidad de corriente durante un tiempo controlado generará un campo magnético el cual moverá la aguja del inyector figura(2.16).

En la parte superior del inyector se encuentra la alimentación de combustible a una presión constante de entre 2,5 - 3,5 bar suministrada por una bomba eléctrica situada a la salida del depósito de combustible o dentro del mismo. Para suministrar combustible, el inyector debe recibir una señal a la entrada tipo pulso de duración finita suministrado por la UEC, y además el momento o avance de inyección también es comandado por la la misma en base a la información captada por el sensor de posición de cigüeñal y su respectiva velocidad, la masa de combustible suministrada es directamente proporcional al tiempo de inyección. En la figura (2.17) se muestra cómo es la conexión del inyector, y en la figura

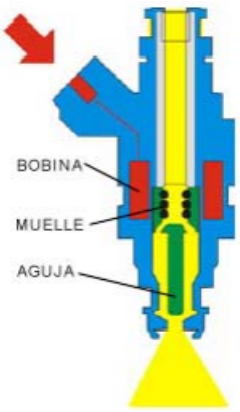

Figura 2.16: Electroinyector

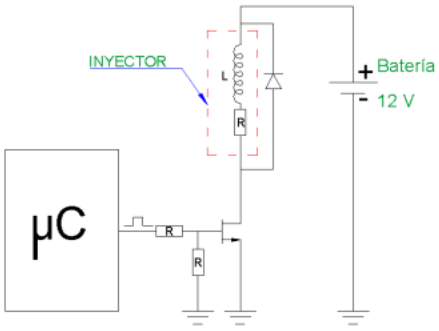

Figura 2.17: Diagrama del electroinyector

(2.18) se muestra la señal de entrada del inyector tomada desde un osciloscopio configurado con una base de tiempo de 10 ms/div y escala en tensión de 20 V/div.

Sistema de ignición

El encendido del motor es un sistema de producción y distribución de la chispa de alta tensión necesaria en la bujía para producir el encendido provocado en los motores de ciclo Otto.

Funcionamiento:

En la figura (2.19) se observan dos esquemas de encendido: **A** encendido clásico por ruptor, **B** encendido electrónico. En las figuras **Lp** es el bobinado primario, **Ls** es el bobinado secundario, **S** es el ruptor, **C** es el condensador, y por últimoT es el transistor de mando del primario.

A continuación se enumeran los componentes del sistema de ignición:

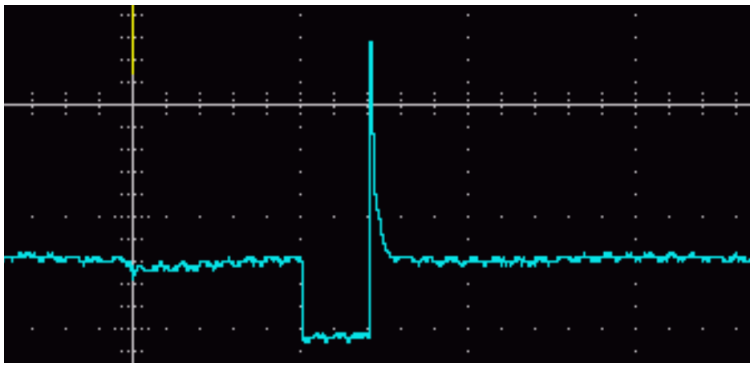

Figura 2.18: Señal de entrada al electroinyector

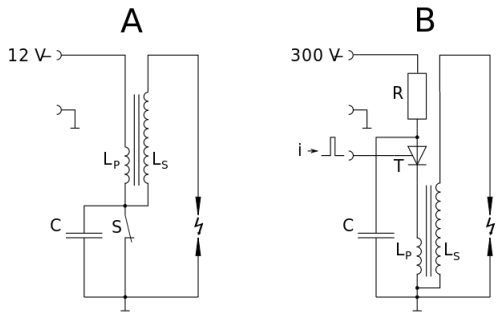

Figura 2.19: Diagrama de sistema de ignición

- Bobina: es un transformador inductivo con núcleo de hierro y dos devanados, uno de pocas espiras alimentado con el voltaje de batería (12V) desde el contacto o primario (Lp), y otro paralelo con 1000 veces más espiras, llamado secundario (Ls).Cuando en el circuito primario se interrumpe bruscamente la corriente, en el devanado secundario se genera una corriente de alta tensión, en este caso de 12.000V.

- Dispositivo de interrupción del primario: antiguamente mecánico, los llamados platinos o ruptor figura (2.19 A), fue sustituido por el encendido electrónico figura (2.19 B). Básicamente el encendido electrónico utiliza transistores de potencia con sincronización electrónica comandado por la UEC.

- Dispositivo de distribución de la corriente de alta a las bujías: se hacía antiguamente de forma mecánica mediante el Distribuidor. Hoy en día se hace de forma electrónica, ya que se agrupan las bujías por parejas en los cilin-

dros cuyos pistones trabajan paralelos, es decir la bujía del cilindro 1 con la del cilindro 4 y la bujía del cilindro 2 con la del cilindro 3 (dos Lp y dos Ls). Últimamente, inclusive se acopla una bobina por bujía (cuatro Lp y cuatro Ls), distribuyéndose únicamente la función de corte de cada bobinado primario desde la unidad electrónica de control de forma completamente independiente.

- Entre los electrodos de las bujías, dentro de la cámara de combustión, se produce un arco de plasma de unos 2 ms de duración. Este arco enciende la mezcla de combustible y aire, previamente comprimida. De esta forma se genera un aumento de presión considerable que produce la carrera útil de trabajo del pistón. [Wikipedia, 2016].

Capítulo 3

Modelo matemático de motor

3.1. Introducción

En el capítulo anterior se vio el funcionamiento del MCI junto con los sensores y actuadores que lo gestionan. El diseño de control de mezcla de aire/combustible requiere un modelo en el que pueda basarse y así lograr un control óptimo. En este capítulo se definen las ecuaciones del modelo matemático de un MCI aplicando la modelización de valor medio.

3.2. Antecedentes de modelos matemáticos

3.2.1. Sistema de aire

Existen diferentes autores que modelan el comportamiento del aire que circula dentro del colector de admisión e ingresa al cilindro, están los que modelan según las fluctuaciones de presión dentro del colector de admisión, y están los que mantienen contaste el flujo de aire dentro del cilindro asimilándola a una máquina rotativa en lugar de una máquina alternativa.

El modelo adoptado para esta practica es el de flujo de aire constante del autor Aquino [Aquino, 1981], donde se considera el colector de admisión como un depósito de volumen finito, al cual entra el aire por la válvula de mariposa, y sale a través de la válvula de admisión de cada uno de los cilindros. El aire es bombeado por los propios cilindros durante la carrera de admisión, y la válvula de mariposa actúa como pérdida de carga variable. Entonces se aplica la **ecuación de continuidad de la masa** de aire del colector de admisión. La figura (3.1) representa el sistema de aire, en donde la entrada es el flujo de aire que pasa por la válvula mariposa, y la salida es el flujo de aire que entra en los cilindros. La ecuación (3.1) modela el comportamiento de la masa de aire dentro del múltiple de admisión.

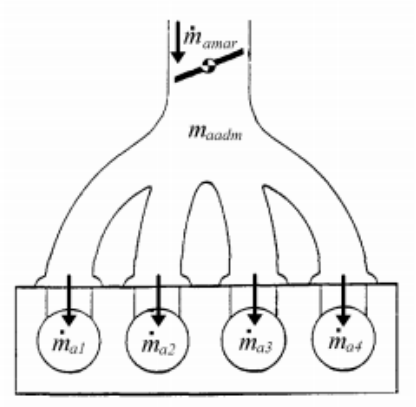

Figura 3.1: Diagrama de mariposa y colector de admisión.

$$\frac{dm_{aadm}}{dt} = \dot{m}_{amar} - \sum_{i=1}^{N_{cil}} \dot{m}_{ai} \tag{3.1}$$

Donde m_{aadm} es la masa de aire dentro del múltiple, $\dot{m}_{amar}$ es el flujo de aire que pasa por la válvula mariposa, y $\dot{m}_{ai}$ es el flujo de aire que entra en el cilindro i.

3.2.2. Válvula de mariposa

El flujo de aire que pasa por la válvula mariposa se trata como un fluido compresible que atraviesa un conducto, impulsado por una diferencia de presiones, en el que se produce una reducción de sección, debida a la mariposa [Taylor, 1966].

$$\dot{m}_{amar}(\alpha, \Phi) = \dot{m}^{max}_{amar} C_d f_1(\alpha) f_2(\Phi) + \dot{m}^0_{amar} \tag{3.2}$$

donde:

$$\dot{m}^{max}_{amar} = \frac{\pi D^2_{mar}}{4} . \frac{P_{adm}}{\sqrt{R.T_{adm}}} \tag{3.3}$$

Siendo $\dot{m}^{max}_{amar}$ el caudal de aire máximo posible, C_d el coeficiente de descarga, D_{mar} diámetro de la válvula de mariposa, P_{adm} la presión de admisión, T_{adm} la temperatura de admisión, $f_1(\alpha)$ el coeficiente de sección en función al ángulo de la mariposa, $f_2(\Phi)$ es una función por tramos que depende de la relación entre la presión ambiente y la presión de admisión y representa el factor de presión, el término $\dot{m}^0_{amar}$ representa el flujo de aire en estado ralentí .

$$\Phi = \frac{P_{adm}}{P_{amb}} \tag{3.4}$$

La función $f_2(\Phi)$ se encuentra dividido en dos tramos, uno para el flujo sónico, donde el flujo de aire alcanza la velocidad del sonido y otro para el flujo subsónico, este limite esta dado para Φ^* y se llama presión crítica de Laval:

$$\Phi^* = (\frac{2}{k+1})^{\frac{k}{k-1}} \tag{3.5}$$

Donde k es el cociente de calores específicos a presión y volumen constantes. Si se trata al aire como un gas ideal entonces $k = 1,4$, se obtiene $\Phi^* = 0,528$.

$$f_2(x) = \begin{cases} \sqrt{\frac{2k}{k-1}(\Phi^{\frac{2}{k}} - \Phi^{\frac{k+1}{k}})} & si \quad \Phi^* \leq \Phi \leq 1 \\ \\ \sqrt{k}(\frac{2}{k+1})^{\frac{k+1}{2(k-1)}} = 0,684 & si \quad \Phi \leq \Phi^* \end{cases} \tag{3.6}$$

La función $f_1(\alpha)$ representa la sección de paso de la válvula de mariposa, que depende del ángulo α. Para el caso de un tubo cilíndrico, con mariposa circular articulada por su diámetro:

$$f_1(\alpha) \simeq (1 - \frac{cos(\alpha)}{cos(\alpha_0)}) \tag{3.7}$$

Siendo α el ángulo de posición de la mariposa, y α_0 el ángulo de la mariposa en posición en régimen ralentí. La expresión 3.2 no se cumple para ángulos grandes y velocidades de motor bajas, donde se produce un efecto de saturación debido a que la pérdida de carga lineal en todo el conducto de admisión deja de ser despreciable respecto a la creada por la mariposa. En tal caso el flujo de aire depende fuertemente de las revoluciones del motor (n) y de dos constantes a y b que dependen del motor.

$$\dot{m}_{amar}(n) = a + b.n \tag{3.8}$$

3.2.3. Válvula de admisión

A la hora de calcular el aire que atraviesa la válvula de admisión, a diferencia de la válvula de mariposa en donde se trabaja con el coeficiente de descarga según la apertura de la misma, es preferible trabajar con lo que se llama **rendimiento volumétrico**, el cual equivale a un valor promediado para todo el ciclo, y depende de la presión de admisión y la temperatura de aire de admisión, asumiendo que el aire es un gas ideal:

$$\dot{m}_a = \frac{1}{2}.n.V.\rho_{adm}.\eta_{vol} = \frac{1}{2}.n.V.\frac{P_{adm}}{R.T_{adm}}.\eta_{vol} \tag{3.9}$$

donde V es la cilindrada. El factor$\frac{1}{2}$ procede del hecho que, en un motor de cuatro tiempos, se produce una renovación de carga cada dos vueltas, n son las revoluciones por minuto, P_{adm} es la presión dentro del múltiple de admisión, R es la constante universal de los gases ideales, T_{adm} es la temperatura del aire dentro del múltiple de admisión [K], y η_{vol} es la eficiencia volumétrica que es una función que depende de n y de P_{adm} .

3.2.4. Ecuación de estado del colector de admisión

Partiendo de la ecuación 3.1, y suponiendo que el aire es un gas ideal, se obtiene la expresión de la variación de la presión en el colector de admisión. Considerando que la temperatura no varía o lo hace muy lentamente, y que el volumen del colector es constante:

$$\dot{P}_{adm} = \frac{R.T_{adm}}{V_{adm}}.(\dot{m}_{amar} - \sum_{1}^{N_{cil}} \dot{m}_a) \tag{3.10}$$

Si ahora se sustituye la expresión 3.9 en la anterior, se obtiene la ecuación de estado del colector de admisión.

$$\dot{P}_{adm} = \frac{R.T_{adm}}{V_{adm}}.\dot{m}_{amar} - \frac{1}{2}.n.V.\frac{P_{adm}}{V_{adm}}.\eta_{vol} \tag{3.11}$$

3.3. Admisión de la mezcla

Al cerrarse la válvula de admisión, el dosado de la mezcla permanece constante hasta que la inyección se repita para el mismo cilindro. Sin embargo, hasta que se obtenga información sobre λ, suceden una serie de fenómenos:

1. Expulsión de los gases de escape.
2. Transporte de los gases hasta la posición de la sonda.
3. Medición a cargo de la sonda.

3.3.1. Expulsión de los gases de escape

Luego de la inyección comienzan los ciclos del motor y pasa un tiempo hasta que los gases de escape lleguen a la sonda. En primera instancia surgen los primero 3 tiempos o fases de motor (admisión, compresión y combustión), y tan pronto como se presenta la fase de escape en la cual la válvula de escape comienza a abrirse, los gases quemados del cilindro son expulsados hacia el colector de escape. El primer proceso antes de la apertura de la válvula de escape tiene un tiempo

de ejecución que depende sólo del régimen del motor (RPM). Por tanto, si se define θ_{esc} como el ángulo de cigüeñal comprendido entre el instante de la inyección y la apertura de las válvulas de escape en el mismo cilindro, y n es el régimen en RPM, el retardo de ciclo en un motor de 4 tiempos es equivalente a:

$$t_{cic} = \frac{\theta_{esc}(^\circ/360)}{n(RPM/60)} \tag{3.12}$$

3.3.2. Transporte de los gases de escape

Una vez finalizado el proceso de escape, los gases van ocupando el volumen existente en el colector y tubo de escape, hasta llegar a la posición donde está situada la sonda de medición. Este tiempo se llama tiempo de transporte t_r el cual se puede hallar mediante la formula siguiente:

$$t_r = \frac{2}{V}.\frac{1}{n.\eta_{vol}}.\frac{1}{1-Xr}.\frac{P_{esc}}{P_{adm}}.\frac{T_{adm}}{T_{esc}} \tag{3.13}$$

Donde Xr es la fracción de gases residuales respecto a la mezcla admitida, V es la cilindrada total y η_{vol} es el rendimiento volumétrico.
Puede verse que el retado de transporte tiene una fuerte dependencia del punto de funcionamiento del motor. En efecto, además de la intervención directa de n, hay una dependencia implícita del régimen, de la carga en el rendimiento volumétrico, las fracciones residuales, las presiones y temperaturas de escape.

3.4. Retardo total

Según los resultados experimentales de laboratorio hechas por Cercós Javier [Cercós, 2001], el retardo puede quedar representado solamente por el régimen de funcionamiento y el ángulo α de la mariposa del colector de admisión quedando el retardo representado de la siguiente forma:

$$T_d = t_\lambda = t_{cic} + t_r = \frac{C_1}{n} + C_2.e^{C_3.\alpha} \tag{3.14}$$

C_i son contantes que dependen del motor, dando como resultado:

$$C_1 = 180, C_2 = 0,319, C_3 = -0{,}17$$

$$T_d = \frac{180}{n} + 0,319.e^{-0{,}17.\alpha} \tag{3.15}$$

3.5. Dinámica del sistema

En condiciones de funcionamiento normal del coche, la potencia mecánica generada por el motor se utiliza en parte para vencer las resistencias pasivas del motor y del vehículo, y la potencia todavía disponible permite acelerar el motor, que esta rígidamente unido mediante un embrague a la caja de cambios y luego las ruedas.

3.5.1. Combustión y generación de par motor

Dado un caudal de combustible $\dot{m}_f$, la potencia mecánica teórica disponible es:

$$N = \dot{m}_f.H_f.\eta_t(P,\lambda,n) = \frac{\dot{m}_a}{\lambda.\lambda_e}.H_f.\eta_t(P_{adm},\lambda,n) \tag{3.16}$$

Donde H_f es el poder calorífico inferior del combustible, y η_t es el rendimiento térmico. La entrada de potencia se considera instantánea, y si el ángulo de encendido es óptimo, esto sucede aproximadamente cuando el pistón está 60° después del PMS, y si se supone que la inyección se produce 60° antes del PMS del pistón, en el tiempo de escape, se tiene un retardo de θ_{comb} de aproximadamente 480°. Esto significa que hay un retardo entre la inyección y la generación de par motor de:

$$t_{comb} = \frac{\theta_{comb}}{360} \cdot \frac{1}{n(RPM/60)} = \frac{\theta_{comb}}{60n} \tag{3.17}$$

El rendimiento térmico es la fracción de la energía de la gasolina que se aprovecha para producir trabajo mecánico, respecto a la teóricamente disponible a partir de la reacción de combustión perfecta. Esto evita tener que modelar las pérdidas de energía térmica por los gases de escape y el refrigerante. Por el momento no es posible hallar un modelo que determine el valor del rendimiento, pero sí se conoce su dependencia respecto a variables como el régimen, presión de admisión y el dosado de la mezcla. No se tiene en cuenta el encendido porque se supone óptimo.

3.6. Balance dinámico

Las ecuaciones que gobiernan la dinámica del sistema se obtienen de aplicar la segunda ley de Newton en el volante de motor. La potencia obtenida de la combustión de la mezcla genera un par indicado, el cual permite vencer las resistencias pasivas del motor, las resistencias de avance del vehículo y si queda par disponible, acelerar el conjunto **transmisión-vehículo**, caracterizado por su

inercia equivalente. El régimen se calcula por integración. Como hipótesis, se supone que no hay deslizamiento de las ruedas motrices ni del embriague, y que la carretera es llana, así:

$$M_{neto} = \dot{n} I_{eq} = \frac{N(Kw \cdot 10^3)}{n(RPM \cdot 2\pi/60)} - M_f(n) - M_l(v) = \frac{3 \cdot 10^4}{\pi} \frac{N}{n} - M_f(n) - M_l(v) \tag{3.18}$$

Las perdidas del motor $M_f(n)$ son las que representan por un lado el rozamiento interno del tren mecánico y la distribución, y por otro lado los periféricos, como la bomba de aceite, alternador, compresor de aire acondicionado, etc y depende de las revoluciones del motor y no de la velocidad del vehículo. Hay autores que representan $M_f(n)$ como un modelo polinomial de segundo orden quedando expresado de la siguiente forma:

$$M_f(n) = M_{f_0} + M_{f_1} n + M_{f_2} n^2 \tag{3.19}$$

El par $M_l(v)$ es producto del avance del vehículo y es función de la velocidad, además depende del radio del neumático (R_r) , el rendimiento global de la transmisión ($\boldsymbol{\eta}_{tr}$), y la relación de transmisión de cada marcha (RT(i)):

$$M_l(v) = \frac{R_r}{RT(i) \cdot \eta_{tr}} \cdot (F_{lo} + F_{l1}.v + F_{l2}.v^2) \tag{3.20}$$

La expresión $F_{lo} + F_{l1}.v + F_{l2}.v^2$ representa la resistencia al avance del vehículo sobre las ruedas que procede de la fuerza aerodinámica y de la rodadura de los neumáticos.

La inercia equivalente (I_{eq}) incluye el momento de inercia del volante y el cigüeñal del motor más el eje primario de la transmisión (I_{mot}), las ruedas (R_r,I_r) y la masa del vehículo (M_v), todo reducido al volante del motor y teniendo en cuenta la relación de transmisión de cada marcha, $RT(i)$:

$$I_{eq} = I_{mot} + \frac{M_v.R_r^2 + 4I_r}{RT(i)} \tag{3.21}$$

Capítulo 4

Control de flujo de combustible

4.1. Introducción

Como se vio en el Capítulo 3, la forma de suministrar y controlar el flujo de combustible en los cilindros se logra por medio de los inyectores actuando directamente sobre el tiempo de apertura de los mismo (ancho de pulso), y así se obtiene el dosado que el motor necesita para su correcto funcionamiento (eficiencia y reducción de gases contaminantes). Este ancho de pulso o tiempo de inyección tiene que estar basado en un esquema de control dependiendo de los parámetros de entrada (temperatura, presión, RPM, etc) y la realimentación de la sonda lambda.

4.2. Flujo de combustible

Como se mencionó en capítulos anteriores, en un motor de 4 tiempos, un ciclo de motor corresponde a 2 vueltas de motor, entonces por cilindro hay una inyección cada 2 vueltas. Al trabajar sobre un motor de 4 cilindros en los que se inyecta sobre 2 cilindros en la primer vuelta y 2 en la segunda, el flujo másico de combustible queda representado por la siguiente ecuación:

$$\dot{m}_c[g/min] = 2.t_i.G_i.n \tag{4.1}$$

donde t_i [s] es el tiempo de apertura del inyector, G_i [g/s] es la ganancia del inyector, y n son las revoluciones o régimen.

4.3. Esquema de control

Un esquema convencional de control de inyección de combustible es del tipo **feedforward**+**feedback**, donde el control feedforward (FF) es un componente

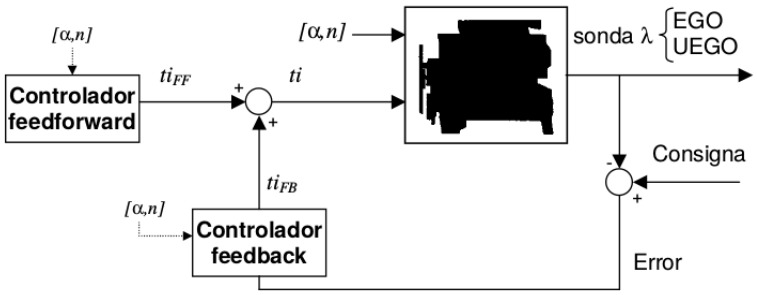

Figura 4.1: Diagrama control de inyección

anticipativo, en lazo abierto, y el control feedback (FB) es del tipo correctivo, en lazo cerrado. Entonces la ECU toma los parámetros de entrada y gestiona primero el control feedforwar y luego tomando la realimentación por medio de la sonda lambda se hace una corrección del flujo de combustible.
En la figura (4.1) se ve un diagrama de cómo opera este tipo de control. Comienza el control FF tomando únicamente parámetros de entrada y ninguno de salida, y calcula un tiempo de inyección base. El control FB sólo toma el estado de la salida (rica o pobre) y se encarga de aumentar o disminuir el tiempo de inyección calculado por el otro controlador.
Al utilizar el esquema de control FF + FB, se obtiene un tiempo de inyección FF (ti_{FF}) el cual se utiliza para tener una aproximación inicial del dosado necesario para mantener una relación estequiométrica entre el aire que ingresa y el combustible que se inyecta. Una vez estabilizado le motor, se ejecuta el control feedback el cual compara la salida con la referencia o consigna ($\lambda_{ref} = 1$) dando como resultado un tiempo de corrección (ti_{FB}), que luego se suma al tiempo inicial para obtener un valor de tiempo total ($ti = ti_{FF} + ti_{FB}$).

4.4. Control Feedforward

La ecuación 2.2 demuestra que para mantener un $\lambda = 1$, es necesario tener un $AFR = 14{,}7$ (ec. 2.1), en donde el numerador del cociente es la masa de aire ingresado y el denominador es la masa de combustible inyectada. Entonces el control feedforward, de manera robusta, usa como principal parámetro de entrada el **grado de carga** o la **masa de aire** que ingresa y así calcula la masa de combustible necesaria para mantener una $\lambda = 1$.

4.4.1. Carga o caudal de aire

Existen dos formas de obtener el grado de carga de un motor, una es la forma **directa** en el que el vehículo tiene un sensor de caudal másico y así se obtiene de forma directa la masa de aire. Por otro lado existe la forma **indirecta** en el que se estima el caudal de aire en base de los parámetros de RPM, presión de admisión, temperatura, ángulo de la mariposa.
Cuando el vehículo no cuenta con un sensor de caudal másico, se utiliza la estimación, por lo tanto se estaría cometiendo un error mayor que en los métodos directos. La ecuación (4.2) representa el flujo de la masa de aire en función de las **RPM**, la temperatura del aire de admisión [kelvin], la cilindrada del motor **V** [litros], y la contstante **R** ($\frac{Kj}{Kg}$) de los gases ideales y el rendimiento o eficiencia volumétrica.

$$\dot{m}_a[g/min] = \frac{1}{2} \cdot \frac{V.\eta_{vol}.P_{adm}.n}{R.T_{adm}} \cdot 1000 \tag{4.2}$$

Los parámetros a medir son la presión de admisión, las RPM y la temperatura del aire que ingresa por el colector de admisión. Por otro lado η_{vol} depende del punto de funcionamiento del motor y puede tomar valores que están tabulados en una tabla o mapa de rendimiento volumétrico, que depende de la presión de admisión y de las revoluciones. En este proyecto se empleó una tabla genérica para motores de características similares al empleado.

$\eta_{vol} =$

$Pad[Kpa]$									
10,3	19,9	29,9	39,8	50,2	60,1	70,1	80,0	90,0	RPM
75,5	75,0	74,5	74,5	74,0	70,5	70,0	71,5	71,5	750
75,0	75,5	78,9	75,0	74,0	73,5	72,5	71,5	71,5	1000
77,0	77,5	79,0	76,5	75,5	74,5	74,0	73,0	73,0	1125
77,5	79,5	81,0	77,5	76,5	75,5	74,0	73,0	73,0	1250
80,0	84,0	84,0	79,0	77,5	76,0	74,0	73,0	74,5	1500
80,0	83,0	82,0	80,0	78,0	77,0	75,5	74,5	74,5	1750
79,5	82,0	84,0	83,0	81,0	79,5	79,0	78,0	80,5	2000
78,5	82,0	86,0	87,0	85,0	84,0	82,0	81,0	81,0	2260
78,0	82,0	88,5	88,0	86,5	85,5	83,0	82,0	81,0	2500
77,5	82,0	89,5	88,0	89,0	83,5	81,5	80,0	72,5	2750
75,5	81,5	89,0	87,0	91,5	81,0	79,0	79,0	75,0	3000
68,0	79,5	90,5	90,0	91,5	85,0	84,0	81,5	76,0	3500
66,0	78,0	90,0	92,0	92,0	89,0	86,5	83,5	80,0	4000
73,5	77,5	85,5	90,0	91,0	88,5	87,0	84,0	80,0	4500
77,5	78,5	85,5	89,5	91,0	88,0	85,0	81,0	78,0	5000
84,0	83,5	84,5	88,0	91,0	87,0	83,0	79,5	76,5	5500
84,0	83,5	84,5	87,5	91,0	87,0	84,0	79,0	76,5	6000
84,0	83,5	84,5	87,5	91,0	87,5	84,5	79,5	76,5	6500

4.4.2. Tiempo de inyección FeedForward

El objetivo principal del control de inyección es mantener la relación estequiométrica alrededor de 14,6. lo que significa que cada 14,7 miligramos de aire que ingresa al cilindro tiene que haber 1 miligramo de combustible. Entonces para obtener el **tiempo de inyección Feedforward** $(t_{i_{FF}})$ hay que hacer el cociente entre la ecuación 4.2 y 4.1 e igualarlos a 14,7 y despejando se obtiene $t_{i_{FF}}$:

$$t_{i_{FF}}[s] = \frac{1}{2} \cdot \frac{1000.V.\eta_{vol}.P_{adm}.n}{R.T_{adm}} \cdot \frac{1}{14,6} \cdot \frac{1}{2.G_i.n} \tag{4.3}$$

cancelando **n** entre el numerador y denominador:

$$t_{i_{FF}}[s] = \frac{1}{2} \cdot \frac{1000.V.\eta_{vol}.P_{adm}}{R.T_{adm}} \cdot \frac{1}{14,6} \cdot \frac{1}{2.G_i} \tag{4.4}$$

Se puede observar que la cantidad de combustible ha inyectar no depende directamente de las RPM, pero sí depende de manera directa de la presión de admisión P_{adm}.

4.5. Control Feedback

El control feedback consiste en medir los gases de escape por medio de una sonda EGO (binaria) y establecer una estrategia de corrección de flujo de combustible. La estrategia de control tradicional es del tipo controlador **P**, **I**, o **PI**. Es importante destacar que no se puede utilizar el control derivativo **D** a una sonda EGO debido a su forma de actuar del tipo binaria o relay, al derivar este tipo de sonda, en los flancos de subida o bajada durante los cambios de estado de la mezcla generarían un comportamiento divergente saturando al sistema de forma alternada.
Debido a los importantes retardos del sistema es necesario establecer como condición necesaria que el motor se encuentre en un estado estacionario, es decir que las variaciones de RPM y las variaciones de la presión del colector de admisión sean nulas o dentro de un rango limitado.

La figura (4.2) representa el diagrama en bloque del sistema de control para la relación estequiométrica λ. Esta figura muestra como el factor λ es controlado por la suma de dos variables que se suman. La variable que sale del controlador FB adopta el nombre de **tiempo de inyección feedback** (ti_{fb}) el cual se suma al tiempo feedforward (ti_{ff}) para obtener un tiempo total (ti), al multiplicar el tiempo **ti** por el bloque ganancia de inyector se obtiene la masa total de combustible que entra al motor para luego realizar una combustión lo mas completa posible. En el caso de este proyecto se trabajó con un inyector marca **Magneti Marelli**

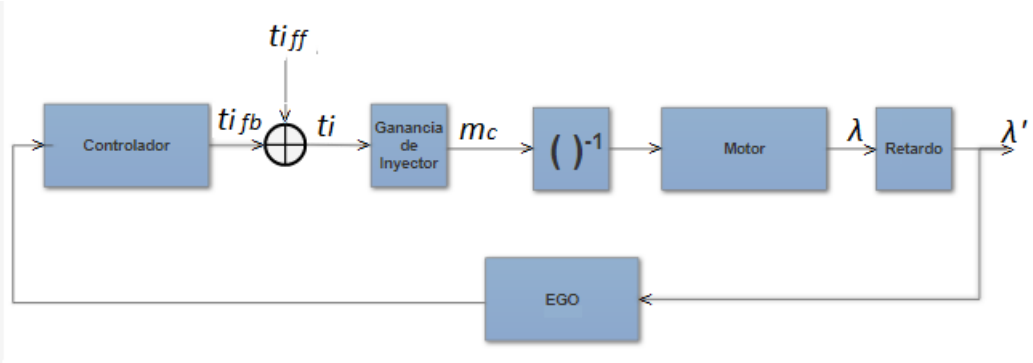

Figura 4.2: Sistema A/C

modelo iwp 143 que según los datos del fabricante posee una ganancia de 2.333 mg/ms a una presión de 3 bar.
Como λ es el cociente entre la masa de aire (m_a) y la masa de combustible (m_c) es importante notar que la variable que entra al motor es $\frac{1}{m_c}$ (inversa de la masa de combustible) y es por esto que aparece un bloque $\boxed{()^{-1}}$ posterior a m_c y previo a la planta que representa el bloque Motor.
Por último hay que tener en cuenta que la salida medida es luego del Retardo donde es λ' y no λ. Cabe destacar que este retardo quedó plasmado en la ecuación (3.15) y depende del ángulo de mariposa, el régimen de funcionamiento y el tipo de motor.

4.5.1. Aproximación de función transferencia de motor

Como se mencionó en el Capítulo 1, una de las dificultades a la hora de plantear el control de λ es la función transferencia de la planta cuyo orden es desconocido. Los fenómenos de formación de mezcla son muy complejos, la dinámica depende del punto de funcionamiento y el proceso es discontinuo. Por estas razones se adopta un modelo genérico para la función transferencia $G(s)$ y así se logra un control más robusto.

$$G(s) = \frac{G_p}{1 + sT_0} \cdot e^{-sT_d} \tag{4.5}$$

Donde G_p es la ganancia de la planta $G(s)$, y representa la masa de aire dividido AFR (Recordar que $\lambda = \frac{m_a}{m_c.AFR}$), $T_0 = 0{,}5$ es un polo que simplifica el comportamiento dinámico del combustible y la evolución de los gases, y T_d es el retardo de λ. Estos valores de la ecuación (4.5) dependen fuertemente del punto de funcionamiento y es por esto que el análisis se centra solamente bajo condiciones de

ralentí ($RPM = 750 \pm 100$, $\alpha = 11\,\% = 10°$) debido a sus grandes retardos que generan límites a la hora de obtener un control óptimo.
Ganancia de planta G_p:

$$G_p[g] = \frac{m_{a-cilindro}}{AFR}$$

$$G_p[g] = \frac{1}{4} \cdot \frac{V.\eta_{vol}.P_{adm}}{R.T_{adm}} \cdot \frac{1000}{14,6}$$

$$G_p[g] = \frac{1}{4} \cdot \frac{1{,}149\ l \cdot 0,74 \cdot 39\ KPa}{287 \frac{j}{Kg.K} 300\ K} \cdot \frac{1000}{14,6}$$

$$G_p[g] = 0,00659g$$

$$G_p[mg] = 6,59mg \tag{4.6}$$

Retardo:

$$T_d = \frac{180}{n} + 0,319 \cdot e^{-0,17 \cdot \alpha}$$

$$T_d = \frac{180}{750} + 0,319 \cdot e^{-0,17 \cdot 10°}$$

$$T_d = 0,42s \tag{4.7}$$

4.6. Controlador FB

La forma de resolver el error cometido por el controlador FF sobre el valor de lambda es por medio de la realimentación por sonda EGO. Este proceso compara la presencia o ausencia de oxígeno en los gases de escape y en base a su respuesta se ejecuta una acción de control que busca asegurar una zona de trabajo. La trasferencia de un control PI queda conformada por un valor proporcional Kp sumado a una integración con una constante Ki.

$$PI(S) = (Kp + \frac{Ki}{S}) \tag{4.8}$$

$$PI(S) = \frac{Ki}{S} \cdot (\frac{Kp}{Ki}S + 1) = \frac{Ki}{S} \cdot (z_1 S + 1) \tag{4.9}$$

Como se puede observar en la figura (4.3), la entrada al controlador PI es un escalón unitario positivo para el caso en que $\lambda > 1$ (mezcla pobre) y un escalón unitario negativo para el caso en que $\lambda < 1$ (mezcla rica). Entonces teniendo en cuenta este comportamiento de entrada se puede analizar la salida del controlador como la respuesta al escalón más las condiciones iniciales del mismo. Notar que

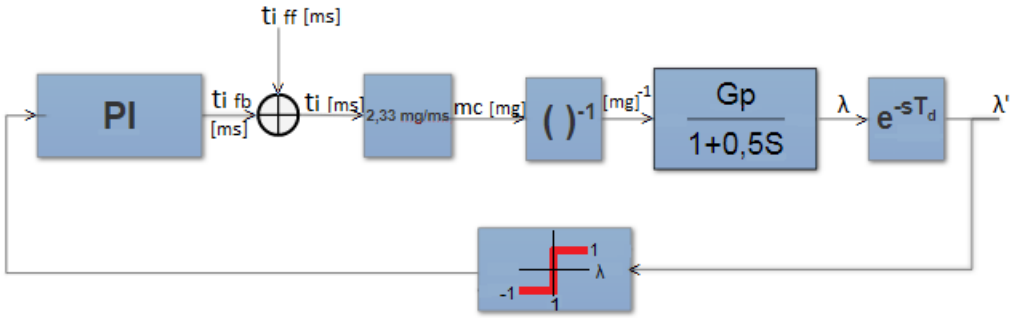

Figura 4.3: Sistema A/C

para el análisis de control hay que linealizar el bloque $\boxed{()^{-1}}$ dentro de un punto de trabajo.

- Primer caso:$\lambda > 1$ $EGO = 1$

$$ti_{fb}[ms] = ti_{fb0}[ms] + Kp[ms] + Ki[ms/s].t[s] \tag{4.10}$$

- Segundo caso $\lambda < 1$ $EGO = -1$

$$ti_{fb}[ms] = ti_{fb0}[ms] - Kp[ms] - Ki[ms/s].t[s] \tag{4.11}$$

Las ecuaciones (4.10) y (4.11) reflejan el comportamiento y la evolución temporal de la corrección del tiempo de inyección del control PI (ti_{fb}) y no el tiempo total de inyección (t_i), y es por esto que pueden tomar valores tanto positivos como negativos o incluso nulos. El termino ti_{fb0} representa la condición inicial de la corrección, que luego se modifica con un salto en escalón de Kp milisegundos y a medida que evoluciona la variable **t**, el escalón de entrada se integra en forma de rampa con pendiente Ki milisegundos/segundo.

En la imagen (4.4) se muestra de manera conceptual la respuesta del control integral, proporcional y proporcional-integral tomando a la sonda lambda como un ciclo límite con período $2.T_1$. Cuando el estado de la sonda EGO es rico entonces el tiempo de inyección baja y de manera inversa cuando el estado de la sonda EGO es pobre entonces el tiempo de inyección sube.

4.6.1. Valores Kp y Ki

Para obtener los valores de Kp y Ki se parte del análisis de respuesta temporal de λ al rededor de un punto de trabajo y se linealiza $\boxed{()^{-1}}$, luego se fijan los

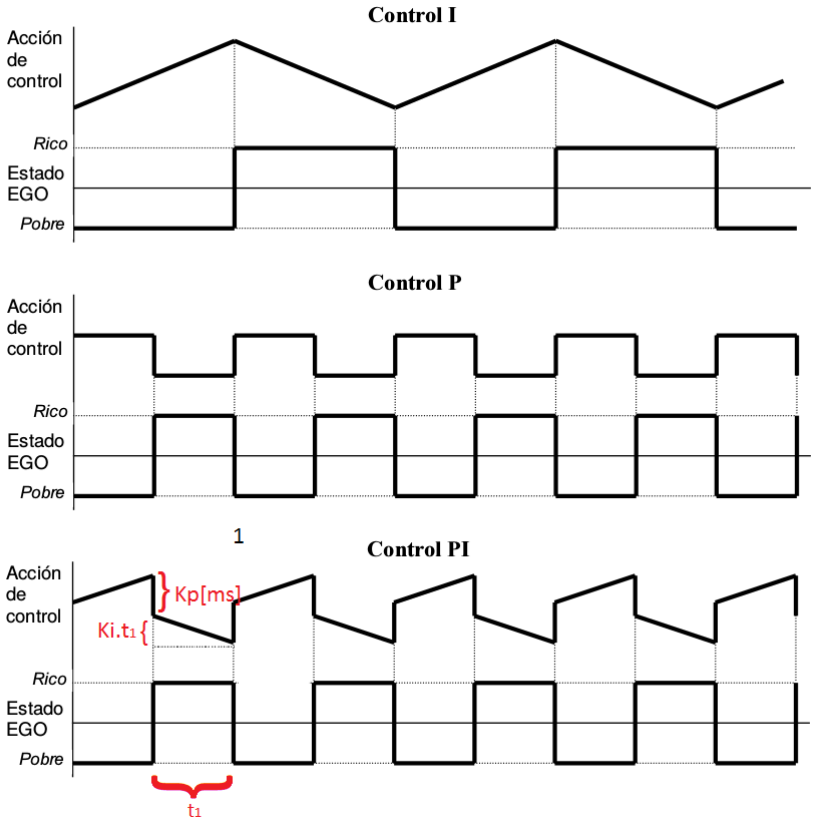

Figura 4.4: Control I, P, PI

límites de valores que λ puede tomar.

Punto de trabajo

El punto de trabajo se fija para valores de RPM de 700 ± 50, temperatura de 300 K, presión de admisión entre 35 y 39 Kpa, y retardo de 420 ms. Como se vio en la ecuación (4.6) la ganancia de la planta es de $6,59[mg]$ por lo tanto para que λ dé lo más cercano posible a 1, la masa de combustible tiene que ser igual a la ganancia:

$$\frac{6,59\ mg}{m_c} = \lambda = 1$$

$$m_c = 6,59\ mg$$

Como la ganancia del inyector es de $2,333$ mg/ms, entonces el tiempo de inyec-

ción t_i es:

$$m_c = G_i \cdot t_i$$
$$t_i = \frac{6,59}{2,33}\ ms$$
$$t_i = 2,82\ ms$$

El siguiente paso es linealizar por Tylor la función f(T_i)=$\frac{1}{G_i.t_i}$ alrededor del punto de trabajo $t_i = 2,82\ ms$:

$$f(t_i) = f(t_{i0}) + (\frac{\partial f(t_{i0})}{\partial t}) \cdot (t_i - t_{i0})$$
$$f(t_i) = \frac{1}{G_i t_{i0}} + (\frac{-1}{(G_i t_i)^2}) \cdot G_i(t_i - t_{i0})$$
$$f(t_i)[\frac{1}{mg}] = 0,151 - 0,0537 \cdot (t_i - 2,82)$$

Análisis de la respuesta temporal de λ en función de Kp y Ki

Para obtener la respuesta temporal de λ en función de Kp y Ki hay que hacer la transformada inversa de laplace de la respuesta al escalón del sistema realimentado y analizar el comportamiento junto con el retardo.

$$G'_p = 0,0537 mg^{-1}.ms^{-1} \cdot G_p \tag{4.12}$$
$$G'_p = 0,353 ms^{-1} \tag{4.13}$$
$$\lambda(t) = \mathcal{L}^{-1}\{\frac{1}{s}(Kp + \frac{Ki}{s})(\frac{0,353}{1 + 0,5s})\} \tag{4.14}$$
$$\lambda(t) = 0,353 ms^{-1}[\frac{1}{2}(Ki - 2s^{-1} \cdot Kp)e^{-2t} + \frac{1}{2}(2Ki.t - Ki + 2s^{-1} \cdot Kp)] \tag{4.15}$$

Reagrupando los términos y sacando factor común queda:

$$\lambda(t) = \frac{0,353 ms^{-1}}{2}[(Ki - 2s^{-1} \cdot Kp)e^{-2t} + 2Ki.t - (Ki - 2s^{-1} \cdot Kp)]$$

tomando $A = (Ki - 2s^{-1} \cdot Kp)$:

$$\lambda(t) = \frac{0,353 ms^{-1}}{2} \cdot [A.e^{-2t} + 2Ki \cdot t - A]$$
$$\lambda(t) = \frac{0,353 ms^{-1}}{2} \cdot [A(e^{-2t} - 1) + 2Ki \cdot t]$$

Se pueden observar dos términos que dependen de la variable t. Uno es la función exponencial que representa la respuesta transitoria y el otro término es lineal que representa la respuesta permanente. Para obtener los valores de Kp y Ki es necesario limitar el valor de excursión de λ y a la vez asumir una relación entre Kp y Ki de modo de reducir el sistema a una sola incógnita. Notar que al hacer $Ki = 2s_{-1}ms^{-1}Kp$ el cero del controlador coincide con el polo de la planta, esto hace que $A = 0$ y deja solo el término lineal quedando el sistema completo solamente como un integrador.

$$\lambda(t) = 0,353ms^{-1} \cdot Ki \cdot t \tag{4.16}$$

Se considera que λ tiene una tolerancia de $1 \pm 0{,}05$, de modo que la amplitud del ciclo límite se tiene que mantener dentro de un rango $\lambda_{min} = 0,95$ a $\lambda_{max} = 1,05$.

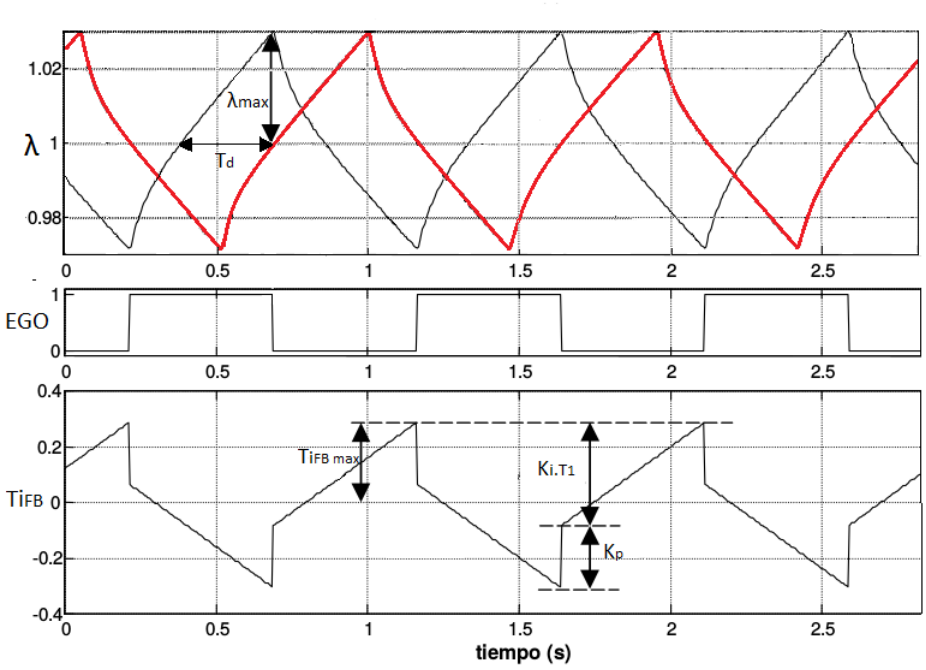

Figura 4.5: Respuesta del sistema al control PI

Para entender mejor el comportamiento temporal de λ con un control PI, la figura (4.5) no sólo ilustra el ciclo límite de lambda en linea negra, sino que también ilustra la misma señal en rojo retrasada un tiempo T_d la cual es la que detecta la sonda EGO.
Si tomamos como límites superior $\lambda_{max} = 1,05$, límite inferior $\lambda_{min} = 0,95$,

$Ki = 2s^{-1} \cdot Kp$, y $\lambda(t)$ se toma como la ecuación (4.16), se procede a obtener Ki partiendo de $\lambda = 1$ hasta λ_{max} ó λ_{min}:

$$0,353ms^{-1} \cdot Ki \cdot T_d \leq \lambda_{max} - 1$$

$$0,353ms^{-1} \cdot Ki \cdot 0,42[s] \leq 0,05$$

$$Ki \leq 0{,}338\frac{ms}{s} \tag{4.17}$$

$$Kp = \frac{Ki}{2s^{-1}}$$

$$Kp \leq 0{,}169ms \tag{4.18}$$

Como conclusión del análisis de respuesta temporal del control PI, se destaca que al tomar como parámetro de diseño $Ki \cong 2s^{-1}ms \cdot Kp$, la función $\lambda(t)$ representa un ciclo límite triangular con periodo $4T_d$ con ripple pico-pico de $2 \cdot 0{,}353ms^{-1} \cdot Ki \cdot T_d$. El rango de $\lambda(t)$ es $1 \pm 0{,}353ms^{-1} \cdot Ki \cdot T_d$. Para bajar el modulo del ripple del ciclo límite, se tiene que disminuir Kp y ki manteniendo la relación $Ki = 2s^{-1} \cdot Kp$. Al disminuir estos valores en igual proporción, el ripple disminuye pero la frecuencia del ciclo límite no se ve afectada.

Análisis de plano de fase

A continuación se plantea un análisis de plano de fase del comportamiento del error de $\lambda(t)$ bajo el control PI visto previamente. En primera instancia, el análisis se toma como un sistema sin retardo y al luego se lo incluye.
Error de control:

$$\mathbf{e} = \lambda_{ref} - \lambda$$

$$\mathbf{e} = 1 - \lambda$$

Comportamiento de λ:

$$\lambda = ti_{FF} \cdot Gp' + ti_{FB} \cdot \frac{Gp'}{(T_0 s + 1)}$$

$$\lambda_0 = ti_{FF} \cdot Gp'$$

$$\lambda = \lambda_0 + EGO(\mathbf{e}) \cdot \frac{Ki}{s} \cdot (z_1 s + 1) \cdot \frac{Gp'}{(T_0 s + 1)}$$

Como $z_1 = T_0$, el polo de la planta y el cero del controlador se cancelan:

$$\lambda = \lambda_0 + EGO(\mathbf{e}) \cdot \frac{Ki.Gp'}{s}$$

$$EGO(\mathbf{e}) = \begin{cases} +1 \quad para \quad e < 0 \\ \\ -1 \quad para \quad e > 0 \end{cases} \tag{4.19}$$

Volviendo al error del control y reemplazando λ:

$$\mathbf{e} = 1 - \lambda_0 - EGO(\mathbf{e}) \cdot \frac{Ki.Gp'}{s}$$

$1 - \lambda_0$ es la diferencia entre la consigna y el lambda inicial. Esta diferencia es el error inicial del control e_0 :

$$e_0 = 1 - \lambda_0$$

$$\mathbf{e}.s = e_0.s - EGO(\mathbf{e}) \cdot Ki.Gp'$$

e_0 es una constante por lo tanto la derivada queda:

$$e_0.s = 0$$

$$\dot{\mathbf{e}} = -EGO(\mathbf{e}) \cdot Ki.Gp'$$

$$\dot{\mathbf{e}} = \begin{cases} Ki.Gp' \quad para \quad e < 0 \\ \\ -Ki.Gp' \quad para \quad e > 0 \end{cases} \tag{4.20}$$

El resultado obtenido de $\dot{\mathbf{e}}$ es para un sistema sin retardo. Al tomar en cuenta el retardo nos encontramos que la derivada del error no es única para valores

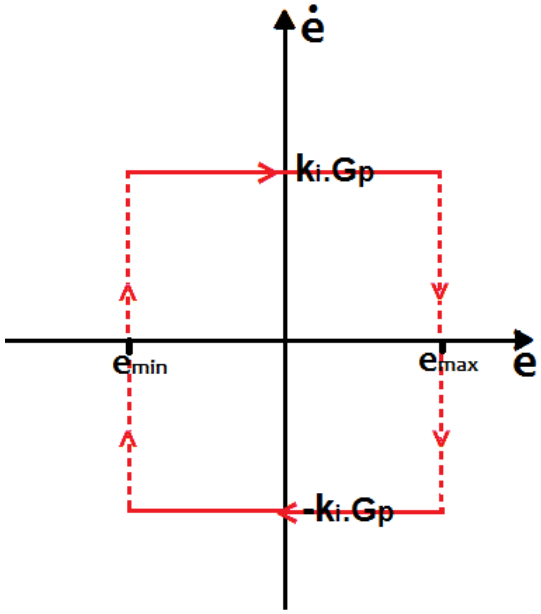

Figura 4.6: Plano de fase PI.

positivos o negativos de e, sino que gira en ese entorno alrededor de e_{max} y e_{min} con valores $\pm Ki.Gp'$ como se muestra en la figura (4.6). Es importante notar que no importa cuál sea el error inicial e_0, el sistema siempre termina en un ciclo límite alrededor de e=0. En otras palabras e_{max} y e_{min} no dependen de e_0 ya que este último se elimina cuando se deriva el error inicial $e_0.s = 0$. Entonces los límites e_{max} y e_{min} sólo dependen de Ki, Gp y del retardo T_d

$$|e_{max}| = |e_{min}| = Ki.Gp'.T_d$$

Análisis de plano de fase con control P

A diferencia del control PI, el control P no posee término integral I, entonces la respuesta temporal $\lambda(t)$ y el plano de fase del control tiene algunas diferencias. Si se parte de la misma forma que el control PI, entonces:

Error de control:

$$\mathbf{e} = \lambda_{ref} - \lambda$$

$$\mathbf{e} = 1 - \lambda$$

Comportamiento de λ:

$$\lambda = ti_{FF} \cdot Gp' + ti_{FB} \cdot \frac{Gp'}{(T_0 s + 1)}$$

$$\lambda_0 = ti_{FF} \cdot Gp'$$

$$\lambda = \lambda_0 + EGO(\mathbf{e}) \cdot Kp \cdot \frac{Gp'}{(T_0 s + 1)}$$

$$EGO(\mathbf{e}) = \begin{cases} +1 \quad para \quad e < 0 \\ \\ -1 \quad para \quad e > 0 \end{cases} \tag{4.21}$$

Volviendo al error del control y reemplazando λ:

$$\mathbf{e} = 1 - \lambda_0 - EGO(\mathbf{e}) \cdot Kp \cdot \frac{Gp'}{(T_0 s + 1)}$$

$1 - \lambda_0$ es la diferencia entre la consigna y el lambda inicial. Esta diferencia es el error inicial del control e_0 :

$$e_0 = 1 - \lambda_0$$

Reemplazando:

$$\mathbf{e} = e_0 - EGO(\mathbf{e}) \cdot Kp \cdot \frac{Gp'}{(T_0 s + 1)} \tag{4.22}$$

$$\mathbf{e} \cdot (T_0 s + 1) = e_0 \cdot (T_0 s + 1) - EGO(\mathbf{e}) \cdot Kp \cdot Gp'$$

$$\mathbf{e}.T_0.s + \mathbf{e} = e_0.T_0.s + e_0 - EGO(\mathbf{e}) \cdot Kp \cdot Gp'$$

$e_0.T_0$ es una constante por lo tanto la derivada queda:

$$e_0.T_0.s = 0$$

Reemplazando y agrupando nuevamente:

$$\dot{\mathbf{e}} = \frac{e_0 - EGO(\mathbf{e}) \cdot Kp \cdot Gp' - e}{T_0}$$

$$\dot{\mathbf{e}} = \begin{cases} \frac{e_0 + Kp \cdot Gp' - e}{T_0} & para \quad e < 0 \\ \\ \frac{e_0 - Kp \cdot Gp' - e}{T_0} & para \quad e > 0 \end{cases} \tag{4.23}$$

A diferencia del control PI, el plano de fase del control P no es una constante, y además depende del error inicial e_0. Pero al igual que en el caso del control PI hay que tomar en cuenta los retardos que inicialmente no fueron tenidos en cuenta.

La figura (4.7) muestra dos posibles casos de ciclo límite de lambda y se muestra que depende necesariamente del error inicial. El lado izquierdo se observa un ciclo límite cuando el error inicial es nulo, y el lado derecho se muestra el ciclo límite cuando el módulo del error es menor a $|Kp.Gp'|$. El error inicial tiene un impacto en cuanto al ciclo de trabajo de la sonda EGO, cuanto mas grande sea el error inicial, mas desequilibrado va a ser el tiempo de lambda que esté entre el estado rico y pobre de la mezcla.

Cuando el módulo del error inicial de lambda sea mayor que $|Kp.Gp'|$, se anula por completo la posibilidad de existencia de un ciclo límite ya que lo único que se logra es reducir el error pero nunca cambiar de estado. La figura (4.8) ilustra de manera cualitativa lo que sucede cuando la corrección de combustible del control P no supera el error inicial cometido por el controlador FF. El plano de fase comienza con un error inicial e_0 desconocido, y termina con un error final e_f desconocido.

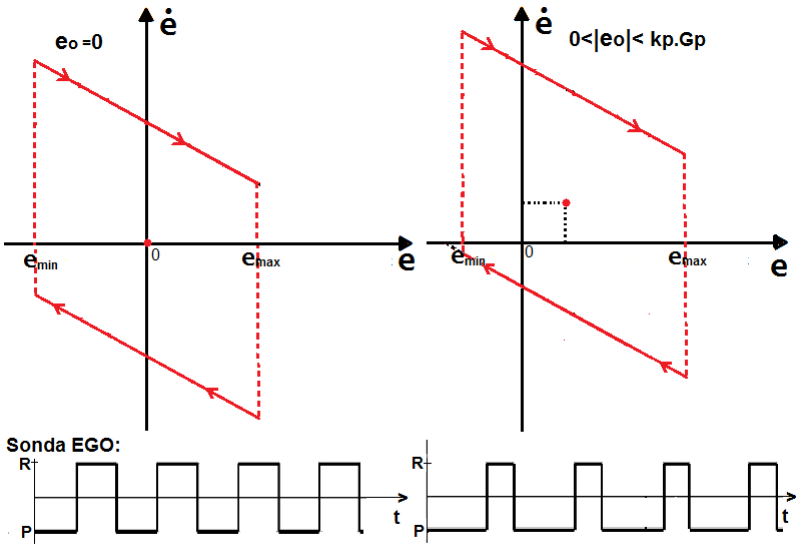

Figura 4.7: Plano de fase P con ciclo límite.

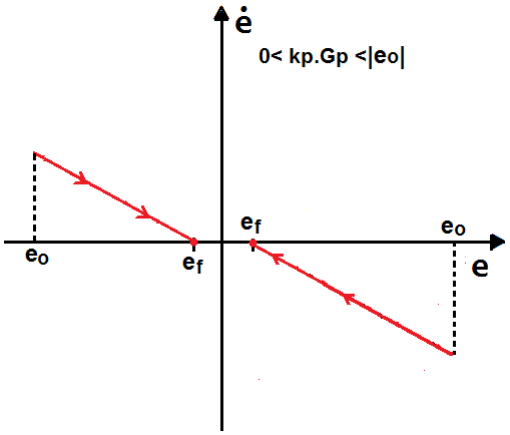

Figura 4.8: Plano de fase P sin ciclo límite.

Análisis de plano de fase con control I

El control I consta de un polo en el origen y no posee un término proporcional, a diferencia del control P o PI, el control I tiene un comienzo mucho más lento. Partiendo del mismo análisis de plano de fase del control P, pero reemplazando el termino Kp por $\frac{Ki}{s}$ en la ecuación (4.22) queda de la siguiente manera:

$$\mathbf{e} = e_0 - EGO(\mathbf{e}) \cdot \frac{Ki}{s} \cdot \frac{Gp'}{(T_0 s + 1)} \tag{4.24}$$

$$\mathbf{e}.T_0.s^2 + \mathbf{e}.s = e_0.T_0.s^2 + e_0 s - EGO(\mathbf{e}) \cdot Ki \cdot Gp'$$

$$\ddot{\mathbf{e}}.T_0 + \dot{\mathbf{e}} = -EGO(\mathbf{e}) \cdot Ki \cdot Gp'$$

$$\ddot{\mathbf{e}} = \frac{\dot{\mathbf{e}} - EGO(\mathbf{e}) \cdot Ki \cdot Gp'}{T_0}$$

Reemplazando:

$$X_1 = \mathbf{e}$$

$$X_2 = \dot{X}_1 = \dot{\mathbf{e}}$$

$$\dot{X}_2 = \ddot{\mathbf{e}}$$

$$\begin{cases} \dot{X}_1 = X_2 \\ \\ \dot{X}_2 = \frac{X_2 - EGO(X_1) \cdot Ki \cdot Gp'}{T_0} \end{cases} \tag{4.25}$$

$$EGO(X_1) = \begin{cases} +1 \quad para \quad X_1 < 0 \\ \\ -1 \quad para \quad X_1 > 0 \end{cases} \tag{4.26}$$

Resolviendo la ecuación diferencial:

$$\frac{\dot{X}_1}{\dot{X}_2} = \frac{X_2.T_0}{X_2 - EGO(X_1) \cdot Ki \cdot Gp'}$$

$$\partial X_1 = \frac{X_2.T_0}{X_2 - EGO(X_1) \cdot Ki \cdot Gp'} \partial X_2$$

$$X_1 = -EGO(X_1).Ki.Gp'.T_0.Ln(|X_2 - EGO(X_1).Ki.Gp'|) - T_0.X_2 \tag{4.27}$$

si $X_1 \geq 0$:

$$X_1 = Ki.Gp'.T_0.Ln(|X_2 + Ki.Gp'|) - T_0.X_2$$

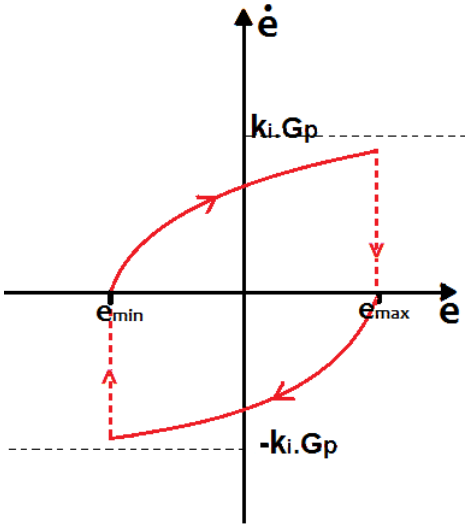

Figura 4.9: Plano de fase I.

si $X_1 \leq 0$:

$$X_1 = -Ki.Gp'.T_0.Ln(|X_2 - Ki.Gp'|) - T_0.X_2$$

Como conclusión de este análisis se puede destacar que, si el controlador FB posee un integrador entonces el error inicial e_0 desaparece del plano de fase y no tiene ninguna relación con el ciclo límite una vez que éste llegue a un régimen estabilizado. En cambio en un controlador que sólo sea proporcional, el ciclo límite queda sujeto al error inicial e_0 y el ciclo de trabajo de la sonda EGO puede quedar desbalanceado hacia el error inicial.

Capítulo 5

CIAA y OSEK RTOS

5.1. Introducción

La implementación de una UEC libre requiere un sistema que sea de **hardware libre** y **robusto**, para que soporte las condiciones hostiles en los ambientes que abunda ruido, vibraciones y temperaturas extremas. Para cumplir con estas condiciones es necesario que tenga implementadas las medidas de **seguridad y protección** y sobre toda las cosas, que cumpla con las **regulaciones**, y **certificaciones de seguridad y calidad**. Estos requerimientos están considerados en la **CIAA** ya que está creada como una plataforma industrial por lo cual dispone de mecanismos de protección eléctrica contra fallas o sobrecargas. De esta forma, brinda una base sólida para diseñar sistemas robustos y confiables de considerar cuestiones de disponibilidad, confiabilidad, verificación, validación y seguridad intrínseca, cumpliendo con las normas internacionales como las IEC 61131 y la IEC 61508. Toda la información recopilada en este Capítulo es citada de la página del Proyecto CIAA [ProyectoCIAA, 2016]. Este Capítulo hace un breve recorrido sobre los aspectos más importante a tener en cuenta a la hora de implementar una UEC sobre la mencionada plataforma.

5.2. Placa CIAA NXP

La **CIAA-NXP** es una computadora que reúne dos cualidades:

1. Ser **Industrial**, ya que su diseño está preparado para las exigencias de confiabilidad, temperatura, vibraciones, ruido electromagnético, tensiones, cortocircuitos, etc., que demandan los productos y procesos industriales.

2. Ser **Abierta**, ya que toda la información sobre su diseño de hardware, firmware, software, etc. está libremente disponible en internet bajo la Licencia

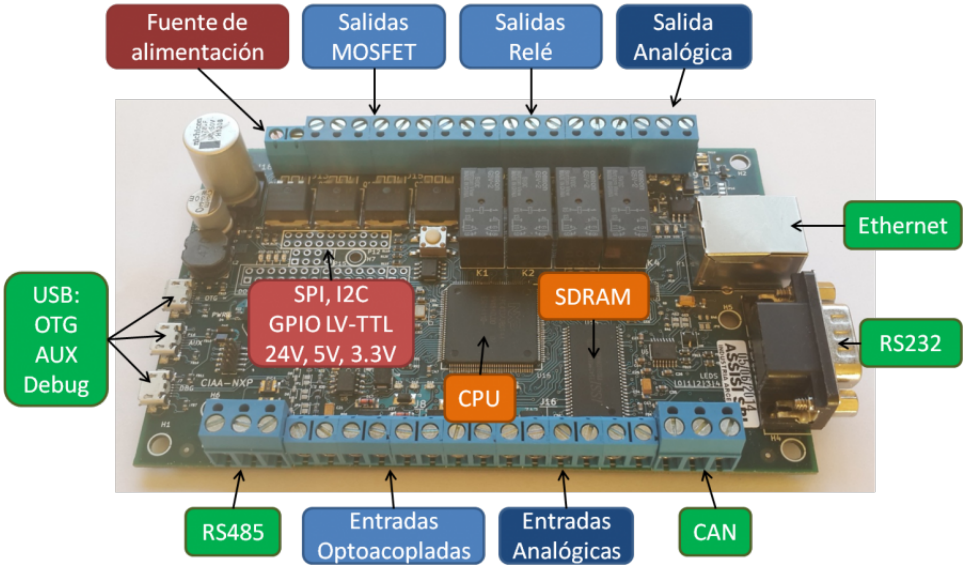

Figura 5.1: CIAA NXP.

BSD, para que cualquiera la utilice como quiera [ProyectoCIAA, 2016].

5.2.1. Características de hardware

El sistema se basa en la placa LPC4337. Cuenta con un microcontrolador ARM Cortex-M4 que incluye un coprocesador ARM Cortex-M0, 1 MB de memoria flash, 136 kB de SRAM, 16 kB de memoria EEPROM, periféricos como el timer de estado configurable (State Configurable Timer) (SCT) y el Serial General Purpose I/O (SGPIO) interface, dos controladores de alta velocidad USB, Ethernet, LCD, un controlador de memoria externa y múltiples entradas/salidas digitales y entradas analógicas. Opera a una frecuencia de reloj de más de 204 MHz.

El ARM Cortex-M4 es la próxima generación de cores de 32 bit que ofrece bajo consumo de energía, mejoras en las características de debug, y alto nivel para soportar integración de bloques. Incorpora un pipeline de 3 etapas, arquitectura Harvard con buses separados para datos e instrucciones y un tercer bus para periféricos. Incluye una unidad interna de prebúsqueda que soporta ejecución especulativa en bifurcaciones. Soporta procesamiento de señales en un ciclo e instrucciones SIMD (Single Instruction Múltiple Data). También, tiene integrado en el core hardware para procesamiento en punto flotante.

El coprocesador ARM Cortex-M0 es de uso eficiente de energía, de 32 bit compatible en código y herramientas con el Cortex-M4. El coprocesador Cortex-M0,

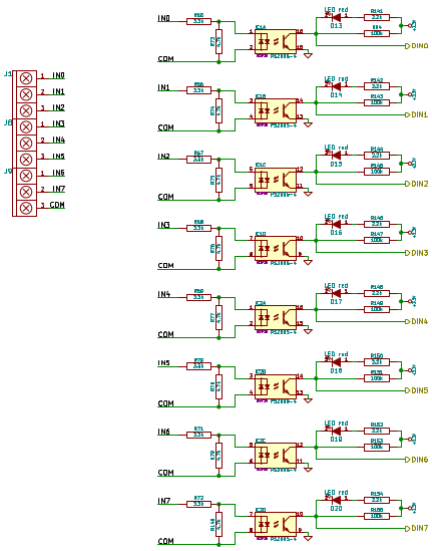

Figura 5.2: Entradas digitales.

está diseñado para reemplazar los microcontroladores existentes de 8/16-bit.

Entradas digitales

La CIAA pose 8 entradas digitales optoacopladas (PS2805-4), de este modo se combinan en un solo dispositivo semiconductor, un fotoemisor y un fotorreceptor cuya conexión entre ambos es óptica.

Entradas Analógicas

Debido a la gran cantidad de sensores analógicos utilizados en la industria y en la electrónica en general, la CIAA ha incorporado 4 entradas analógicas configurables. Estas tienen la capacidad de poder funcionar en lazo de corriente (4-20mA) o en tensión (0-10V) mediante la selección por medio de jumpers. Esta versatilidad convierte a las entradas analógicas de la CIAA en uno de sus fuertes para los procesos de automatización.
El circuito ha sido diseñado teniendo en cuenta las variaciones de temperatura, corriente inversa de las protecciones, tensión de offset del AO, etc. Por esto, tiene componentes de alta precisión y baja variación con la temperatura, minimizando la incertidumbre asociada a la medición. Características:

- 4 Entradas analógicas configurables por Corriente/Tensión.

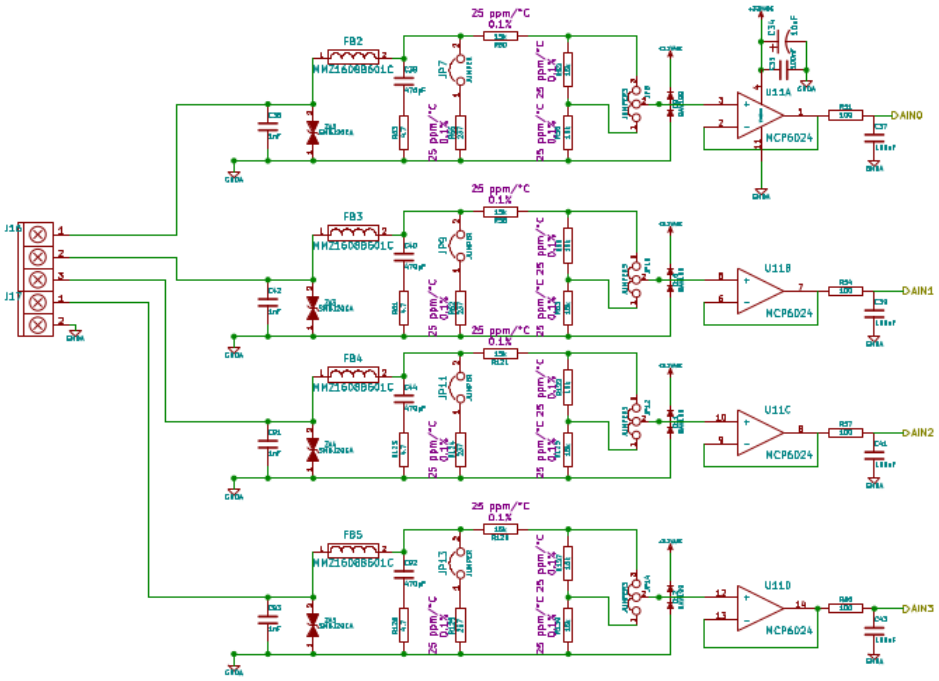

Figura 5.3: Entradas analógicas.

- Lazo de corriente 4-20mA (con rango extendido 0-22mA), impedancia de carga 237 Ohm.
- Control por tensión 0-10V, impedancia de entrada 45 KOhm.
- Protección contra transitorios.
- Protección por filtrado de alta frecuencia.
- Protección por diodos de enclavamiento.
- Amplificador-Buffer, estable y Rail-to-Rail

Salidas digitales de potencia

La CIAA-NXP posee 4 salidas digitales open-drain. Esto significa que en lugar de emitir una señal de una tensión o corriente específica, la señal de salida se aplica al GATE de un MOSFET interno cuyo DRAIN se encuentra abierto y está a disposición para su conexión.
En la figura (5.4) se muestra el diagrama esquemático de las salidas open-drain de los MOSFET FQT13N06L (60V LOGIC N-Channel MOSFET) que contiene la placa.

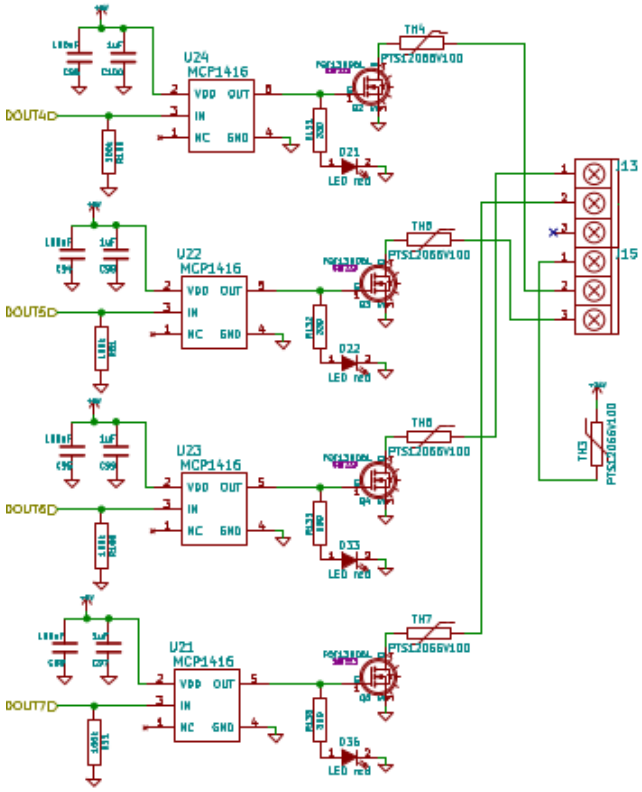

Figura 5.4: Salidas digitales open-drain.

Entre el microcontrolador y los MOSFET se encuentra un adaptador de nivel de tensión MCP1416 (high-speed power MOSFET driver) y sirve para poder adaptar los 3.3 V de salida del microcontrolador a los 5 V del MOSFET.

Características del MOSFET:

- Corriente: 2.8 A.
- Tensión drain-source: 60 V.
- $R_{DS(on)}$: 0,11 Ω

5.3. OSEK

OSEK-VDX es un comité de estandarización creado en 1994 por las automotrices europeas. OSEKVDX incluye varios estándares que se utilizan en la industria automotriz, entre ellos los más relevantes:

- OSEK OS
- OSEK COM
- OSEK NM
- OSEK Implementation Language (OIL)
- OSEK RTI
- OSEK Time Trigger Operating System

El estándar fue creado principalmente para poder reutilizar el SW de un proyecto a otro gracias a la definición de una interfaz estandarizada. Además al proveer un estándar se da la posibilidad a nuevas empresas de proveer SW compatible con el estándar y permitir la implementación de diversos sistemas OSEK compatibles. Esto último permite a la industria automotriz la posibilidad de elegir el proveedor dentro de una lista mayor de proveedores de SW ya que cualquiera puede implementar un OSEK-OS u otro estándar especificado por OSEK-VDX [Cerdeiro, 2015].

5.3.1. Sistema operativo ESTATICO vs DINAMICO

OSEK-OS, a diferencia de otros sistemas operativos como Linux y Windows, es un sistema operativo estático. Esto significa que las tareas, sus prioridades, cantidad de memoria que utilizan, etc. son definidos antes de compilar el código, en un proceso que se llama generación. En OSEK-VDX no es posible crear una tarea de forma dinámica, no es posible cambiar la prioridad a una tareas como estamos acostumbrados en Windows y Linux. Por ende, OSEKVDX sería un sistema impensable para una computadora o un celular donde constantemente estamos cargando nuevos programas, corriéndolos, cerrándolos etc. OSEK-OS está pensado para un sistema embebido que debe realizar una tarea específica en tiempo real y donde no se necesita cargar nuevas tareas de forma dinámica. El ser un sistema operativo estático trae grandes ventajas a su comportamiento en tiempo real. El sistema se comporta de forma totalmente determinística. No existe la posibilidad de que una tarea no pueda ser cargada por que no hay más memoria disponible como podría pasar en Linux/Windows. Las tareas tienen una prioridad asignada de ante mano, por ende, una tarea de alta prioridad debida a que realiza un control crítico de seguridad tendrá siempre esa misma prioridad. Esto es sobre todo importante en sistemas de control críticos con requerimientos SIL[1]. Sobre todo para

[1] Safety Integrity Level (Nivel de Integridad de Seguridad): Nivel relativo de reducción del riesgo que provee una función de seguridad, o bien para especificar el nivel objetivo para la reducción de riesgo.

el sistemas en donde los fallos no son aceptables o tienen un costo demasiado alto [Medina et al., 2016].

OSEK-OS puede optimizar los recursos gracias a que se trata de un sistema estático es su Scheduler. En la configuración del scheduler se indican distintos parámetros como la prioridad, tipo de scheduling, tamaño de la pila, cantidad de activaciones así como otros. Gracias a esto el Scheduler del sistema operativo podrá adaptarse a las necesidades del usuario. Conociendo la cantidad de tareas y sus prioridades se podrán generar las FIFOs para el scheduling. El Scheduler de OSEK utiliza una FIFO por prioridad. El largo de cada FIFO será la suma de las tareas con una prioridad determinada.

5.3.2. Tareas

A diferencia de otros sistemas operativos donde las tareas corren por un tiempo indeterminado hasta ser terminadas, en OSEK-VDX y por lo general en sistemas de tiempo real las taras realizan su cometido y terminan. No se utilizan estructuras como while(1) para mantener la tarea corriendo, ni sleeps para dormir entre activaciones. Sino que, la tarea se inicial, realiza su cometido y termina. Ya sea leer la temperatura de un sensor, recibir un paquete de comunicación, procesar datos de audio, sin importar lo que sea: se comienza, se procesa y se termina.

Para el control de tareas OSEK-OS ofrece las siguientes interfaces:

- **ActivateTask**: activa una tarea.
- **ChainTask**: realiza la combinación de ActivateTask seguido de TerminateTask .
- **TerminateTask**: termina una tarea.

Estado de las tareas

Cada tarea en OSEK-VDX se encuentra siempre en uno de los siguientes 4 estados:

- **running**: la tarea se encuentra corriendo, está utilizando los recursos del procesador en este mismo momento. En cada momento una única tarea puede encontrarse en este estado.
- **ready**: en este estado están todas las tareas que se encuentran esperando los recursos del procesador para poder correr. No se encuentran en el estado running porque una tarea de mayor prioridad se encuentra corriendo.

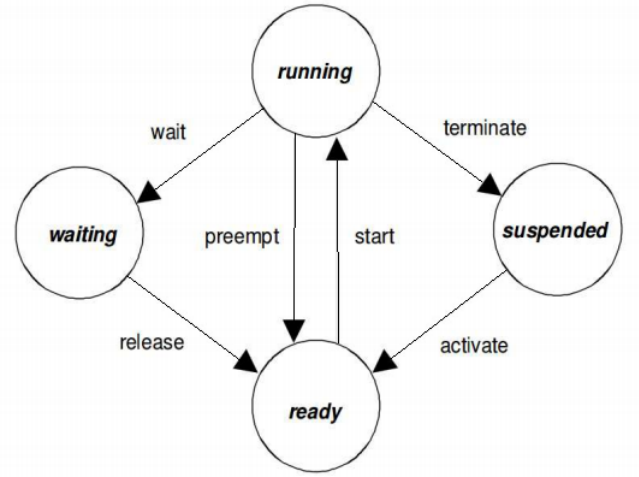

Figura 5.5: Estados de las tareas.

- **waiting**: la tarea se encontraba corriendo y decidió esperar la ocurrencia de un evento, hasta que el evento que espera ocurra la misma se encontrará en el estado waiting.
- **suspended**: es el estado por defecto de las tareas, la tarea esta momentáneamente desactivada.

En la figura (5.5) y (5.6) se muestra un diagrama de los posibles estados en los que se encuentre una tarea del sistema operativo junto con sus transiciones.

En los casos en donde una tarea realmente debe esperar y no puede terminar OSEK-OS provee la utilización de eventos a través del llamado de la interfaz WAITEVENT. Debido a esto, la tareas se pueden clasificar en dos grupos:

1. BASIC: son aquellas tareas que no tengan eventos y por ende carezcan del estado waiting.
2. EXTENDED: son aquellas tareas que tienen eventos y por ende pueden esperar hasta que uno o más eventos ocurran.

Por otro lado en OSEK-OS existen tareas que pueden ser interrumpidas (SCHEDULE = FULL) y otras que no (SCHEDULE = NON).

1. NON PREEMPTIVE: son tareas que no interrumpidas por aquellas de mayor prioridad, salvo que la misma tarea llame a Schedule, pase al estado waiting llamando a WaitEvent o terminen.
2. PREEMTIVE: son tareas que pueden ser interrumpida en cualquier momento cuando se encuentre una tarea de mayor prioridad en la lista ready.

Transición	Estado anterior	Estado futuro	Descripción
activate	suspended	ready	Una nueva tarea es activada y puesta en la lista de ready para correr. Esta transición se puede realizar por ejemplo con las siguientes interfaces: • ActivateTask • ChainTask
start	ready	running	Una tarea es llevada al estado running de forma automática cuando es la tarea de mayor prioridad en la lista de *ready*.
wait	running	waiting	La tarea es llevada a este estado para esperar la ocurrencia de un evento. Esto se puede lograr con la siguiente interfaz: • WaitEvent
realease	waiting	ready	Al ocurrir el evento que una tarea esperaba la misma es llevada de nuevo al estado *ready*. Esto se puede realizar con la siguiente interfaz: • SetEvent
preempt	running	ready	Una tarea que estaba corriendo es desactivada, esto ocurre cuando una tarea de mayor prioridad se encuentra en la lista de *ready* y la tarea actual tiene una scheduling police FULL o llama a la siguiente interfaz: • Schedule
terminate	running	suspended	La tarea termina su ejecución, esto lo puede llevar a cabo con las siguientes interfaces: • ChainTask • TerminateTask

Figura 5.6: Descripción de transiciones.

Capítulo 6

Implementación de UEC sobre CIAA

6.1. Introducción

En los capítulos anteriores se hizo una explicación detallada tanto del funcionamiento de un MCI como de las principales características de la CIAA. Este capítulo pretende hacer una unión entre ambas partes, partiendo de las salidas y entradas de un sistema MCI (sensores de variables físicas e inyectores como actuadores) con las entradas y salidas de la CIAA, por medio del sistema operativo OSEK a ejecutar para lograr el objetivo como implementación de UEC de inyección de combustible.

6.2. Acondicionamiento de señal

En general, a una señal de salida de un sensor de un sistema de medición se la debe procesar de forma adecuada para la siguiente etapa de operación. Por ejemplo, la señal puede ser demasiado pequeña, y en ese caso es necesario amplificarla; podría contener interferencias que se deben eliminar; ser no lineal y requerir su linealización; ser analógica y requerir su digitalización; ser digital y requerir su conversión en analógica; ser un cambio en el valor de la resistencia, y convertirla a un cambio en corriente o tensión; consistir en un cambio de voltaje y convertirla en un cambio de corriente de magnitud adecuada, etcétera.

Adecuación de señal del sensor de rotación

Como hemos visto en el Capítulo 2, la señal de un sensor inductivo (pickup diente o CKP) se utiliza para medir **RPM** y para detectar la posición de los pis-

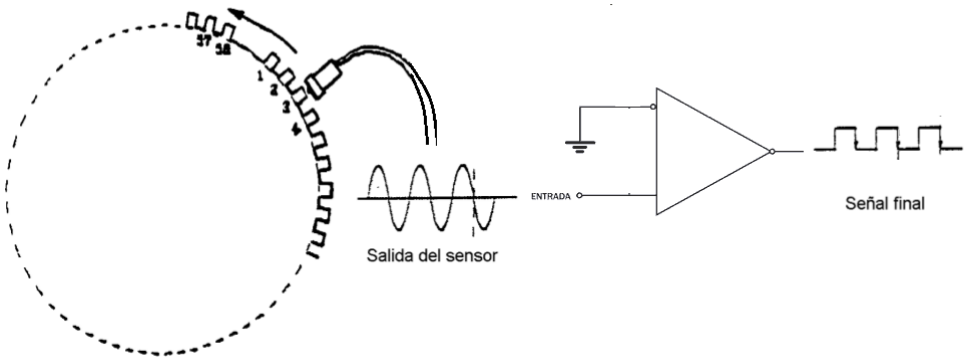

Figura 6.1: Adecuador por comparador.

tones dentro de los cilindros, y así poder sincronizar la inyección en el momento correcto. Esta señal analógica es necesario transformarla en una señal cuadrada para luego utilizarla como entrada digital de interrupción por flanco.

Para trasformar la señal del sensor CKP en una señal cuadrada se utiliza un comparador de cruce por cero con realimentación positiva para generar histeresis, también denominado Schmitt Trigger, y evitar que el ruido superpuesto a la señal original causara falsos cambios de estado y arruinara la medición. la parte superior de la figura (6.2) muestra un circuito esquemático de un comparador con histéresis.

Adecuación de señal EGO

La señal de un sonda EGO puede utilizarse a través de una entrada de ADC y ser analizada según su rango de valores (200 mV a 800 mV) o, ya que es una señal de dos estados, puede ser usada como entrada digital luego de haber pasado por un comparador de tensión. A diferencia del comparador de cruce por cero que se utiliza para adecuar la señal CKP, el comparador para la señal EGO necesita una tensión de referencia que se ubique en el valor medio entre 200 mV y los 800 mV de la tensión de salida de la sonda.

$$Vref = \frac{200 + 800}{2} = 500mv \tag{6.1}$$

Las principales desventajas de usar la sonda EGO como entrada digital son:

- La ECU no podrá detectar una falla (desconexión, o rotura de sonda) debido a que la salida del comparador toma sólo dos valores, y en cambio la señal analógica posee un rango de valores que puede ser analizado.

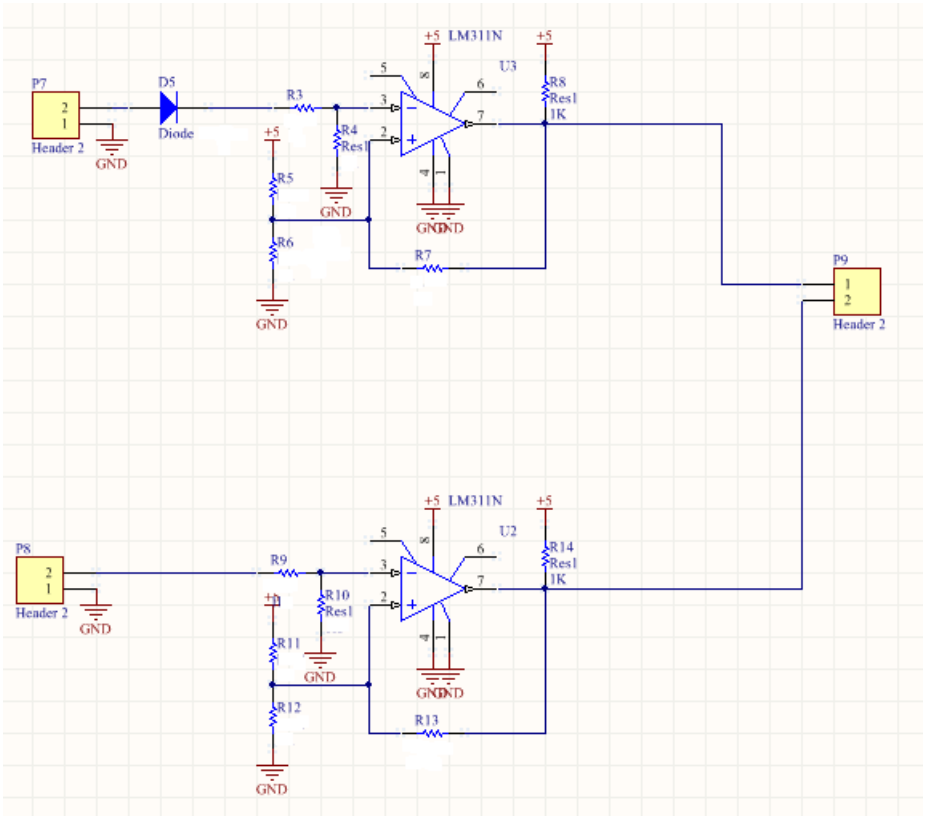

Figura 6.2: Arriba: Adecuador señal CKP.
Abajo: Adecuador señal EGO.

- Si bien la sonda se considera de dos estados, igual posee un comportamiento dinámico lento para pasar de un estado a otro por lo tanto hay presente un importante retardo desde que comienza el cambio de estado hasta que pasa por Vref.

A continuación se enumeran todos los componentes utilizados para los adecuadores de señal:

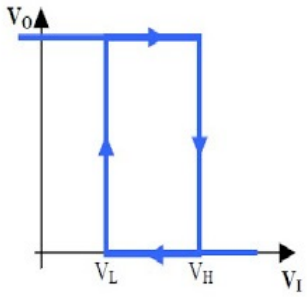

Figura 6.3: Histéresis.

Nombre	Valor
D5	1n5819
R3	39 Ω
R4	10 $K\Omega$
R5	100 $K\Omega$
R6	2,2 $K\Omega$
R7	68 $K\Omega$
R8	1 $K\Omega$
R9	100 Ω
R10	100 $K\Omega$
R11	10 $K\Omega$
R12	1 $K\Omega$
R13	100 $K\Omega$
R14	1 $K\Omega$
A.O.	LM 311 N

Comportamiento de los comparadores

Como se mencionó previamente, los comparadores son circuitos no lineales que sirven para comparar dos señales, una de las cuales generalmente es una tensión de referencia, y así determinar si la tensión de entrada es mayor o menor.
A continuación se hace un análisis de las configuraciones de las mallas resistivas de la entrada de tensión de referencia, y de este modo poder determinar el nivel de señal de referencia junto con el rango de histéresis (VL y VH) que se muestran en la figura 6.3. La configuración depende sólo del estado de la salida del comparador.

Comparador de señal CKP:

En primer lugar, para obtener los valores de VH y VL de la histéresis del ade-

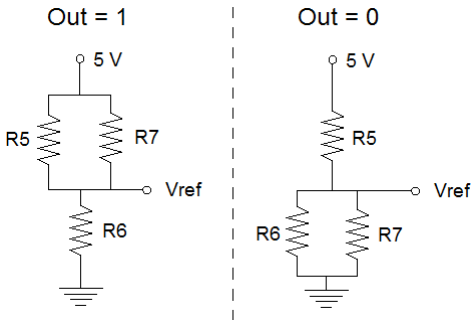

Figura 6.4: Malla resistiva según el estado de salida del comparador de señal CKP.

cuador de señal CKP hay que separar el análisis dependiendo del estado de la salida del comparador. En la figura (6.4) se muestra cómo se configura la malla resistiva que genera la tensión de referencia (Vref) dependiendo de la salida del comparador.

$$Out = 1 \Rightarrow Vref = VH$$
$$VH = \frac{R6}{R6 + R5//R7} \cdot 5V$$
$$VH = 258mV$$

$$Out = 0 \Rightarrow Vref = VL$$
$$VL = \frac{R6//R7}{R6//R7 + R5} \cdot 5V$$
$$VL = 104mV$$

Comparador de señal de sonda lambda:

El análisis para obtener los valores de VH y VL del adecuador de señal de sonda lambda es el mismo que se hizo para el adecuador de señal CKP.

$$Out = 1 \Rightarrow Vref = VH$$
$$VH = \frac{R12}{R12 + R11//R13} \cdot 5V$$
$$VH = 496mV$$

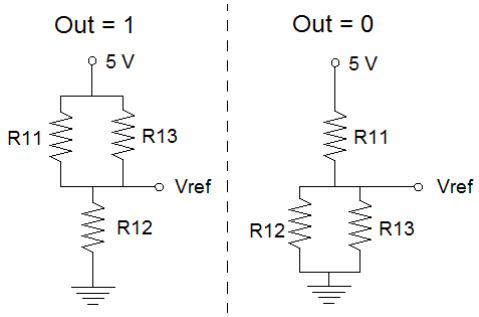

Figura 6.5: Malla resistiva según el estado de salida del comparador de sonda lambda.

$$Out = 0 \Rightarrow Vref = VL$$

$$VL = \frac{R12//R13}{R12//R13 + R11} \cdot 5V$$

$$VL = 450mV$$

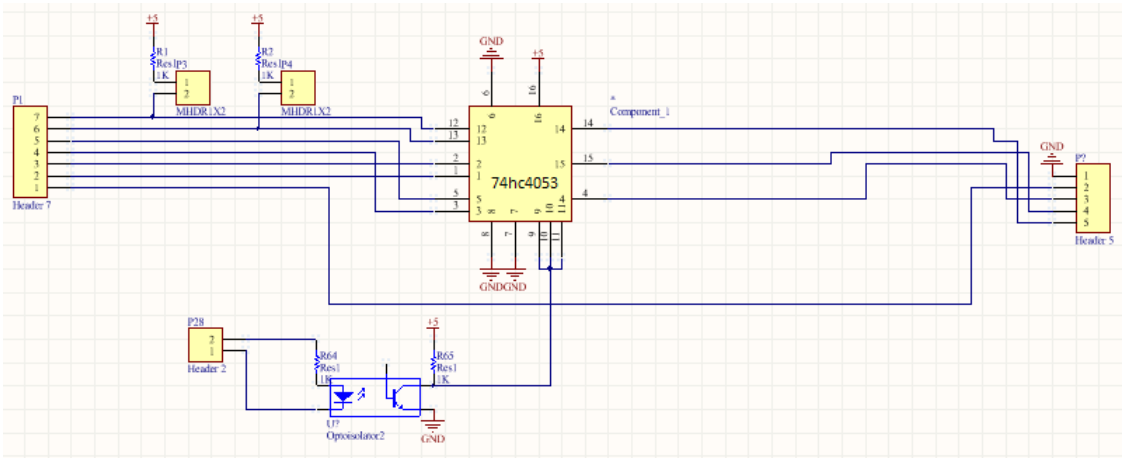

Figura 6.6: Multiplexor de entradas analógicas.

Señales analógicas

En el Capítulo 3 se nombraron todos los sensores que intervienen en el cálculo del control de inyección. A continuación, se enumeran nuevamente detallando los rangos de tensión y cantidad, de forma de saber cuantas entradas analógicas se necesitan.

Señal	Cantidad	Rango de tension
Presión de múltiple de admisión	1	0 a 5 V
Temperatura de aire de admisión	1	0 a 5 V
Temperatura de líquido refrigerante	1	0 a 5 V
Posición de mariposa	2	0 a 5 V
Posición del pedal de acelerador	2	0 a 5 V
Total:	7	

El número total de señales necesarias por el sistema, que deben ser digitalizadas es 7 en este caso, y esto presenta un problema si uno tiene en cuenta que la CIAA posee solo 4 canales de entrada analógica. Por esta razón se ve la necesidad de expandir la cantidad de entradas mediante un multiplexor. Se opta por el multiplexor 74hc4053 (Triple 2-channel analog multiplexer/demultiplexer) que posee 3 canales multiplexados en 2 canales cada uno, y que ingresan a los canales de entrada CH0, CH1 y CH2 de la CIAA, y dejando CH3 como entrada directa de la señal de presión del múltiple de admisión.

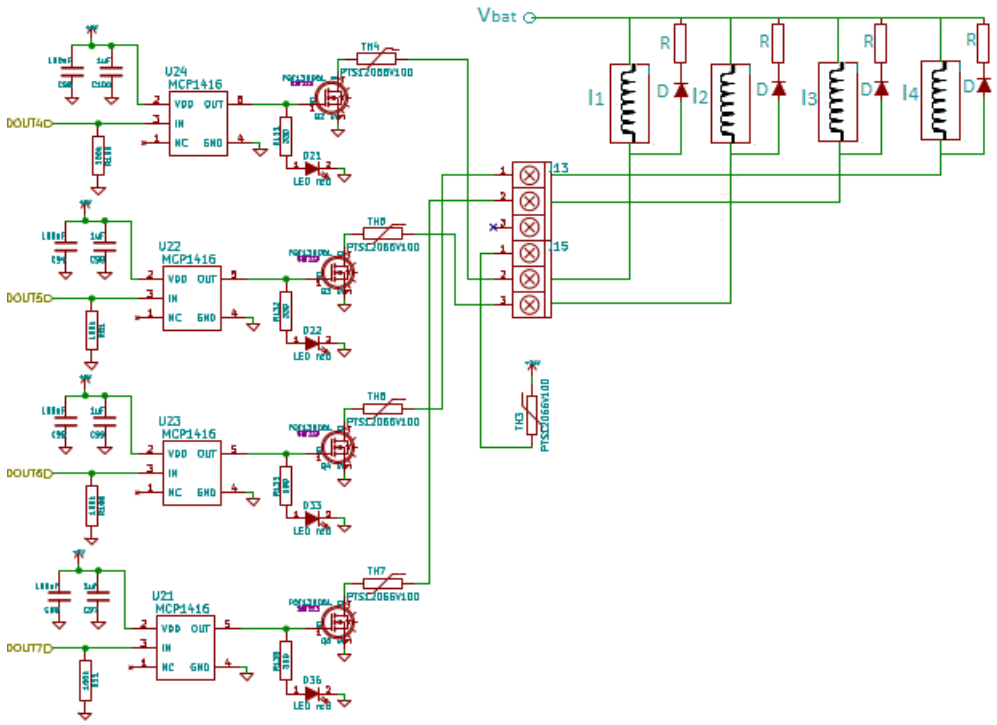

Figura 6.7: Conexión de los Inyectores.

Conexión de inyectores

Dentro de las características del HW de la CIAA, se nombró que posee 4 salidas digitales open-drain compuestas por MOSFET de canal N (FQT13N06L) que pueden soportar hasta 2.5 A de corriente de drain (Corriente de inyector $\approx 1A$) y hasta 60 V de tensión drain-source. Estas características son suficientes para que los MOSFET puedan ser utilizados para activar los inyectores y así controlar el flujo de combustible. La figura (6.7) muestra la conexión esquemática de los inyectores al DRAIN de los MOSFET.
Cabe destacar que es necesario conectar en paralelo a los inyectores un diodo en serie con una resistencia. Esta conexión cumple la función de desenergizar la bobina del inyector cuando el MOSFET pasa al estado off, y así reducir al mínimo el tiempo que tarda en cerrarse el inyector.

6.2.1. Interfaz gráfica o HMI

El principal uso de una interfaz gráfica, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina, en este caso la comunicación con la CIAA. Es importante contar con

Figura 6.8: Interfaz gráfica.

una HMI para poder leer e interpretar las variables físicas del motor como RPM, temperatura, tiempo de inyección (total, FF, FB), lectura de sonda (mezcla rica o mezcla pobre), etc y además poder modificar parámetros de entrada tales como las constantes del controlador PI o modificar de forma manual las salidas de la CIAA. El diseño de la HMI se desarrolló en lenguaje visual de linux llamado **GAMBAS**. Gambas es un lenguaje de programación libre derivado de BASIC (de ahí que Gambas quiere decir Gambas Almost Means Basic). Se distribuye con licencia GNU GPL. Cabe destacar que presenta ciertas similitudes con Java ya que en la ejecución de cualquier aplicación, se requiere un conjunto de librerías intérprete previamente instaladas (Gambas Runtime) que entiendan el bytecode de las aplicaciones desarrolladas y lo conviertan en código ejecutable por el computador.
La interfaz diseñada no sólo permite mostrar al usuario los puntos de trabajo del motor (RPM, tempereatura, presión, tiempo de inyección) sino que también permite modificar y ajustar los valores de Kp y Ki, y además permite un control manual del caudal de combustible para realizar pruebas y mejorar el comportamiento del mismo sistema.

6.2.2. Descripción de las tareas

A continuación se muestra una tabla con los nombres de las tareas, sus prioridades y una descripción general de la función que cumple y como se activa.

Tarea	Prioridad	Descripción
InitTask	2	Primer tarea que se ejecuta. Inicializa los dispositivos de la CIAA (ADC, entradas digitales, etc.)
SerialEchoTask	1	Queda a la espera de entrada de algún carácter del bus serie. Activada desde el inicio.
WriteTask	2	Envía datos al bus serie. Se activa periódicamente si el usuario lo solicita.
SensoresTask	7	Lee y obtiene los valores de los sensores analógicos utilizando ADC. Se activa periódicamente.
ContadorTask	10	Sincroniza la CIAA con el motor. Calcula la posición de los pistones y las RPM. Se activa por interrupción generada por el sensor CKP.
AvanceInyeccTask	10	Calcula el momento en el que la inyección debe comenzar en cada uno de los cilindros. Se activa por ContadorTask.
Inyeccion_1Task	9	Activa y desactiva el inyector $N°1$ Se activa por AvanceInyeccTask y se desactiva por timer0.
Inyeccion_2Task	9	Activa y desactiva el inyector $N°2$ Se activa por AvanceInyeccTask y se desactiva por timer1.
Inyeccion_3Task	9	Activa y desactiva el inyector $N°3$ Se activa por AvanceInyeccTask y se desactiva por timer2.
Inyeccion_4Task	9	Activa y desactiva el inyector $N°4$ Se activa por AvanceInyeccTask y se desactiva por timer3.
Controlador_FeedforwardTask	5	Calcula de forma anticipativa el tiempo de inyección. Se activa periódicamente por alarma
Controlador_FeedbackTask	5	Corrige el tiempo de inyección en base al estado de la sonda lambda Se activa periódicamente siempre y cuando la tarea Controlador_FeedforwardTask no requiera modificar el tiempo de inyección.

Capítulo 7

Resultados experimentales

En este Capítulo se muestran las mediciones realizadas sobre la señal de la sonda EGO bajo diferentes configuraciones de Kp y Ki, y un posterior análisis. El osciloscopio fue configurado en 200 mV/div y una base de tiempo de 5 s/div.

7.1. Medición de sonda EGO bajo el control de ECU SIM 32

La medición que se muestra en la figura (7.1) corresponde a la señal de la sonda lambda cuando la inyección se encuentra bajo el control de la ECU que viene de fábrica (marca **siemens** modelo **sim 32**). En esta medición se determina que el tiempo en el que la mezcla permanece en estado rico es casi el mismo tiempo en el que la mezcla permanece en estado pobre.
El objetivo de esta medición es obtener una muestra del comportamiento de la mezcla, y así tener un parámetro para comparar con el sistema de control adoptado.

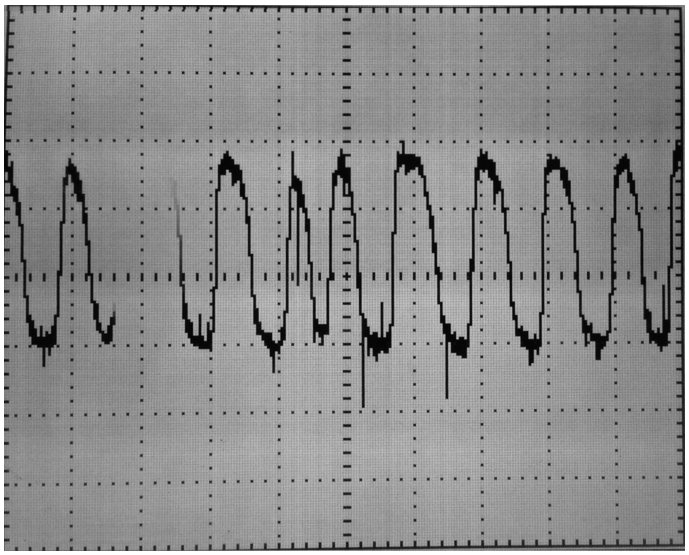

Figura 7.1: Medición de sonda EGO.

Mezcla	Tiempo mínimo	Tiempo máximo	Promedio
Rica	1,5 seg	3 seg	2,31 seg
Pobre	1,5 seg	3 seg	2,28 seg

7.2. Medición de sonda EGO bajo el control de CIAA

En esta sección se muestran las pruebas del control de inyección desarrollado en los capítulos anteriores. Todos estos ensayos se hacen implementando distintas constantes de control Kp y Ki, y lo que se busca es determinar las constantes más adecuadas desde el punto de vista tanto del control de dosado, como del correcto funcionamiento del motor. Hay que recordar que un mal funcionamiento del control de mezcla no solo perjudica el rendimiento del catalizador, sino que también puede generar importantes variaciones de potencia ocasionando un comportamiento irregular o inestable del régimen de giro del motor.

En cada uno de los ensayos se pueden determinar los ciclos en los que la mezcla permanece en estado rico o en estado pobre, y así poder determinar los tiempos de cada estado y si se llega a un equilibrio entra cada uno.

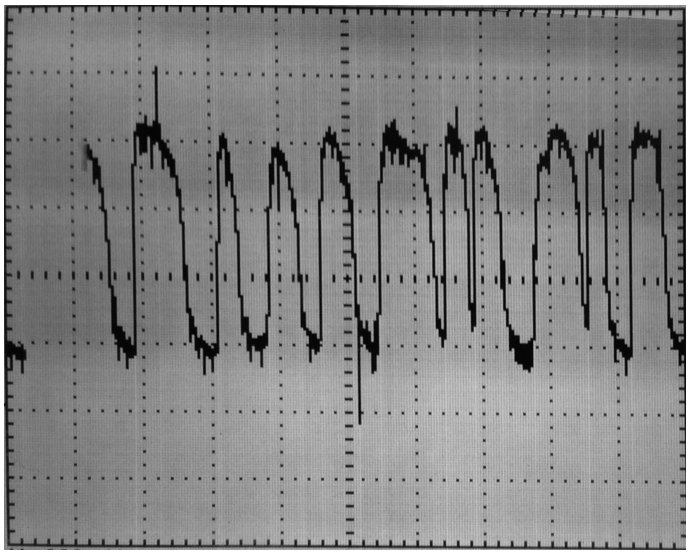

Figura 7.2: Medición de sonda EGO.

Mezcla	Tiempo mínimo	Tiempo máximo	Promedio
Rica	1,2 seg	4 seg	2,5 seg
Pobre	0,5 seg	2,5 seg	1,56 seg

Kp :	0,1
Ki :	0,1

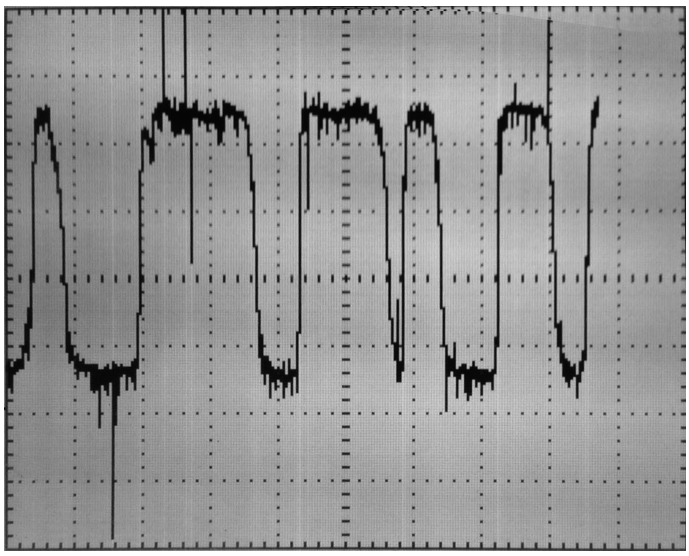

Figura 7.3: Medición de sonda EGO.

Mezcla	Tiempo mínimo	Tiempo máximo	Promedio
Rica	2,1 seg	8,1 seg	4,9 seg
Pobre	1 seg	5,5 seg	3,26 seg

Kp :	0,1
Ki :	0,02

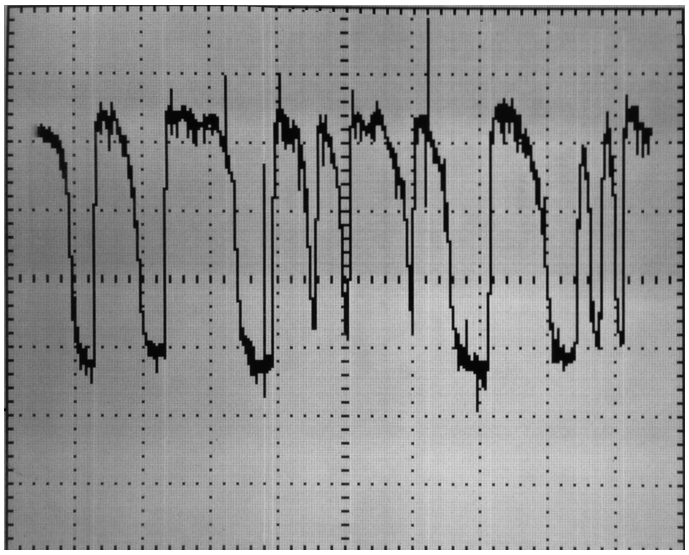

Figura 7.4: Medición de sonda EGO.

Mezcla	Tiempo mínimo	Tiempo máximo	Promedio
Rica	1 seg	5,2 seg	2,77 seg
Pobre	0,5 seg	3 seg	1,45 seg

Kp :	0,5
Ki :	0,11

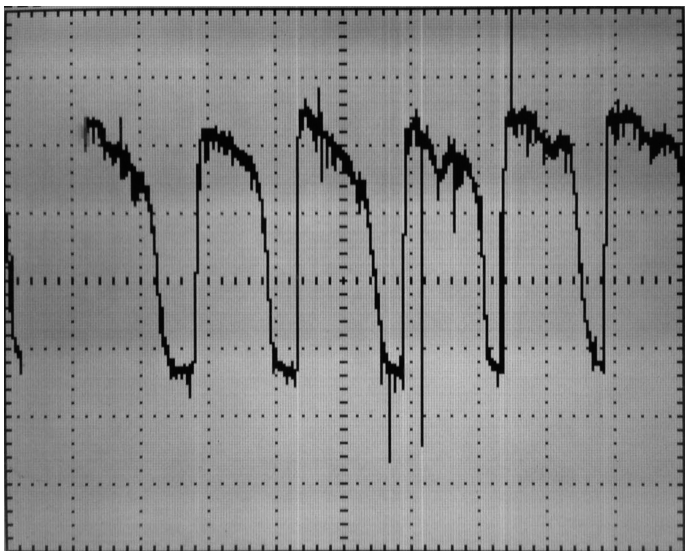

Figura 7.5: Medición de sonda EGO.

Mezcla	Tiempo mínimo	Tiempo máximo	Promedio
Rica	5,1 seg	5,7 seg	5,45 seg
Pobre	1,5 seg	3 seg	2,52 seg

Kp :	0,05
Ki :	0,15

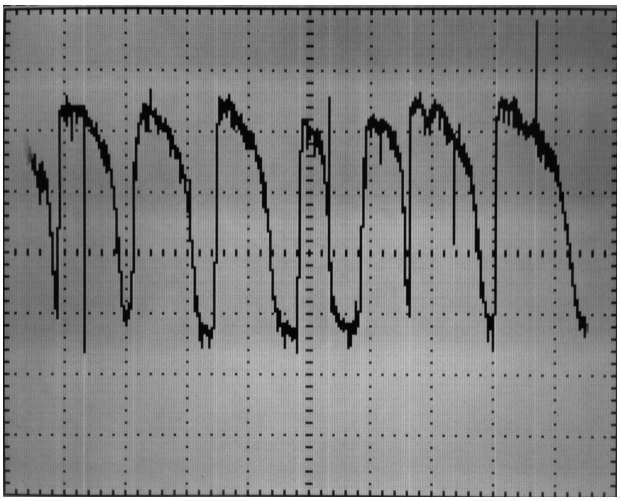

Figura 7.6: Medición de sonda EGO.

Mezcla	Tiempo mínimo	Tiempo máximo	Promedio
Rica	2,5 seg	5,7 seg	4,22 seg
Pobre	0,5 seg	2,3 seg	1,58 seg

Kp	Ki
0,05	0,1

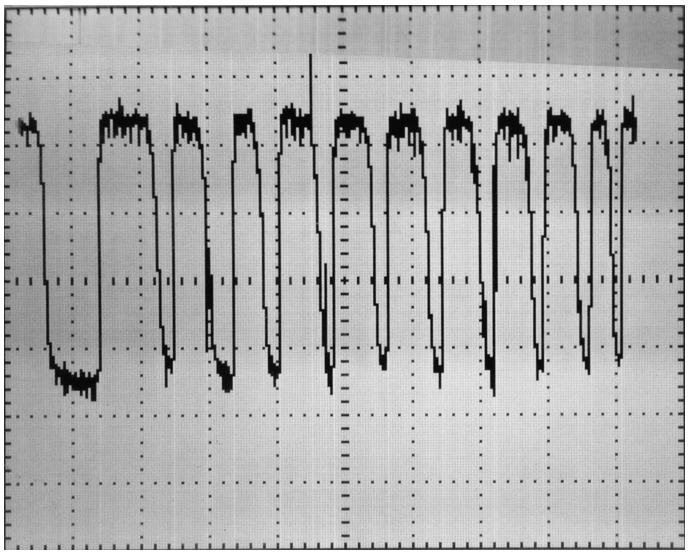

Figura 7.7: Medición de sonda EGO.

Mezcla	Tiempo mínimo	Tiempo máximo	Promedio
Rica	2 seg	4,1 seg	2,9
Pobre	1 seg	4 seg	1,37 seg

Kp	Ki
0	0,1

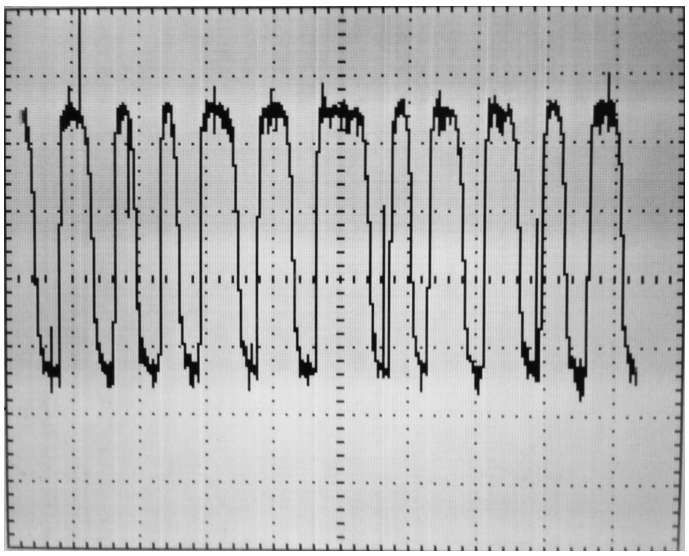

Figura 7.8: Medición de sonda EGO.

Mezcla	Tiempo mínimo	Tiempo máximo	Promedio
Rica	2 seg	2,8 seg	2,37 seg
Pobre	1,5 seg	2 seg	1,62 seg

Kp	Ki
0	0,02

En cada uno de los ensayos se puede observar como es el comportamiento de la mezcla, y el transitorio del control hasta que queda oscilando dentro de un entorno estable o hasta que llega al punto en el que la cantidad de combustible sea insuficiente para mantener el régimen de giro.
La insuficiencia de combustible tiene consecuencias dependiendo si esta es leve o grave. En el caso que sea leve sólo se produce variaciones no muy significativas en el régimen de giro generando fluctuaciones en la presión de admisión, estas fluctuaciones de la presión modifican directamente la salida del controlador FF y todo el ciclo de corrección de inyección vuelva a comenzar. Si la falta de combustible es grave se podría llegar al punto en el cual la energía liberada en la combustión sea insuficiente para mantener el motor funcionando (caso menos deseado).
En base a los distintos ensayos se puede observar que los resultados más parecidos entre la UEC desarrollada y la UEC original se logran con valores de $Kp = 0$ y $Ki \leq 0{,}1$.

Capítulo 8

Conclusiones y trabajo futuro

8.1. Conclusiones

El desarrollo de una UEC de inyección comienza con la comprensión del funcionamiento de un motor de combustión interna. Esta comprensión no solo involucra el funcionamiento mecánico, sino que hay que relacionarlo con un modelo matemático que lo describa en todo aspecto.
Otro punto que ha tener en cuenta en el desarrollo es la importancia de mantener una relación estequiométrica en la mezcla, ya que está relacionado directamente con el consumo y el rendimiento del combustible. Por otro lado está la importancia del impacto ambiental que ésto genera y las leyes que lo regula.

Para la implementación del sistema se optó por utilizar un control del tipo feedforward + feedback. El control feedforward es un control anticipativo que sólo tiene en cuenta variables de entrada, en cambio el control feedback es del tipo correctivo que sólo tiene en cuenta la salida, y juntos se complementan en un solo sistema anticipativo y correctivo a la vez.
El principio de funcionamiento del control feedfoward consta en estimar el caudal de aire que ingresa al motor y luego calcular la cantidad de combustible necesaria. Debido a la complejidad de modelar un motor de combustión interna sumado a los importantes retardos y la alinealidad en la realimentación, se consideró que el controlador más adecuado para el sistema feedback fue el algoritmo PID con resultados positivos.

Uno de los objetivos planteados fue que la UEC sea nacional y abierta, es por esto que la CIAA cumple con el requisito debido a su procedencia nacional, de bajo costo y además la escasa necesidad de agregar electrónica adicional para su implementación. La CIAA cumple con estándares de seguridad en sofware con su

sistema operativo diseñado principalmente para la industria automotriz.

El sistema desarrollado fue ensayado y puesto a prueba sobre un motor en marcha bajo condiciones reales, y los resultados obtenido fueron buenos pero con la necesidad de complementarlo para llegar a un producto que se pueda insertar en el mercado automotriz.

8.2. Trabajo futuro

En este apartado se enuncia un serie de temas que pueden complementar o mejorar el trabajo que se realizó hasta este punto.

A continuación se enumeran algunas propuestas de las mejoras que pueden contribuir al el sistema de inyección basado en la CIAA, y así poder lograr una UEC abierta que pueda reemplazar a las ya existentes en el mercado.

- Extender el sistema de control de inyección a múltiples puntos de funcionamiento. Para lograr este objetivo es necesario trabajar sobre un banco de pruebas en el que se pueda introducir carga al motor en forma de torque y así poder lograr un análisis bajo distintos puntos de trabajo. El principal objetivo es conseguir un mapa o tabla estática que contenga las constantes de control Kp y Ki óptimas en función al grado de carga.

- Mejorar la precisión del controlador feedforward planteado, y por otro lado diseñarlo para que sea a prueba de fallas.

- Proponer y desarrollar un sistema de control de inyección distinto al planteado.

- Complementar las funciones que faltan para que la CIAA funcione como una UEC:

  - Desarrollar el sistema de ignición de la mezcla controlando el avance de encendido de las bujías.
  - Controlar la mariposa motorizada en función a la posición del pedal de aceleración.
  - Desarrollar un sistema de comunicación CAN para el control de dispositivos del automóvil (climatizador, cierre centralizado, techo solar, asientos eléctricos, etc).

- Desarrollar una HMI que permita realizar gráficos de los diferentes parámetros de funcionamiento del motor:
  - RPM.
  - Presión.
  - Temperatura.
  - Tiempo de inyección.
  - Posición de mariposa.
  - Posición de pedal de aceleración.
- Realizar un sistema de auto guardado en memoria de códigos de fallas.

Apéndice A

Código de programa

A.1. Código fuente OSEK OS

```
1
2  /*==================[inclusions
       ]============================================*/
3  #include "chip.h"
4  #include "os.h"               /* <= operating system header
       */
5  #include "ciaaPOSIX_stdio.h"  /* <= device handler header
       */
6  #include "ciaaPOSIX_stdlib.h"
7  #include "ciaaPOSIX_string.h" /* <= string header */
8  #include "ciaak.h"              /* <= ciaa kernel header */
9  #include "ciaaGPIOINT.h"
10 #include "inyeccion4.h"         /* <= own header */
11
12
13 /** \brief File descriptor for digital output ports
14  *
15  * Device path /dev/dio/out/0
16  */
17 static int32_t fd_out;
18 static int32_t fd_in;
19 static int32_t fd_adc_0, fd_adc_1;
20 uint8_t outputs=0;
21 uint16_t salida;
22 TaskStateType TaskState1;
23 /** \brief File descriptor of the USB uart
```

```
24   *
25   * Device path /dev/serial/uart/1
26   */
27          static int32_t fd_uart1;
28  /*==================[ variables de TASK(ContdorTask)
       ]================================*/
29  volatile uint32_t * DWT_CTRL = (uint32_t *)0xE0001000;
30  volatile uint32_t * DWT_CYCCNT = (uint32_t *)0xE0001004;
31  uint32_t Ciclos;                //ciclos entre diente y
       diente
32  uint8_t N_diente=0;             //Numero de diente: de 1 a
       58
33  float T_diente=0;                       //Tiempo de un
       diente
34  float T_vuelta;                 //Tiempo de una vuelta
35  float RPM=750;                  //Revoluciones por minuto
       del motor
36  /*==================[constantes del micro
       ]================================*/
37  const float F_clok=204000;   // Frecuencia del clok en KHz
38  const uint16_t NADC=1023;       // Cantidad de Valores
       posibles del adc
39  uint8_t Vref=10;                //tension de referencia del
        ADC (10V en un divisor resistivo de 15K ==3.3v)
40  /*==================[ variables de TASK(SensoresTask)
       ]================================*/
41  float Pos_Mar;          //Posicion de la mariposa: de 0 a
       90
42  float Pr_Ad=50;                 // Presion de admision en
       Kpa
43  float T_aireK=300;// Temperatura del aire de admision
       kelvin
44  float T_aire=15;  // Temperatura del aire de admision C
45  float T_agua=90;                        // Temperatura de
       agua de radiador
46  uint8_t Pos1_pedal, Pos2_pedal;
47  uint8_t Pos1_mariposa, Pos2_mariposa;
48
49  /*==================[ variable of task(Deteccion_faseTask)
       ]================================*/
50  uint8_t Fase=2;                 //Fase del Motor 0 o 1
51  /*==================[ constantes fisicas
       ]================================*/
```

```
52 const float Vol=1.149; // volumen cilindrada [litros]
53 const float G_i=0.002333;   // ganancia de inyector en [kg/
      s]
54 const uint32_t R=287; // R constente de gas ideal [kj/kg]
55 const float AFR=14.6;   // Relacion aire combustible
      deseada
56 const uint32_t RPM_corte=6000; //Revoluciones maximas
      permitidas
57 const uint32_t RPM_on=5000; // Revolucion de activacion de
      inyeccion luego de corte
58
59 /*  Presion de admision [Kpa]  %
      10,3|19,9|29,9|39,8|50,2|60,1|70,1|80.0|90.0      RPM*/
60 const float Rend[18][9]= {
61 {75.5,75.0,74.5,74.5,74.0,70.5,70.0,71.5,71.5}, //750
62 {75.0,75.5,78.9,75.0,74.0,73.5,72.5,71.5,71.5}, //1000
63 {77.0,77.5,79.0,76.5,75.5,74.5,74.0,73.0,73.0}, //1125
64 {77.5,79.5,81.0,77.5,76.5,75.5,74.0,73.0,73.0}, //1250
65 {80.0,84.0,84.0,79.0,77.5,76.0,74.0,73.0,74.5}, //1500
66 {80.0,83.0,82.0,80.0,78.0,77.0,75.5,74.5,74.5}, //1750
67 {79.5,82.0,84.0,83.0,81.0,79.5,79.0,78.0,80.5}, //2000
68 {78.5,82.0,86.0,87.0,85.0,84.0,82.0,81.0,81.0}, //2260
69 {78.0,82.0,88.5,88.0,86.5,85.5,83.0,82.0,81.0}, //2500
70 {77.5,82.0,89.5,88.0,89.0,83.5,81.5,80.0,72.5}, //2750
71 {75.5,81.5,89.0,87.0,91.5,81.0,79.0,79.0,75.0}, //3000
72 {68.0,79.5,90.5,90.0,91.5,85.0,84.0,81.5,76.0}, //3500
73 {66.0,78.0,90.0,92.0,92.0,89.0,86.5,83.5,80.0}, //4000
74 {73.5,77.5,85.5,90.0,91.0,88.5,87.0,84.0,80.0}, //4500
75 {77.5,78.5,85.5,89.5,91.0,88.0,85.0,81.0,78.0}, //5000
76 {84.0,83.5,84.5,88.0,91.0,87.0,83.0,79.5,76.5}, //5500
77 {84.0,83.5,84.5,87.5,91.0,87.0,84.0,79.0,76.5}, //6000
78 {84.0,83.5,84.5,87.5,91.0,87.5,84.5,79.5,76.5}};//6500*/
79 /*==================[ variables de conrol FF: TASK(
      Controlador_FeedforwardTask)
      ]================================*/
80 float Ti_FF=2;          //Tiempo de inyeccion Feedforward [
      ms];
81 /*==================[ variables de conrol FB: TASK(
      Controlador_FeedbackTask)
      ]================================*/
82 float Ti_FB=0;          //Tiempo de inyeccion Feedback [ms
      ];
83 float K_I=0.1;
```

```
84  float Ki=0.001;
85  float K_P=0.1;
86  float e_n=0;
87  float e_anterior=0;
88  /*==================[ variable de tiempo de inyeccion
       ]==============================*/
89  float T_inyeccion=4;// Tiempo de inyeccion en mS
90  uint32_t T_i=4000;// Tiempo de inyeccion en uS
91  uint32_t T_i2=5000;// Tiempo de inyeccion en uS
92  float Ti_FBanterior;
93  uint8_t Estado1, Estado2, Estado3, Estado4;
94  uint8_t Estado_corte=0;
95  float tiempo_us, tiempo;
96
97  static uint8_t x1 = 0;
98  static uint8_t x2 = 0;
99  static uint8_t x3 = 0;
100 static uint8_t x4 = 0;
101
102
103 static uint8_t num1 = 48;
104 static uint8_t num2 = 49;
105 static uint8_t num3 = 50;
106 static uint8_t num4 = 51;
107
108 static uint8_t num21 = 48;
109 static uint8_t num22 = 49;
110 static uint8_t num23 = 50;
111 static uint8_t num24 = 51;
112 char mez[5]={'R','I','C','A',' '};;
113 const char RICA[5]={'R','I','C','A'};
114 const char POBRE[5]={'P','O','B','R','E'};
115 uint8_t alarmFF=0;
116 uint8_t alarmFB =0;
117 float TiFF, TiFB;
118
119 int main(void)
120 {
121    /* Starts the operating system in the Application Mode 1
        */
122    /* This example has only one Application Mode */
123    StartOS(AppMode1);
124
```

```
125     /* StartOs shall never returns, but to avoid compiler
           warnings or errors
126      * 0 is returned */
127     return 0;
128   }
129
130   /** \brief Error Hook function
131    *
132    * This fucntion is called from the os if an os interface (
          API) returns an
133    * error. Is for debugging proposes. If called this
          function triggers a
134    * ShutdownOs which ends in a while(1).
135    *
136    * The values:
137    *    OSErrorGetServiceId
138    *    OSErrorGetParam1
139    *    OSErrorGetParam2
140    *    OSErrorGetParam3
141    *    OSErrorGetRet
142    *
143    * will provide you the interface, the input parameters and
           the returned value.
144    * For more details see the OSEK specification:
145    * http://portal.osek-vdx.org/files/pdf/specs/os223.pdf
146    *
147    */
148   void ErrorHook(void)
149   {
150   TaskType task;
151   TaskType taskID;
152   GetTaskID(&task);
153   ciaaPOSIX_printf("ErrorHook was called\n");
154   ciaaPOSIX_printf("Service: %d, P1: %d, P2: %d, P3: %d, RET:
          %d\n", OSErrorGetServiceId(), OSErrorGetParam1(),
         OSErrorGetParam2(), OSErrorGetParam3(), OSErrorGetRet())
         ;
155   ShutdownOS(0);
156   }
157
158   /** \brief Application IRQ Handler Callback for GPIO */
159   void appIrqHandler(void)
160   {
```

```
161 Ciclos=*DWT_CYCCNT;
162 *DWT_CTRL |= 1;
163 *DWT_CYCCNT=0;
164 ActivateTask(ContadorTask);
165 }
166
167 /** \brief Initial task
168  *
169  * This task is started automatically in the application
        mode 1.
170  */
171 TASK(InitTask)
172 {
173          /* init CIAA kernel and devices */
174 ciaak_start();
175 /* init CIAA Timers */
176 T_Inyec1_Tim0Init();
177 T_Inyec2_Tim1Init();
178 T_Inyec3_Tim2Init();
179 T_Inyec4_Tim3Init();
180 /* print message (only on x86) */
181 ciaaPOSIX_printf("Init Task...\n");
182 /* open CIAA digital inputs */
183 fd_in = ciaaPOSIX_open("/dev/dio/in/0", ciaaPOSIX_O_RDONLY)
        ;
184 /* open CIAA digital outputs */
185 fd_out = ciaaPOSIX_open("/dev/dio/out/0", ciaaPOSIX_O_RDWR)
        ;
186 /* Enable interrupt with GPIO DI0*/
187 ciaaGpioIntEnable(0, appIrqHandler); //Abilita interrupcion
        por GPIO de entrada digital 0
188 /* open UART connected to USB bridge (FT2232) */
189 fd_uart1 = ciaaPOSIX_open("/dev/serial/uart/1",
        ciaaPOSIX_O_RDWR);
190 /* change baud rate for uart usb */
191 ciaaPOSIX_ioctl(fd_uart1, ciaaPOSIX_IOCTL_SET_BAUDRATE, (
        void *)ciaaBAUDRATE_115200);
192 /* change FIFO TRIGGER LEVEL for uart usb */
193 ciaaPOSIX_ioctl(fd_uart1,
        ciaaPOSIX_IOCTL_SET_FIFO_TRIGGER_LEVEL, (void *)
        ciaaFIFO_TRIGGER_LEVEL3);
194 /* open CIAA ADC0 */
```

```
195 fd_adc_0 = ciaaPOSIX_open("/dev/serial/aio/in/0",
       ciaaPOSIX_O_RDONLY);
196 ciaaPOSIX_ioctl(fd_adc_0, ciaaPOSIX_IOCTL_SET_SAMPLE_RATE,
       100000);
197 ciaaPOSIX_ioctl( fd_adc_0 , ciaaPOSIX_IOCTL_SET_RESOLUTION,
        ciaaRESOLUTION_10BITS);
198 ciaaPOSIX_ioctl(fd_adc_0, ciaaPOSIX_IOCTL_SET_CHANNEL,
       ciaaCHANNEL_0);
199 /* open CIAA ADC1 */
200 fd_adc_1 = ciaaPOSIX_open("/dev/serial/aio/in/1",
       ciaaPOSIX_O_RDONLY);
201 ciaaPOSIX_ioctl(fd_adc_1, ciaaPOSIX_IOCTL_SET_SAMPLE_RATE,
       100000);
202 ciaaPOSIX_ioctl( fd_adc_1, ciaaPOSIX_IOCTL_SET_RESOLUTION,
       ciaaRESOLUTION_10BITS);
203 ciaaPOSIX_ioctl(fd_adc_1, ciaaPOSIX_IOCTL_SET_CHANNEL,
       ciaaCHANNEL_3);
204 /*SET Alarms*/
205 SetRelAlarm(ActivateFaseTask, 100, 5);
206 ActivateTask(SerialEchoTask);
207 SetRelAlarm(ActivateControladorFeedforwardTask, 102, 50);
208 alarmFF=1;
209 SetRelAlarm(ActivateControladorFeedbackTask, 1000, 20);
210 alarmFB=1;
211 Ki=K_I*0.02;
212 /* terminate task */
213 TerminateTask();
214 }
215 TASK(SerialEchoTask)
216 {
217 int8_t buf[16],buf2[16];   /* buffer for uart operation
                  */
218 uint8_t i_n=0;  /* to store outputs status
       */
219 int32_t ret;      /* return value variable for posix calls
        */
220 char C_ON[16] = "C_ON\r\n\0";
221 char C_OFF[16] = "C_OFF\r\n\0";
222 char DataON[16] = "DataON\r\n\0";
223 char DataOFF[16] = "DataOFF\r\n\0";
224 float tiempo_us, tiempo;
225 static uint8_t alarmD=0;
226 static uint8_t c_manual;
```

```
227 ciaaPOSIX_printf("SerialEchoTask...\n");
228 /* send a message to the world :) */
229 char message[] = "Hi! :)\nSerialEchoTask: Waiting for
       characters...\n";
230 ciaaPOSIX_write(fd_uart1, message, ciaaPOSIX_strlen(message
       ));
231
232 while(1)
233    {
234 /* wait for any character ... */
235 ret = ciaaPOSIX_read(fd_uart1, buf, 20);
236 if(ret > 0)
237 {
238 GetResource(RES1);
239 for (i_n=0; i_n<=19; i_n++)
240 {
241 buf2[ i_n]=buf[ i_n];
242 }
243 for ( i_n=ret ;  i_n<=19;  i_n++)
244  {
245 buf2[ i_n]=0;
246 }
247 if(ciaaPOSIX_strcmp(buf2,C_ON) == 0)
248                    {
249 if (alarmFF == 1)
250 {
251 CancelAlarm(ActivateControladorFeedforwardTask);
252 alarmFF=0;
253 }
254 if (alarmFB == 1)
255 {
256 CancelAlarm(ActivateControladorFeedbackTask);
257 alarmFB=0;
258 }
259 c_manual=1;
260 }
261 if(ciaaPOSIX_strcmp(buf2,C_OFF) == 0)
262 {
263 if (alarmFF == 0)
264 {
265 SetRelAlarm(ActivateControladorFeedforwardTask, 102, 50);
266 alarmFF=1;
267 }
```

```
268 if (alarmFB == 0){
269 SetRelAlarm(ActivateControladorFeedbackTask, 1000, 20);
270 alarmFB=1;
271 }
272 c_manual=0;
273  }
274 if (c_manual==1)
275 {
276 if ((buf2[0]=='T') && (buf2[1]=='I'))
277 {
278 tiempo_us=(buf2[2]-48)*1000 + (buf2[3]-48)*100 + (buf2
       [4]-48)*10 + buf2[5]-48;
279 tiempo=tiempo_us/(float)1000;
280 T_i=tiempo_us;
281 T_inyeccion=tiempo;
282 }
283 if ((buf2[0]=='K') && (buf2[1]=='P'))
284 {
285 K_P=(buf2[2]-48)*0.1 + (buf2[3]-48)*0.01;
286 }
287 if ((buf2[0]=='K') && (buf2[1]=='I'))
288 {
289 K_I=(buf2[2]-48)*0.1 + (buf2[3]-48)*0.01;
290 Ki=K_I*0.02;
291 }
292 }
293 if((ciaaPOSIX_strcmp(buf2,DataON) == 0)&&(alarmD==0))
294 {
295 SetRelAlarm(ActivateWriteTask, 700, 0);
296 alarmD=1;
297 }
298 if((ciaaPOSIX_strcmp(buf2,DataOFF) == 0)&&(alarmD==1))
299 {
300 CancelAlarm(ActivateWriteTask);
301 alarmD=0;
302 }
303 ReleaseResource(RES1);
304 }
305 }
306 }
307
308 TASK(WriteTask)
309 {
```

```
310 uint8_t n, input, Pos1_mariposa1;
311 char TI[3], TAI[2], TAG[3], PM[2],TF[3], TB[3], S[1], KP
       [2], KI[2];
312 float RPM1, Pr_Ad1, T_aire1, TiFF1, TiFB1, T_agua1, K_P1,
       K_I1;
313 static uint8_t selec=0;
314 uint32_t T_i1;
315 GetResource(RES1);
316 RPM1=RPM;
317 Pr_Ad1=Pr_Ad;
318 T_i1=T_i;
319 T_aire1=T_aire;
320 TiFF1=TiFF;
321 TiFB1=TiFB;
322 Pos1_mariposa1=Pos1_mariposa;
323 T_agua1=T_agua;
324 K_P1=K_P;
325 K_I1=K_I;
326 ReleaseResource(RES1);
327         x1 = (RPM1/1000);
328         num1 = x1+48;
329         x2=((RPM1-x1*1000)/100);
330         num2=x2+48;
331         x3=((RPM1-x1*1000-x2*100)/10);
332         num3=x3+48;
333         x4=(RPM1-x1*1000-x2*100-x3*10);
334         num4=x4+48;
335         x1 = (Pr_Ad1/1000);
336         num21 = x1+48;
337         x2=((Pr_Ad1-x1*1000)/100);
338         num22=x2+48;
339         x3=((Pr_Ad1-x1*1000-x2*100)/10);
340         num23=x3+48;
341         x4=(Pr_Ad1-x1*1000-x2*100-x3*10);
342         num24=x4+48;
343         x1 = (T_i1/1000);
344         TI[0] = x1+48;
345         x2=((T_i1-x1*1000)/100);
346         TI[1]=x2+48;
347         x3=((T_i1-x1*1000-x2*100)/10);
348         TI[2]=x3+48;
349
```

```
350          ciaaPOSIX_read(fd_in, &input, 1); // lectura de
                entradas digitales
351          input&=0x02; //Selecciona la entrada de sonda
                lambda
352          switch (input)
353          {
354          case 0x00:// Mezcla pobre
355                  for (n=0; n<=5 ; n++)
356                  {
357                  mez[n]=POBRE[n];
358                  }
359          break;
360          case 0x02: //mezcla Rica
361                  for (n=0; n<=5 ; n++)
362                  {
363                  mez[n]=RICA[n];
364                  }
365          break;
366          }
367          char dato[] = {'R','P','M',' ',num1,num2,num3,num4,
                ' ','P','_','A',' ',num21,num22,num23,num24,' ',
                'M','e','z',' ',mez[0],mez[1],mez[2],mez[3],mez
                [4],' ','\n'};
368          x1 = (TiFF1/1000);
369          TF[0] = x1+48;
370          x2=((TiFF1-x1*1000)/100);
371           TF[1]=x2+48;
372           x3=((TiFF1-x1*1000-x2*100)/10);
373           TF[2]=x3+48;
374  if (TiFB1<0){
375          S[0]=45;
376          TiFB1=TiFB1*-1;
377  }
378  else
379  {
380          S[0]=43;
381  }
382          x1 = (TiFB1/1000);
383          TB[0] = x1+48;
384          x2=((TiFB1-x1*1000)/100);
385          TB[1]=x2+48;
386          x3=(TiFB1-x1*1000-x2*100)/10;
387          TB[2]=x3+48;
```

```
388         x1 = (Pos1_mariposa1/10);
389         PM[0] = x1+48;
390         x2=((Pos1_mariposa1-x1*10));
391         PM[1]=x2+48;
392         x1 = (T_aire1/10);
393         TAI[0] = x1+48;
394         x2=((T_aire1-x1*10));
395         TAI[1]=x2+48;
396         char dato2[] ={' ','T','I',' ',TI[0],'.',TI[1],TI
               [2],' ','T','F',' ',TF[0],'.',TF[1],TF[2],' ','T
               ','B',' ',S[0],TB[0],'.',TB[1],TB[2],' ','\n'};
397         x1 = (T_agua1/100);
398         TAG[0] = x1+48;
399         x2=((T_agua1-x1*100)/10);
400         TAG[1]=x2+48;
401         x3=((T_agua1-x1*100-x2*10));
402         TAG[2]=x3+48;
403         x1 = (K_P1*10);
404         KP[0] = x1+48;
405         x2=(K_P1*100-x1*10);
406         KP[1]=x2+48;
407         x1 = (K_I1*10);
408         KI[0] = x1+48;
409         x2 = (K_I1*100- x1*10);
410         KI[1] = x2+48;
411 char dato3[] = {' ','P','M',' ',PM[0],PM[1],' ','T','A','I'
      ,' ',TAI[0],TAI[1],' ','T','A','G',' ',TAG[0],TAG[1],TAG
      [2],' ','K','P',' ','0','.',KP[0],KP[1],' ','K','I',' ',
      '0','.',KI[0],KI[1],' ','\n'};
412 ciaaPOSIX_write(fd_uart1, dato, ciaaPOSIX_strlen(dato));
413 ciaaPOSIX_write(fd_uart1, dato2, ciaaPOSIX_strlen(dato2));
414 ciaaPOSIX_write(fd_uart1, dato3, ciaaPOSIX_strlen(dato3));
415 SetRelAlarm(ActivateWriteTask, 700, 0);
416 TerminateTask();
417 }
418
419 /** \brief ..... Task*/
420 TASK(SensoresTask)
421 {
422 static  uint16_t CH0adc,CH1adc,CH2adc,CH3adc;
423 static  float aux1, Pr_ad_anterior;
424 static uint8_t mux=0;
425 float V_CH0, V_CH1, V_CH2;
```

```
426             switch (mux)
427             {
428                      case 0://CHANNEL=0
429                     ciaaPOSIX_read(fd_adc_0, &CH0adc, sizeof(
                           CH0adc));
430                     V_CH0=CH0adc*10/(float)1024;
431                     Pos1_mariposa=20.6*V_CH0;
432
433
434                     ciaaPOSIX_ioctl(fd_adc_0,
                           ciaaPOSIX_IOCTL_SET_CHANNEL,
                           ciaaCHANNEL_1);
435                     mux=1;
436                     break;
437                     case 1: //CHANNEL=1
438                      ciaaPOSIX_read(fd_adc_0, &CH1adc, sizeof(
                            CH1adc));
439                     V_CH1=CH1adc*10/(float)1024;
440                     T_agua=(-40.75*V_CH1)+111;
441                      ciaaPOSIX_ioctl( fd_adc_0,
                            ciaaPOSIX_IOCTL_SET_CHANNEL,
                            ciaaCHANNEL_2);
442                     mux=2;
443                      break;
444                     case 2: // CHANNEL=2
445                     ciaaPOSIX_read(fd_adc_0, &CH2adc, sizeof(
                           CH2adc));
446                     V_CH2=CH2adc*10/(float)1024;
447                     GetResource(RES1);
448
449                                     if (V_CH2<1.24)
450                                             {
451                                             T_aire=(-34*V_CH2)
                                                   +82;
452                                             }
453                                     else
454                                             {
455                                             T_aire=(-23*V_CH2)
                                                   +68;
456                                             }
457
458                                     ReleaseResource(RES1);
```

```
459                 ciaaPOSIX_ioctl( fd_adc_0,
                       ciaaPOSIX_IOCTL_SET_CHANNEL,
                       ciaaCHANNEL_0);
460                 mux=0;
461                 break;
462                 default :
463                 mux=0;
464                 ciaaPOSIX_ioctl( fd_adc_0,
                       ciaaPOSIX_IOCTL_SET_CHANNEL,
                       ciaaCHANNEL_0);
465                 break;
466          }
467         /* Lectura de Presion de admision por canal
               analogico CHANEL 3*/
468         ciaaPOSIX_read(fd_adc_1 , &CH3adc, 2);
469         GetResource(RES1);
470         Pr_Ad=(19.06*Vref*CH3adc/NADC)+13.18;//Presion de
               admision de colector en Kpa
471         if ((Pr_Ad<22)||(Pr_Ad>90))
472                 {
473                 Pr_Ad=45;
474                 }
475         ReleaseResource(RES1);
476 /* terminate task */
477 TerminateTask();
478 }
479
480 TASK(FaseTask)
481 {
482         uint8_t fase1;
483         ciaaPOSIX_read(fd_in, &fase1, 1);
484         fase1&=0x04;
485         if (fase1==0x00)
486         {
487         Fase=0x00;
488         CancelAlarm(ActivateFaseTask);
489         }
490 /* terminate task */
491 TerminateTask();
492 }
493
494 TASK(ContadorTask)
495 {
```

```
496 static uint32_t Ciclos_ant=0;
497 static uint32_t Ciclos_nue=0;
498         if ((Ciclos_ant!=0)&&(Ciclos_nue!=0))
499                 {
500                 if((2.4*Ciclos_nue<Ciclos)&&(Pr_Ad<50)){
501                 N_diente=1;
502                 Fase^= 0x01;
503                 }
504                 else{
505                 N_diente++;
506                 T_diente=Ciclos/F_clok;  //Tiempo entre
                       diente y diente [ms] Variable externa
507                 T_vuelta=T_diente*60*0.001;// tiempo de una
                        vuelta [s]
508                 GetResource(RES1);
509                 RPM=60/T_vuelta;        //Revoluciones del
                       motor (Variable global)
510                 ReleaseResource(RES1);
511                 Ciclos_ant=Ciclos_nue;
512                 Ciclos_nue=Ciclos;
513                 if (N_diente>58)
514                 {
515                 N_diente=1;
516                 Fase^= 0x01;
517                 }
518                 }
519                 }
520         if((Ciclos_ant!=0)&&(Ciclos_nue==0))
521                 {
522                 if((Ciclos<1.5*Ciclos_ant)&&(Ciclos_ant
                       *0.5<Ciclos))
523                         {
524                          Ciclos_nue=Ciclos;
525                         }
526                 else
527                         {
528                         Ciclos_ant=Ciclos;
529                         }
530                 }
531
532         if((Ciclos_ant==0)&&(Ciclos<510000))
533                 {
534                 Ciclos_ant=Ciclos;
```

```
535                     }
536             if(Ciclos>999000)
537                     {
538                     Ciclos_ant=0;
539                     Ciclos_nue=0;
540                     T_diente=0;
541                     }
542             if ((T_diente!=0)&&(Fase<=1))
543                     {
544                     ActivateTask(AvanceInyeccTask);
545                     }
546     /* terminate task */
547     TerminateTask();
548     }
549
550     TASK(AvanceInyeccTask)
551     {
552     TaskStateType TaskState0;
553     static float N_adelanto, N_adelanto_anterior, Diferencia;
554     static uint8_t Fase_1, Fase_2, Fase_3 , Fase_4, In_1_4,
           In_2_3, In_1_4_anterior, In_2_3_anterior,
           Fase_1_anterior, Fase_2_anterior, Fase_3_anterior,
           Fase_4_anterior;
555     N_adelanto_anterior=N_adelanto;
556     N_adelanto=T_inyeccion/T_diente;
557     N_adelanto=(int)N_adelanto;
558     Diferencia=N_adelanto-N_adelanto_anterior;
559     In_1_4_anterior=In_1_4;
560     In_2_3_anterior=In_2_3;
561     Fase_1_anterior=Fase_1;
562     Fase_2_anterior=Fase_2;
563     Fase_3_anterior=Fase_3;
564     Fase_4_anterior=Fase_4;
565             if((In_1_4!=N_diente)&&(In_2_3!=N_diente))
566                     {
567                     if (N_adelanto<16)
568                             {
569                             In_1_4=16-N_adelanto;
570                             In_2_3=46-N_adelanto;
571                             Fase_1=0x00;
572                             Fase_2=0x01;
573                             Fase_3=0x00;
574                             Fase_4=0x01;
```

```
575                         }
576                 else
577                         {
578                         if ((N_adelanto>=16)&&(N_adelanto
                               <=17))
579                                 {
580                                 In_1_4=58;
581                                 Fase_1=0x01;
582                                 In_2_3=46-N_adelanto;
583                                 Fase_1=0x01;
584                                 Fase_2=0x01;
585                                 Fase_3=0x00;
586                                 Fase_4=0x00;
587                                 }
588                         else
589                         {
590                         if ((N_adelanto>=18)&&(N_adelanto
                               <46))
591                                 {
592                                 In_1_4=75-N_adelanto;
593                                 In_2_3=46-N_adelanto;
594                                 Fase_1=0x01;
595                                 Fase_2=0x01;
596                                 Fase_3=0x00;
597                                 Fase_4=0x00;
598                                 }
599                         else
600                                 {
601                                 if (N_adelanto>46)
602                                         {
603                                         In_1_4=75-
                                                N_adelanto;
604                                         In_2_3=104-
                                                N_adelanto;
605                                         Fase_1=0x01;
606                                         Fase_2=0x00;
607                                         Fase_3=0x01;
608                                         Fase_4=0x00;
609                                         }
610                                 }
611                         }
612                         }
613                 }
```

```
614
615         if(Diferencia>2){
616         if((Fase_1_anterior==Fase_1)&&(In_1_4<=N_diente)&&(
               N_diente<=In_1_4_anterior)){
617         In_1_4=N_diente;
618         }
619         else{
620                 if((Fase_1_anterior=!Fase_1)&&((In_1_4<=
                       N_diente)||(N_diente<=In_1_4_anterior)))
                       {
621                 In_1_4=N_diente;
622                 Fase_1=Fase;
623         }
624
625         }
626         if((Fase_4_anterior==Fase_4)&&(In_1_4<=N_diente)&&(
               N_diente<=In_1_4_anterior)){
627         In_1_4=N_diente;
628         }
629         else{
630                 if((Fase_4_anterior=!Fase_4)&&((In_1_4<=
                       N_diente)||(N_diente<=In_1_4_anterior)))
                       {
631                 In_1_4=N_diente;
632                 Fase_4=Fase;
633                 }
634         }
635         if((Fase_2_anterior==Fase_2)&&(In_2_3<=N_diente)&&(
               N_diente<=In_2_3_anterior)){
636         In_2_3=N_diente;
637         }
638         else{
639                 if((Fase_2_anterior=!Fase_2)&&((In_2_3<=
                       N_diente)||(N_diente<=In_2_3_anterior)))
                       {
640                 In_2_3=N_diente;
641                 Fase_2=Fase;
642                 }
643         }
644         if((Fase_3_anterior==Fase_3)&&(In_2_3<=N_diente)&&(
               N_diente<=In_2_3_anterior)){
645         In_2_3=N_diente;
646         }
```

```
647             else{
648                     if((Fase_3_anterior=!Fase_3)&&((In_2_3<=
                           N_diente)||(N_diente<=In_2_3_anterior)))
                           {
649                     In_2_3=N_diente;
650                     Fase_2=Fase;
651             }
652             }
653             }
654
655
656             if ((N_diente==In_1_4)&&(Fase==Fase_1)&&(T_i>1000))
657                     {
658                     GetTaskState(Inyeccion_1Task, &TaskState0);
659                     if (SUSPENDED == TaskState0)
660                             {
661                             ActivateTask(Inyeccion_1Task);
662                             }
663                     }
664
665             if ((N_diente==In_2_3)&&(Fase==Fase_2)&&(T_i>1000))
666                     {
667                     GetTaskState(Inyeccion_2Task, &TaskState0);
668                     if (SUSPENDED== TaskState0)
669                             {
670                             ActivateTask(Inyeccion_2Task);
671                             }
672                     }
673             if ((N_diente==In_2_3)&&(Fase==Fase_3)&&(T_i>1000))
674                     {
675                     GetTaskState(Inyeccion_3Task, &TaskState0);
676                     if (SUSPENDED == TaskState0)
677                             {
678                             ActivateTask(Inyeccion_3Task);
679                             }
680                     }
681             if ((N_diente==In_1_4)&&(Fase==Fase_4)&&(T_i>1000))
682                     {
683                     GetTaskState(Inyeccion_4Task, &TaskState0);
684                     if (SUSPENDED == TaskState0)
685                             {
686                             ActivateTask(Inyeccion_4Task);
687                             }
```

```
688                     }
689     /* terminate task */
690     TerminateTask();
691     }
692
693     TASK(Inyeccion_1Task)
694     {
695             uint16_t Inyector_1;
696             ciaaPOSIX_read(fd_out, &Inyector_1, 1);
697             Inyector_1|=0x10;
698             ciaaPOSIX_write(fd_out, &Inyector_1, 1);
699             T_Inyec1(T_i);// Activacion de timer en us (
                   T_inyeccion)
700             WaitEvent(corte);
701             ClearEvent(corte);
702             ciaaPOSIX_read(fd_out, &Inyector_1, 1);
703             Inyector_1&=0xEF;
704             ciaaPOSIX_write(fd_out, &Inyector_1, 1);
705     /* terminate task */
706     TerminateTask();
707     }
708
709
710
711     TASK(Inyeccion_2Task)
712     {
713             uint16_t Inyector_2;
714             ciaaPOSIX_read(fd_out, &Inyector_2, 1);
715             Inyector_2|=0x20;
716             ciaaPOSIX_write(fd_out, &Inyector_2, 1);
717             T_Inyec2(T_i); // Activacion de timer en us (T_i)
718             WaitEvent(corte);
719             ClearEvent(corte);
720             ciaaPOSIX_read(fd_out, &Inyector_2, 1);
721             Inyector_2&=0xDF;
722             ciaaPOSIX_write(fd_out, &Inyector_2, 1);
723     /* terminate task */
724     TerminateTask();
725     }
726
727     TASK(Inyeccion_3Task)
728     {
729             uint16_t Inyector_3;
```

```
730         ciaaPOSIX_read(fd_out, &Inyector_3, 1);
731         Inyector_3|=0x40;
732         ciaaPOSIX_write(fd_out, &Inyector_3, 1);
733         T_Inyec3(T_i); // Activacion de timer en us (T_i)
734         WaitEvent(corte);
735         ClearEvent(corte);
736         ciaaPOSIX_read(fd_out, &Inyector_3, 1);
737         Inyector_3&=0xBF;
738         ciaaPOSIX_write(fd_out, &Inyector_3, 1);
739 /* terminate task */
740 TerminateTask();
741 }
742 TASK(Inyeccion_4Task)
743 {
744         uint8_t Inyector_4;
745         ciaaPOSIX_read(fd_out, &Inyector_4, 1);
746         Inyector_4|=0x80;
747         ciaaPOSIX_write(fd_out, &Inyector_4, 1);
748         T_Inyec4(T_i); // Activacion de timer en us (T_i)
749         WaitEvent(corte);
750         ClearEvent(corte);
751         ciaaPOSIX_read(fd_out, &Inyector_4, 1);
752         Inyector_4&=0x7F;
753         ciaaPOSIX_write(fd_out, &Inyector_4, 1);
754 /* terminate task */
755 TerminateTask();
756 }
757
758
759 TASK(Controlador_FeedforwardTask)
760 {
761         float X, Y, Z_x0, Z_x1, Rend_vol, Ti_ff1, aux;
762         uint8_t fila1, fila2, columna1,columna2;
763         uint16_t X_0, X_1, Y_0, Y_1;
764         static uint8_t alarma=1;
765         float min, max, min2, max2;
766         GetResource(RES1);
767         Y=Pr_Ad;
768         X=RPM;
769         ReleaseResource(RES1);
770  // Exploracion de Tabla estatica de rendimiento
     volumetrico:
771         if (750>=X){
```

```
772                     fila1=1;
773                     fila2=1;
774                     X_0=0;
775                     X_1=750;
776             }
777             else {
778             if ((750<X)&&(X<1000)){
779                     fila1=1;
780                     fila2=2;
781                     X_0=750;
782                     X_1=1000;
783             }
784             else{
785             if ((1000<=X)&&(X<1125)){
786                     fila1=2;
787                     fila2=3;
788                     X_0=1000;
789                     X_1=1125;
790             }
791             else{
792             if ((1125<=X)&&(X<1250)){
793                     fila1=3;
794                     fila2=4;
795                     X_0=1125;
796                     X_1=1250;
797             }
798             else{
799             if ((1250<=X)&&(X<1500)){
800                     fila1=4;
801                     fila2=5;
802                     X_0=1250;
803                     X_1=1500;
804             }
805             else{
806             if ((1500<=X)&&(X<1750)){
807                     fila1=5;
808                     fila2=6;
809                     X_0=1500;
810                     X_1=1750;
811             }
812             else{
813             if ((1750<=X)&&(X<2000)){
814                     fila1=6;
```

```
815                     fila2=7;
816                     X_0=1750;
817                     X_1=2000;
818             }
819             else{
820             if ((2000<=X)&&(X<2250)){
821                     fila1=7;
822                     fila2=8;
823                     X_0=2000;
824                     X_1=2250;
825             }
826             else{
827             if ((2250<=X)&&(X<2500)){
828                     fila1=8;
829                     fila2=9;
830                     X_0=2250;
831                     X_1=2500;
832             }
833             else{
834             if ((2500<=X)&&(X<3000)){
835                     fila1=9;
836                     fila2=10;
837                     X_0=2500;
838                     X_1=3000;
839             }
840             else{
841             if ((3000<=X)&&(X<3500)){
842                     fila1=10;
843                     fila2=11;
844                     X_0=3000;
845                     X_1=3500;
846             }
847
848             else{
849             if ((3500<=X)&&(X<4000)){
850                     fila1=11;
851                     fila2=12;
852                     X_0=3500;
853                     X_1=4000;
854             }
855             else{
856             if ((4000<=X)&&(X<4500)){
857                     fila1=12;
```

```
858                     fila2=13;
859                     X_0=4000;
860                     X_1=4500;
861             }
862             else{
863             if ((4500<=X)&&(X<5000)){
864                     fila1=13;
865                     fila2=14;
866                     X_0=4500;
867                     X_1=5000;
868             }
869             else{
870             if ((5000<=X)&&(X<5500)){
871                     fila1=14;
872                     fila2=15;
873                     X_0=5000;
874                     X_1=5500;
875             }
876             else{
877             if ((5500<=X)&&(X<6000)){
878                     fila1=15;
879                     fila2=16;
880                     X_0=5500;
881                     X_1=6000;
882             }
883             else{
884             if ((6000<=X)&&(X<6500)){
885                     fila1=16;
886                     fila2=17;
887                     X_0=6000;
888                     X_1=6500;
889             }
890             else{
891             if (6500<=X){
892                     fila1=17;
893                     fila2=17;
894                     X_0=6500;
895                     X_1=7000;
896             }
897             }
898                     }}}}}}}}}}}}}}}
899
900             if (Y<=10.3){
```

```
901                     columna1=1;
902                     columna2=1;
903                     Y_0=0;
904                     Y_1=10.3;
905             }
906             else{
907             if ((10.3<Y)&&(Y<=19.9)){
908                     columna1=1;
909                     columna2=2;
910                     Y_0=10.3;
911                     Y_1=19.9;
912             }
913             else{
914             if ((19.9<Y)&&(Y<=29.9)){
915                     columna1=2;
916                     columna2=3;
917                     Y_0=19.9;
918                     Y_1=29.9;
919             }
920             else{
921             if ((29.9<Y)&&(Y<=39.8)){
922                     columna1=3;
923                     columna2=4;
924                     Y_0=29.9;
925                     Y_1=39.8;
926             }
927             else{
928             if ((39.8<Y)&&(Y<=50.2)){
929                     columna1=4;
930                     columna2=5;
931                     Y_0=39.8;
932                     Y_1=50.2;
933             }
934             else{
935             if ((50.2<Y)&&(Y<=60.1)){
936                     columna1=5;
937                     columna2=6;
938                     Y_0=50.2;
939                     Y_1=60.1;
940             }
941             else{
942             if ((60.1<Y)&&(Y<=70.1)){
943                     columna1=6;
```

```
944                     columna2=7;
945                     Y_0=60.1;
946                     Y_1=70.1;
947             }
948             else{
949             if ((70.1<Y)&&(Y<=80)){
950                     columna1=7;
951                     columna2=8;
952                     Y_0=70.1;
953                     Y_1=80;
954             }
955             else{
956             if ((80<Y)&&(Y<=90)){
957                     columna1=8;
958                     columna2=9;
959                     Y_0=80;
960                     Y_1=90;
961             }
962             }}}}}}}}
963             //Interpolacion trapezoidal de rendimiendo
                   volumetrico.
964             Z_x0=(X-X_0)*(Rend[fila2][columna1]-Rend[fila1][
                   columna1])/(X_0-X_1)+Rend[fila1][columna1];
965             Z_x1=(X-X_0)*(Rend[fila2][columna2]-Rend[fila1][
                   columna2])/(X_0-X_1)+Rend[fila1][columna2];
966             Rend_vol=((Y-Y_0)*(Z_x1-Z_x0)/(Y_1-Y_0)+Z_x0)*0.01;
967
968             Ti_ff1=Vol*Rend_vol*Y*1000/(float)(G_i*AFR*R*
                   T_aireK*4);/*Ti_ff[ms] (tiempo de inyeccion FF)
                   Vol[L](cilindrada), X=P_adm [Kpa](presion de
                   multiple), Rend_vol (rendimiento volumetrico),
                   G_i [kg/s](ganacia de inyeccion), R constante de
                    gases, T_aire emperatura del aire deadmision,
                   AFR=14,6 (relacion A/C)*/
969             min=Ti_FF*0.9;
970             max=Ti_FF*1.1;
971             min2=T_inyeccion*0.9;
972             max2=T_inyeccion*1.1;
973             if ((Ti_ff1<min)||(Ti_ff1>max))
974                     {
975                     if(alarmFB==1)
976                             {
977                             alarmFB=0;
```

```
978                             CancelAlarm(
                                   ActivateControladorFeedbackTask)
                                   ;
979                             Ti_FB=0;
980                             e_anterior=0;
981                             }
982
983                     Ti_FF=Ti_ff1;
984                     TiFF= Ti_FF*1000;
985                     if(Ti_FF<3.4)
986                     {
987                     Ti_FF=3.4;
988
989                     }
990                     if(Ti_FF>10)
991                     {
992                     Ti_FF=9.9;
993
994                     }
995                     T_inyeccion=Ti_FF;
996                     aux=T_inyeccion*1000;
997                     T_i=(int)aux;
998                     if(alarmFB==0)
999                             {
1000                            alarmFB=1;
1001                            SetRelAlarm(
                                   ActivateControladorFeedbackTask,
                                    1000, 20);
1002                            }
1003
1004                    }
1005    /* terminate task */
1006    TerminateTask();
1007    }
1008
1009    TASK(Controlador_FeedbackTask)
1010    {
1011            uint8_t sonda_lambda;
1012            float aux2;
1013            ciaaPOSIX_read(fd_in, &sonda_lambda, 1); // lectura
                   de entradas digitales
1014            sonda_lambda&=0x02; //Selecciona la entrada de
                  sonda lambda
```

```
1015            switch (sonda_lambda)
1016                    {
1017                    case 0x00:// Mezcla pobre
1018                    e_n=1;
1019
1020                    if(e_n!=e_anterior){
1021                                    e_anterior=0;
1022
1023                            }
1024                    if(T_inyeccion<6){
1025                    if (Ti_FB<1.2)
1026                            {
1027                            Ti_FB= K_P*e_n + Ki*e_n +
                                   Ti_FBanterior - K_P*e_anterior;
1028                            }
1029                    e_anterior=e_n;
1030                    Ti_FBanterior=Ti_FB;
1031
1032
1033                    }
1034
1035
1036                    break;
1037                    case 0x02: //mezcla Rica
1038                    e_n=-1;
1039                    if(e_n!=e_anterior){
1040                            e_anterior=0;
1041
1042                    }
1043                    if (T_inyeccion>2){
1044                    if (Ti_FB>-1.2)
1045                            {
1046                            Ti_FB=K_P*e_n + Ki*e_n +
                                   Ti_FBanterior - K_P*e_anterior;
1047                            }
1048                    e_anterior=e_n;
1049                    Ti_FBanterior=Ti_FB;
1050             }
1051                    break;
1052                    }
1053            GetResource(RES1);
1054            TiFB=Ti_FB*1000;
1055            T_inyeccion=Ti_FF+Ti_FB;
```

```
1056            if((T_inyeccion<6)&&(T_inyeccion>2.7))
1057                    {
1058                    aux2=T_inyeccion*1000;
1059                    T_i=(int)aux2;
1060                    }
1061            else
1062                    {
1063                    if(T_inyeccion>9)
1064                            {
1065                            T_i=9900;
1066                            T_inyeccion=9.9;
1067                            }
1068                    if(T_inyeccion<=2.3)
1069                            {
1070                            T_i=2300;
1071                            T_inyeccion=2.3;
1072                            }
1073                    }
1074            ReleaseResource(RES1);
1075    /* terminate task */
1076    TerminateTask();
1077    }
1078
1079    TASK(Corte_InyeccionTask)
1080    {
1081            SetRelAlarm(ActivateCorteTask, 5, 0);
1082            if (RPM>RPM_corte)
1083                    {
1084                    CancelAlarm(ActivateControladorFeedbackTask
                            );
1085                    CancelAlarm(
                            ActivateControladorFeedforwardTask);
1086                    T_i=0;
1087                    T_inyeccion=0;
1088                    Estado_corte=1;
1089                    }
1090            if (Estado_corte==1)
1091                    {
1092                    if (RPM<=RPM_on)
1093                            {
1094                            SetRelAlarm(
                                    ActivateControladorFeedforwardTask
                                    , 25, 20);
```

```
1095                            CancelAlarm(ActivateCorteTask);
1096                            Estado_corte=0;
1097                            }
1098                    }
1099    /* terminate task */
1100    TerminateTask();
1101    }
1102    /*==================[Declaraciones de funciones
           ]=============================================*/
1103    /* Tiempo de inyeccion 1 us */
1104    void T_Inyec1(uint32_t usec1)
1105    {
1106    Chip_TIMER_SetMatch(LPC_TIMER0, 0, usec1);
1107    Chip_TIMER_Reset(LPC_TIMER0);
1108    Chip_TIMER_Enable(LPC_TIMER0);
1109    }
1110    /* Tiempo de inyeccion 2 us */
1111    void T_Inyec2(uint32_t usec2)
1112    {
1113    Chip_TIMER_SetMatch(LPC_TIMER1, 0, usec2);
1114    Chip_TIMER_Reset(LPC_TIMER1);
1115    Chip_TIMER_Enable(LPC_TIMER1);
1116    }
1117    /* Tiempo de inyeccion 3 us */
1118    void T_Inyec3(uint32_t usec3)
1119    {
1120    Chip_TIMER_SetMatch(LPC_TIMER2, 0, usec3);
1121    Chip_TIMER_Reset(LPC_TIMER2);
1122    Chip_TIMER_Enable(LPC_TIMER2);
1123    }
1124    /* Tiempo de inyeccion 4 us */
1125    void T_Inyec4(uint32_t usec4)
1126    {
1127    Chip_TIMER_SetMatch(LPC_TIMER3, 0, usec4);
1128    Chip_TIMER_Reset(LPC_TIMER3);
1129    Chip_TIMER_Enable(LPC_TIMER3);
1130    }
1131    /* Inicializa al Timer 0 utilizado para realizar el Tiempo
           de inyeccion us N1 */
1132    void T_Inyec1_Tim0Init(void)
1133    {
1134    Chip_TIMER_Init(LPC_TIMER0);
```

```
1135 Chip_TIMER_PrescaleSet(LPC_TIMER0, Chip_Clock_GetRate(
        CLK_MX_TIMER0)/1000000 - 1);
1136 Chip_TIMER_MatchEnableInt(LPC_TIMER0, 0);
1137 Chip_TIMER_ResetOnMatchEnable(LPC_TIMER0, 0);
1138 Chip_TIMER_StopOnMatchDisable(LPC_TIMER0, 0);
1139 Chip_TIMER_SetMatch(LPC_TIMER0, 0, 1000);
1140 Chip_TIMER_Reset(LPC_TIMER0);
1141 Chip_TIMER_Enable(LPC_TIMER0);
1142 NVIC_EnableIRQ(TIMER0_IRQn);
1143 }
1144 /* Desactiva al Timer 0 */
1145 void T_Inyec1_Tim0DeInit(void)
1146 {
1147 Chip_TIMER_Disable(LPC_TIMER0);
1148 NVIC_DisableIRQ(TIMER0_IRQn);
1149 }
1150 /* Inicializa al Timer 1 utilizado para realizar el Tiempo
        de inyeccion us N2 */
1151 void T_Inyec2_Tim1Init(void)
1152 {
1153 Chip_TIMER_Init(LPC_TIMER1);
1154 Chip_TIMER_PrescaleSet(LPC_TIMER1,
1155 Chip_Clock_GetRate(CLK_MX_TIMER1)/1000000 - 1);
1156 Chip_TIMER_MatchEnableInt(LPC_TIMER1, 0);
1157 Chip_TIMER_ResetOnMatchEnable(LPC_TIMER1, 0);
1158 Chip_TIMER_StopOnMatchDisable(LPC_TIMER1, 0);
1159 Chip_TIMER_SetMatch(LPC_TIMER1, 0, 1000);
1160 Chip_TIMER_Reset(LPC_TIMER1);
1161 Chip_TIMER_Enable(LPC_TIMER1);
1162 NVIC_EnableIRQ(TIMER1_IRQn);
1163 }
1164 /* Desactiva al Timer 1 */
1165 void  T_Inyec2_Tim1DeInit(void)
1166 {
1167 Chip_TIMER_Disable(LPC_TIMER1);
1168 NVIC_DisableIRQ(TIMER1_IRQn);
1169 }
1170 /* Inicializa al Timer 2 utilizado para realizar el Tiempo
        de inyeccion us N3 */
1171 void T_Inyec3_Tim2Init(void)
1172 {
1173 Chip_TIMER_Init(LPC_TIMER2);
1174 Chip_TIMER_PrescaleSet(LPC_TIMER2,
```

```
1175 Chip_Clock_GetRate(CLK_MX_TIMER2)/1000000 - 1);
1176 Chip_TIMER_MatchEnableInt(LPC_TIMER2, 0);
1177 Chip_TIMER_ResetOnMatchEnable(LPC_TIMER2, 0);
1178 Chip_TIMER_StopOnMatchDisable(LPC_TIMER2, 0);
1179 Chip_TIMER_SetMatch(LPC_TIMER2, 0, 1000);
1180 Chip_TIMER_Reset(LPC_TIMER2);
1181 Chip_TIMER_Enable(LPC_TIMER2);
1182 NVIC_EnableIRQ(TIMER2_IRQn);
1183 }
1184
1185 /* Desactiva al Timer 2 */
1186 void  T_Inyec3_Tim2DeInit(void)
1187 {
1188 Chip_TIMER_Disable(LPC_TIMER2);
1189 NVIC_DisableIRQ(TIMER2_IRQn);
1190 }
1191 /* Inicializa al Timer 3 utilizado para realizar el Tiempo
       de inyeccion us N4 */
1192 void T_Inyec4_Tim3Init(void)
1193 {
1194 Chip_TIMER_Init(LPC_TIMER3);
1195 Chip_TIMER_PrescaleSet(LPC_TIMER3,
1196 Chip_Clock_GetRate(CLK_MX_TIMER3)/1000000 - 1);
1197 Chip_TIMER_MatchEnableInt(LPC_TIMER3, 0);
1198 Chip_TIMER_ResetOnMatchEnable(LPC_TIMER3, 0);
1199 Chip_TIMER_StopOnMatchDisable(LPC_TIMER3, 0);
1200 Chip_TIMER_SetMatch(LPC_TIMER3, 0, 1000);
1201 Chip_TIMER_Reset(LPC_TIMER3);
1202 Chip_TIMER_Enable(LPC_TIMER3);
1203 NVIC_EnableIRQ(TIMER3_IRQn);
1204 }
1205 /* Desactiva al Timer 3 */
1206 void  T_Inyec4_Tim3DeInit(void)
1207 {
1208 Chip_TIMER_Disable(LPC_TIMER3);
1209 NVIC_DisableIRQ(TIMER3_IRQn);
1210 }
1211 /* Rutina de interrupcion por Timer 0*/
1212 ISR(TIMER0_IRQHandler)
1213 {
1214 if (Chip_TIMER_MatchPending(LPC_TIMER0, 0)) {
1215 Chip_TIMER_ClearMatch(LPC_TIMER0, 0);
1216 GetTaskState(Inyeccion_1Task, &TaskState1);
```

```
1217 if (WAITING == TaskState1)
1218 {
1219 SetEvent(Inyeccion_1Task, corte);
1220 }
1221 }
1222 }
1223 /* Rutina de interrupcion por Timer 1*/
1224 ISR(TIMER1_IRQHandler)
1225 {
1226 if (Chip_TIMER_MatchPending(LPC_TIMER1, 0))
1227 {
1228 Chip_TIMER_ClearMatch(LPC_TIMER1, 0);
1229 GetTaskState(Inyeccion_2Task, &TaskState1);
1230 if (WAITING == TaskState1)
1231 {
1232 SetEvent(Inyeccion_2Task, corte);
1233 }
1234 }
1235 }
1236 /* Rutina de interrupcion por Timer 2*/
1237 ISR(TIMER2_IRQHandler)
1238 {
1239 if (Chip_TIMER_MatchPending(LPC_TIMER2, 0))
1240 {
1241 Chip_TIMER_ClearMatch(LPC_TIMER2, 0);
1242 GetTaskState(Inyeccion_3Task, &TaskState1);
1243 if (WAITING == TaskState1)
1244 {
1245 SetEvent(Inyeccion_3Task, corte);
1246 }
1247 }
1248 }
1249 /* Rutina de interrupcion por Timer 3*/
1250 ISR(TIMER3_IRQHandler)
1251 {
1252 if (Chip_TIMER_MatchPending(LPC_TIMER3, 0))
1253 {
1254 Chip_TIMER_ClearMatch(LPC_TIMER3, 0);
1255 GetTaskState(Inyeccion_4Task, &TaskState1);
1256 if (WAITING == TaskState1)
1257 {
1258 SetEvent(Inyeccion_4Task, corte);
1259 }
```

```
1260 }
1261 }
1262
1263 /** @} doxygen end group definition */
1264 /** @} doxygen end group definition */
1265 /** @} doxygen end group definition */
1266 /*==================[end of file
     ]================================================*/
```

A.1.1. Código OIL OSEK Implementation Language

```
1
2  OSEK OSEK {
3
4     OS   ExampleOS {
5        STATUS = EXTENDED;
6        ERRORHOOK = TRUE;
7        PRETASKHOOK = FALSE;
8        POSTTASKHOOK = FALSE;
9        STARTUPHOOK = FALSE;
10       SHUTDOWNHOOK = FALSE;
11       USERESSCHEDULER = FALSE;
12       MEMMAP = FALSE;
13    };
14
15
16
17    TASK InitTask {
18       PRIORITY = 2;
19       ACTIVATION = 1;
20       AUTOSTART = TRUE {
21       APPMODE = AppMode1;
22       }
23       STACK = 512;
24       TYPE = EXTENDED;
25       SCHEDULE = NON;
26       RESOURCE = POSIXR;
27       EVENT = POSIXE;
28    }
29 TASK SerialEchoTask {
30     PRIORITY = 1;
```

```
    ACTIVATION = 1;
    STACK = 1024;
    TYPE = EXTENDED;
    SCHEDULE = FULL;
    EVENT = POSIXE;
    RESOURCE = POSIXR;
    RESOURCE = RES1;
}
TASK WriteTask {
    PRIORITY = 2;
    ACTIVATION =1 ;
    STACK = 1024;
    TYPE = EXTENDED;
    SCHEDULE = NON;
    EVENT = POSIXE;
    EVENT = READY;
    RESOURCE = POSIXR;
    RESOURCE = RES1;
}
  TASK SensoresTask {
      PRIORITY = 7;
      ACTIVATION = 1;
      STACK = 1024;
      TYPE = EXTENDED;
      SCHEDULE = NON;
      RESOURCE = POSIXR;
      RESOURCE = SALIDA;
      EVENT = POSIXE;
      RESOURCE = POSIXR;
      RESOURCE =RES1;


   }
 TASK FaseTask {
      PRIORITY = 3;
      ACTIVATION = 1;
      STACK =256;
      TYPE = BASIC;
      SCHEDULE = NON;
      RESOURCE = POSIXR;
      EVENT = POSIXE;
      RESOURCE =RES1;
   }
```

```
74
75      TASK ContadorTask {
76         PRIORITY =10;
77         ACTIVATION = 2;
78         STACK = 512;
79         TYPE = BASIC ;
80         SCHEDULE = NON;
81         RESOURCE = POSIXR;
82         EVENT = POSIXE;
83         RESOURCE =RES1;
84
85      }
86    TASK AvanceInyeccTask {
87         PRIORITY = 10;
88         ACTIVATION = 2;
89         STACK = 512;
90         TYPE = BASIC;
91         SCHEDULE = NON;
92         RESOURCE = POSIXR;
93         RESOURCE =RES1;
94      }
95   TASK Inyeccion_1Task {
96         PRIORITY = 9;
97         ACTIVATION = 1;
98         STACK = 1024;
99         TYPE = EXTENDED;
100        SCHEDULE = NON;
101        RESOURCE = POSIXR;
102        EVENT = POSIXE;
103        RESOURCE = SALIDA;
104        EVENT = corte;
105        RESOURCE =RES1;
106     }
107
108     TASK Inyeccion_2Task {
109        PRIORITY = 9;
110        ACTIVATION = 1;
111        STACK = 1024;
112        TYPE = EXTENDED;
113        SCHEDULE = NON;
114        RESOURCE = POSIXR;
115        EVENT = POSIXE;
116        RESOURCE = SALIDA;
```

```
117         EVENT = corte;
118         RESOURCE =RES1;
119      }
120   TASK Inyeccion_3Task {
121         PRIORITY =9;
122         ACTIVATION = 1;
123         STACK = 512;
124         TYPE = EXTENDED;
125         SCHEDULE = NON;
126         RESOURCE = POSIXR;
127         RESOURCE = SALIDA;
128         EVENT = corte;
129         EVENT = POSIXE;
130         RESOURCE =RES1;
131      }
132   TASK Inyeccion_4Task {
133         PRIORITY = 9;
134         ACTIVATION = 1;
135         STACK = 512;
136         TYPE = EXTENDED;
137         SCHEDULE = NON;
138         RESOURCE = POSIXR;
139         RESOURCE = SALIDA;
140         EVENT = corte;
141         EVENT = POSIXE;
142         RESOURCE =RES1;
143      }
144    TASK Controlador_FeedforwardTask {
145         PRIORITY = 5;
146         ACTIVATION = 10;
147         STACK = 512;
148         TYPE = BASIC;
149         SCHEDULE = NON;
150         RESOURCE = POSIXR;
151         EVENT = POSIXE;
152         RESOURCE =RES1;
153      }
154
155    TASK Controlador_FeedbackTask {
156         PRIORITY = 5;
157         ACTIVATION = 10;
158         STACK = 512;
159         TYPE = BASIC;
```

```
160         SCHEDULE = NON;
161         RESOURCE = POSIXR;
162         EVENT = POSIXE;
163         RESOURCE =RES1;
164       }
165
166    TASK Corte_InyeccionTask {
167         PRIORITY = 11;
168         ACTIVATION = 1;
169         STACK = 512;
170         TYPE = BASIC;
171         SCHEDULE = NON;
172         RESOURCE =RES1;
173       }
174
175    ALARM ActivateSensoresTask {
176                COUNTER = HardwareCounter;
177                ACTION = ACTIVATETASK {
178                    TASK = SensoresTask;
179                }
180                   AUTOSTART = TRUE {
181                   APPMODE = AppMode1;
182                   ALARMTIME = 31;
183                   CYCLETIME = 30;
184   }
185   };
186
187   ALARM ActivateFaseTask {
188       COUNTER = HardwareCounter;
189       ACTION = ACTIVATETASK {
190           TASK = FaseTask;
191       }
192   }
193   ALARM ActivateControladorFeedforwardTask{
194       COUNTER = HardwareCounter;
195       ACTION = ACTIVATETASK {
196           TASK = Controlador_FeedforwardTask ;
197       }
198   }
199   ALARM ActivateControladorFeedbackTask {
200       COUNTER = HardwareCounter;
201       ACTION = ACTIVATETASK {
202           TASK = Controlador_FeedbackTask ;
```

```
203     }
204 }
205 ALARM ActivateCorteTask{
206     COUNTER = HardwareCounter;
207     ACTION = ACTIVATETASK {
208         TASK = Corte_InyeccionTask  ;
209     }
210 }
211 ALARM ActivateWriteTask {
212     COUNTER = HardwareCounter;
213     ACTION = ACTIVATETASK {
214         TASK = WriteTask;
215     }
216 }
217  RESOURCE = POSIXR;
218  RESOURCE = SALIDA;
219  RESOURCE = RES1;
220    EVENT = POSIXE;
221    EVENT = corte;
222    APPMODE = AppMode1;
223
224
225
226    COUNTER HardwareCounter {
227       MAXALLOWEDVALUE = 1000;
228       TICKSPERBASE = 1;
229       MINCYCLE = 1;
230       TYPE = HARDWARE;
231       COUNTER = HWCOUNTER0;
232    };
233 ALARM IncrementSWCounter {
234    COUNTER = HardwareCounter;
235    ACTION = INCREMENT {
236       COUNTER = SoftwareCounter;
237    };
238    AUTOSTART = TRUE {
239       APPMODE = AppMode1;
240       ALARMTIME = 1;
241       CYCLETIME = 1;
242    };
243 };
244
245 COUNTER SoftwareCounter {
```

```
246     MAXALLOWEDVALUE = 1000;
247     TICKSPERBASE = 1;
248     MINCYCLE = 1;
249     TYPE = SOFTWARE;
250   };
251
252   ISR GPIOINTHandler0 {
253         INTERRUPT = GPIO0;
254             CATEGORY = 2;
255             PRIORITY = 0;
256     } ;
257
258   ISR ADC0_IRQHandler {
259       INTERRUPT = ADC0;
260       CATEGORY = 2;
261       PRIORITY = 0;
262   };
263
264   ISR ADC1_IRQHandler {
265       INTERRUPT = ADC1;
266       CATEGORY = 2;
267       PRIORITY = 0;
268   };
269    ISR TIMER0_IRQHandler {
270         INTERRUPT = TIMER0;
271             CATEGORY = 2;
272             PRIORITY = 0;
273     };
274
275    ISR TIMER1_IRQHandler {
276         INTERRUPT = TIMER1;
277             CATEGORY = 2;
278             PRIORITY = 0;
279     };
280
281    ISR TIMER2_IRQHandler {
282         INTERRUPT = TIMER2;
283             CATEGORY = 2;
284             PRIORITY = 0;
285     };
286
287    ISR TIMER3_IRQHandler {
288         INTERRUPT = TIMER3;
```

```
289             CATEGORY = 2;
290             PRIORITY = 0;
291     };
292   ISR UART0_IRQHandler {
293     INTERRUPT = UART0;
294     CATEGORY = 2;
295     PRIORITY = 0;
296   };
297
298
299   ISR UART2_IRQHandler {
300     INTERRUPT = UART2;
301     CATEGORY = 2;
302     PRIORITY = 0;
303   };
304
305   ISR UART3_IRQHandler {
306     INTERRUPT = UART3;
307     CATEGORY = 2;
308     PRIORITY = 0;
309   };
310
311
312   };
```

Bibliografía

[01, 2016] 01, W. (2016). Ciclo otto — wikipedia, la enciclopedia libre. [Internet; descargado 19-diciembre-2016].

[Aquino, 1981] Aquino, C. F. (1981). Transient a/f control characteristics of the 5 liter central fuel injection engine. Technical report, SAE Technical Paper.

[Cercós, 2001] Cercós, J. N. (2001). *Diseño de un controlador avanzado basado en redes neuronales para la gestión de la mezcla aire-gasolina en un motor alternativo*. PhD thesis, Universitat Politècnica de Catalunya.

[Cerdeiro, 2015] Cerdeiro, M. (2015). Introducción a osek-os - el sistema operativo del ciaa-firmware. programación de sistemas embebidos. Technical report.

[Falk and Mooney, 1980] Falk, C. and Mooney, J. (1980). Three-way conversion catalysts: Effect of closed-loop feed-back control and other parameters on catalyst efficiency. Technical report, SAE Technical Paper.

[Katashiba et al., 1991] Katashiba, H., Nishida, M., Washino, S., Takahashi, A., Hashimoto, T., and Miyake, M. (1991). Fuel injection control systems that improve three way catalyst conversion efficiency. Technical report, SAE Technical Paper.

[Medina et al., 2016] Medina, S., Pi Puig, M., Dell'Oso, M., Romero, F., De Giusti, A. E., and Tinetti, F. G. (2016). Comparación de sistemas operativos embebidos sobre la computadora industrial abierta argentina. In *XXII Congreso Argentino de Ciencias de la Computación (CACIC 2016).*

[ProyectoCIAA, 2016] ProyectoCIAA (2016). La ciaa una plataforma diferente pensada para la industria. [Internet; descargado 19-diciembre-2016].

[Taylor, 1966] Taylor, C. F. (1966). The internal combustion engine in theory and practice. volumes i and ii. *revised edition*.

[Wikipedia, 2016] Wikipedia (2016). Encendido del motor — wikipedia, la enciclopedia libre. [Internet; descargado 4-enero-2017].