

UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

**MEJORAMIENTO DEL DISEÑO DE CONTROL Y ELECTRÓNICO DE
UN VEHÍCULO AUTOBALANCEADO**

**MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELECTRICISTA**

RODRIGO ANDRÉS MAUREIRA TENORIO

PROFESOR GUÍA:
MANUEL DUARTE MERMOUD

MIEMBROS DE LA COMISIÓN:
HÉCTOR AGUSTO ALEGRÍA
CLAUDIO ALARCÓN REYES

SANTIAGO DE CHILE
DICIEMBRE 2010

"MEJORAMIENTO DEL DISEÑO DE CONTROL Y ELECTRÓNICO DE UN VEHÍCULO AUTOBALANCEADO"

El transporte ha cobrado gran relevancia en las ciudades modernas, en las que el aumento de población, los atochamientos y la contaminación, degradan la calidad de vida. Es en respuesta a estos problemas que surgen nuevos e innovadores medios de transporte, como el que se aborda en este trabajo. Un vehículo de pequeño tamaño consistente en una plataforma autobalanceda con dos ruedas laterales, en que el pasajero se ubica sobre la plataforma mientras las ruedas lo movilizan.

Específicamente, se aborda el mejoramiento de un prototipo desarrollado en 2009, el cual logró ser funcional, pero su operación requería un alto grado de destreza y entrenamiento, además de presentar un comportamiento, en ocasiones, poco predecible. Por este motivo se planteó en este trabajo, la intervención completa del vehículo en cuanto a diseño electrónico y software de control, con el fin de generar una segunda versión del mismo con mejoras técnicas sustanciales. Adicionalmente, se abordaron algunos aspectos mecánicos y estéticos, con el fin de proteger los componentes electrónicos y mejorar la experiencia de manejo, permitiendo que el usuario maniobre el vehículo inclinando el mando de dirección lateralmente, siguiendo el movimiento natural del cuerpo.

El rediseño de la electrónica se hizo en base a los conocimientos adquiridos en el trabajo anterior, integrando la información previa disponible y añadiendo elementos nuevos que permitieran cumplir los objetivos planteados. Los puentes H, encargados de accionar los motores, fueron reconstruidos utilizando una estructura de bus laminado, aumentando así su capacidad de corriente y reduciendo las inductancias parásitas. El mejoramiento de la electrónica de disparo de los puentes H, permitió reducir a una décima parte los tiempos de conmutación, mejorando la eficiencia del circuito. Se incorporaron además protecciones por hardware para evitar cortocircuitos, así como sensores para monitorear y controlar la corriente circulante por los motores.

Se desarrolló una nueva unidad IMU independiente de la tarjeta de control, capaz de obtener mediciones de inclinación y de su derivada en tiempo real, sin contaminación por ruido inducido, gracias a su interfaz completamente digital. La tarjeta de control también fue rediseñada para ajustarse a la nueva configuración de los sistemas, mediante conexiones limpias, minimizando el número de cables.

El nuevo software se desarrolló desde cero, utilizando un esquema modular e incorporando conceptos como la abstracción de hardware, para simplificar la portabilidad y facilitar la comprensión de su funcionamiento. Esto posibilita la utilización del vehículo como plataforma de desarrollo de estrategias de control, en la cual la interacción con el hardware y la administración del sistema ya están resueltas. Además, se implementó una interfaz de comunicación por comandos, a través de la cual se puede monitorear el sistema, modificar parámetros y ejecutar órdenes usando un programa cliente externo al vehículo, conectado vía USB a la tarjeta de control.

Desde el punto de vista electrónico y luego de los cambios introducidos, el vehículo se desempeñó correctamente. Las medidas tomadas para mejorar la robustez del sistema permiten una operación confiable, sin necesidad de precauciones adicionales por parte del usuario. La mayor cantidad y calidad de las mediciones de los instrumentos, junto con la aplicación de dos lazos en cascada (tanto para el control de inclinación a través de un torque en los motores, como para sus perturbaciones en la dinámica) mejoró sensiblemente el comportamiento del vehículo, lo que fue verificado a través de encuestas de percepción realizadas a usuarios novatos antes y después de las modificaciones. Destaca en estas evaluaciones el tiempo requerido para un manejo mínimo, que se redujo de unos 30 minutos a alrededor de 10 minutos.

"El éxito es el premio a la constancia, si ustedes trabajan cada día más y más, si no se detienen por nada llegarán a donde deseen llegar, el límite es el cielo, y el cielo es infinito, así de infinita debe ser tu fe en que las cosas sucederán, piensa en grande y llegarás a ser grande."

Waldylei Yépez

Agradecimientos

La realización de una Memoria para obtener el anhelado Título, es uno de los más grandes hitos dentro de la vida profesional y personal. Marca el instante en que toda una vida, llena de pequeños y grandes detalles, se confabula para consolidar una sola palabra: Ingeniero. Y es precisamente esa infinidad de detalles, lo que hace difícil mirar en retrospectiva y dar el crédito que merece a cada una de las personas que contribuyeron de una u otra forma a alcanzar la primera de las metas profesionales.

Sin embargo, hay actores cuya marca no puede omitirse y, con justa razón, muchos párrafos de agradecimiento nombran a la familia y yo no me quedaré atrás. Por el apoyo incondicional y permanente, por aguantar que desordenara la casa durante semanas e incluso meses con la excusa de estar *“trabajando en el vehículo”*, por no dudar en hacer lo que esté a su disposición para ayudarme a solventar cualquier inconveniente, por crear las condiciones necesarias para que me concentrara en mi carrera y sobre todo por enseñarme desde pequeño que soy yo el responsable de mi vida, que es mi deber hacerme cargo de ella, que es posible alcanzar las metas planteadas y que todos los esfuerzos rinden frutos tarde o temprano, agradezco a mis padres y a mi familia en general.

En cuanto a lo profesional, agradezco la guía, el apoyo y la confianza del Profesor Manuel Duarte, cuya premonitoria frase *“Yo garantizo de que todos ustedes van a terminar su carrera...”* dicha cuando cursábamos tercer año, da cuenta de una persona que cree en su gente, cuya rigurosidad y claridad mental le da crédito a sus palabras que, por supuesto, han resultado ciertas. Gracias también Claudio Alarcón, de quien he obtenido muchas de las herramientas que me permitieron enfrentar este desafío satisfactoriamente, siempre dispuesto a ayudar y sobre todo a enseñar. Leonardo Moreno es también uno de los grandes personajes que merecen ser mencionados, pues no sólo sentó las bases del trabajo apasionante, divertido y desafiante que he realizado, sino que también ha estado siempre dispuesto a cooperar, atento a la evolución del vehículo que, por muchas modificaciones que se le hagan, no dejará de ser su creación.

Agradezco también a todos mis compañeros, familiares y amigos que cooperaron de alguna forma al proyecto. A Ian Pelissier, Gonzalo Díaz, Rodrigo Asenjo, Daniel Rebolledo, Gabriel Maureira, Claudio Barra, a todos los usuarios de prueba y a todos quienes se entusiasmaron con el proyecto de una u otra manera.

Finalmente, nada de esto tendría sentido sin Waldylei Yépez, mi mayor pilar durante este proceso, quien me dio la fuerza necesaria para superar los momentos difíciles, quien siempre ha estado ahí, junto a mí, entregándome su amor, cariño, comprensión y respeto. Mi compañera, mi editora, mi amiga, mi refugio y tantas otras cosas que las palabras no pueden describir. Gracias Waldylei por estar conmigo, pues esto es por y para ti.

Índice General

AGRADECIMIENTOS.....	III
ÍNDICE GENERAL.....	IV
ÍNDICE DE FIGURAS.....	VII
CAPÍTULO 1: INTRODUCCIÓN.....	1
1.1 Motivación	1
1.2 Antecedentes	1
1.3 Objetivos Generales y Específicos	2
1.4 Estructura de la Memoria	3
CAPÍTULO 2: FUNDAMENTOS TÉCNICOS Y TEÓRICOS.....	5
2.1 Motor de Corriente Continua.....	5
2.2 Control de Velocidad de Motores C.C.	6
2.3 Puente H.....	8
2.4 Transistores MOSFET como interruptores de potencia	9
2.4.1 Resistencia de Conducción	10
2.4.2 Velocidad de Conmutación	10
2.4.3 Uso en Paralelo de Transistores MOSFET.....	11
2.5 Unidad de Medición Inercial (IMU)	14
CAPÍTULO 3: EVALUACIÓN INICIAL Y PROPUESTA DE MEJORAMIENTO	15
3.1 Estado Inicial del Vehículo.	15
3.1.1 Evaluación Técnica	15
3.1.2 Pruebas con Usuarios	20
3.2 Mejoras a Realizar.	21
CAPÍTULO 4: DISEÑO E IMPLEMENTACIÓN DE LAS MEJORAS	22
4.1 Mejoras en el Diseño Mecánico	22
4.1.1 Sistema de dirección	23

4.1.2	Montaje de componentes	23
4.1.3	Cubierta Exterior	25
4.2	Mejoras electrónicas	26
4.2.1	Modificación de la Electrónica de Potencia	28
4.2.2	Cambios en el Sistema de Instrumentación	31
4.2.3	Tarjeta de control (DSP)	32
4.2.4	Electrónica auxiliar	34
4.3	Software DSP	35
4.3.1	Requisitos del software	35
4.3.2	Administración de Tareas	36
4.3.3	HAL Timer	38
4.3.4	HAL Main	38
4.3.5	HAL Instrumentos	39
4.3.6	HAL PWM	40
4.3.7	HAL Corriente	40
4.3.8	HAL ADC	42
4.3.9	HAL GPIO	42
4.3.10	HAL Serial	43
4.3.11	Módulo de Comandos	44
4.3.12	Módulo de Control	46
4.3.13	Módulo Principal	48
4.4	Software de Monitoreo	49
 CAPÍTULO 5: PRUEBAS, RESULTADOS Y ANÁLISIS DEL VEHÍCULO MEJORADO		53
5.1	Aspectos Mecánicos	53
5.2	Mejoras Electrónicas.....	54
5.3	Evaluación de los Instrumentos	57
5.4	Estabilidad y Control.....	58
5.5	Experiencia del Usuario	59
5.6	Especificaciones Finales del Vehículo	61
 CAPÍTULO 6: CONCLUSIONES		62
6.1	Conclusiones.....	62
6.2	Trabajo Futuro.....	63
 REFERENCIAS.....		65
 ANEXO A: CIRCUITOS ESQUEMÁTICOS		67
	Tarjeta Unidad IMU	67

Tarjeta Puente H.....	68
Tarjeta de Control.....	69
Tarjeta de Lecturas Analógicas.....	70
ANEXO B: ENCUESTAS	71
Formato de Encuesta	71
Evaluación Inicial con Usuarios Novatos	72
Evaluación del Vehículo Modificado con Usuarios Novatos	72
Evaluación Comparativa del Vehículo con el Mismo Grupo de Usuarios	73
ANEXO C: CÓDIGOS FUENTE	74
Software DSP (Tarjeta de Control)	74
Software del Controlador de Display LCD	110
Software de Monitoreo	113

Índice de Figuras

Figura 2.1: (a) Bobina elemental del motor de C.C. dispuesta sobre un eje de giro y alimentada a través de las escobillas. (b) Bobina montada en un rotor dentro de un campo magnético fijo cuya dirección es perpendicular al eje de giro.....	5
Figura 2.2: Circuito equivalente de un motor C.C.	6
Figura 2.3: Circuito reductor de voltaje aplicado al control de velocidad de un motor C.C.	7
Figura 2.4: Circuito esquemático de un puente H.....	8
Figura 2.5: Representación de un transistor MOSFET canal N.	9
Figura 2.6: Características de salida del transistor IXTH130N10T [7]	10
Figura 2.7: Carga de la compuerta del transistor IXTH130N10T [7]	11
Figura 2.8: Diagrama de conexión para dos transistores en paralelo.	12
Figura 2.9: Circuito equivalente de dos MOSFET conectados en paralelo. Se muestra el camino de baja impedancia para valores pequeños de resistencia de compuerta.	13
Figura 2.10: Mediciones de una unidad IMU de 3 ejes aplicada a un avión [11].....	14
Figura 3.1: Estado inicial del vehículo.	15
Figura 3.2: Estado inicial del esqueleto de acero.	16
Figura 3.3: Diagrama de bloques de la tarjeta de control original [2].....	17
Figura 3.4: (Izquierda) Flanco de subida del pulso en la compuerta del transistor. (Derecha) Flanco de bajada del pulso en la compuerta del transistor.	18
Figura 3.5: Esquema de control presente en el vehículo.....	19
Figura 4.1: Tratamiento aplicado al esqueleto de acero del vehículo.	22
Figura 4.2: (Izquierda) Modificación a la estructura de acero para el montaje de la nueva dirección. (Derecha) Sistema ensamblado, se muestra la dirección de movimiento del mecanismo.....	23
Figura 4.3: Disposición original de los componentes electrónicos.....	24
Figura 4.4: Montaje final de componentes electrónicos en la parte central del vehículo.	24
Figura 4.5: Vista general de la disposición final de los componentes internos del vehículo.....	25
Figura 4.6: Vehículo con cubierta protectora instalada.	25
Figura 4.7: Perspectiva del usuario del visor LCD y los botones de comando del vehículo.	26

Figura 4.8: Diagrama de bloques de la electrónica de control.....	27
Figura 4.9: Protección contra disparo simultáneo.....	29
Figura 4.10: (Izquierda) Puente H con el bus plano ensamblado pero sin los componentes de potencia. (Derecha) Puente H completamente armado y funcional.	30
Figura 4.11: Diagrama lógico de la tarjeta de instrumentos IMU.....	31
Figura 4.12: Tarjeta de instrumentos IMU previa al montaje de sus componentes.....	32
Figura 4.13: Tarjeta de Control.....	33
Figura 4.14: Esquema del circuito utilizado para las lecturas analógicas.	34
Figura 4.15: Diagrama de flujo general del programa.....	37
Figura 4.16: Esquema de control de corriente.....	41
Figura 4.17: Referencia efectiva del controlador de corriente.....	42
Figura 4.18: Diagrama de estados del Módulo de Comandos.....	45
Figura 4.19: Esquema general de control del vehículo.....	46
Figura 4.20: Interfaz gráfica del software de monitoreo del vehículo.....	50
Figura 5.1: (Izquierda) Pulso de encendido en la compuerta de un transistor en el puente H original. (Derecha) Pulso de encendido en la compuerta de un transistor en el nuevo puente H.....	54
Figura 5.2: (Izquierda) En celeste se muestra el flanco de subida del pulso de encendido en la compuerta del transistor. (Derecha) Se muestra el flanco de bajada del pulso de encendido.	55
Figura 5.3: (Izquierda) Tren de pulsos aplicado a la una de las entradas del driver. (Centro) Tren de pulsos aplicado a la otra entrada del driver. (Derecha) Tren de pulsos aplicado a ambas entradas simultáneamente.....	56
Figura 5.4: Captura de datos de los instrumentos y compensación de retardo	57

Capítulo 1: Introducción

1.1 MOTIVACIÓN

El transporte es uno de los temas que ha cobrado gran relevancia en las ciudades modernas, producto de los problemas generados por el aumento de población, los atochamientos y la contaminación producida por los vehículos. Para resolver estos problemas se han impulsado diversas iniciativas que van desde las más clásicas, como el incentivo al uso de bicicletas y el transporte público, hasta las más sofisticadas, como el uso de nuevos vehículos pequeños y no contaminantes.

Entre los vehículos de estas características, destaca uno de tipo autobalanceado (capaz de mantener el equilibrio gracias a un sistema de control) conformado por una pequeña plataforma apoyada sobre dos ruedas laterales. El usuario ubica sus pies sobre la plataforma, la cual se equilibra gracias a la acción de los motores acoplados a las ruedas, realizando una acción de control similar a la del péndulo invertido. De esta manera, una ligera inclinación hacia adelante del usuario generará una acción de control que moverá al vehículo completo en esa dirección, realizando así la función de transporte.

El interés por el desarrollo de este tipo de vehículos se basa en: su tamaño, el cual le permite circular de manera amigable con otros transeúntes, sin requerir mayor infraestructura que la disponible para el uso de una silla de ruedas; su naturaleza es no contaminante por tratarse de un vehículo eléctrico; el funcionamiento silencioso del mismo y la facilidad de manejo, ya que carece de palancas, botones o mandos complicados para avanzar y retroceder.

Existe un único fabricante de este tipo de vehículos de manera comercial, la empresa Segway Inc. [1], cuyos productos poseen un alto precio como para ser usados masivamente. Este hecho, sumado a diversas iniciativas en distintos lugares del mundo para replicar esta tecnología, motiva el interés por desarrollar un vehículo autobalanceado en nuestro país, a un costo razonable y con prestaciones similares a la versión comercial.

1.2 ANTECEDENTES

Durante el año 2009 se desarrolló el primer vehículo autobalanceado en Chile, diseñado y construido por el estudiante Leonardo Moreno [2]. Para ello se realizó un análisis y modelación de las características dinámicas del sistema, con el objetivo de obtener la información necesaria para generar una estrategia de control que permitiera el funcionamiento exitoso del vehículo. Posteriormente se abordaron las etapas de diseño e implementación de las distintas partes que componen el vehículo, como la electrónica de potencia, electrónica digital y aspectos mecánicos, entre otros.

El prototipo logró ser funcional en el sentido de que permitía a una persona transportarse de un lugar a otro, evitando obstáculos, con un nivel básico de maniobrabilidad. Además, el proyecto se realizó con costos razonables para un vehículo de estas características, lo que mostró que es factible realizar este tipo de diseños en Chile con resultados exitosos. Sin embargo, la curva de aprendizaje para la conducción del vehículo no fue la esperada, ya que se buscaba obtener resultados similares al

Segway [1], que requiere de unos 5 minutos de práctica para manejarlo con comodidad. En este caso, eran necesarios alrededor de 30 minutos de práctica para poder manejarlo de forma segura y controlada, ya que una parte no menor del equilibrio dependía de la habilidad del usuario.

Se atribuyó gran parte de los problemas asociados al equilibrio del sistema a la Unidad de Medición Inercial (IMU¹), en particular, al giróscopo, el cual requería una constante corrección que no siempre funcionaba adecuadamente generando, en algunos casos, respuestas vigorosas que desestabilizaban al usuario. En consecuencia, se propuso el reemplazo de la unidad IMU por otra de mejor tecnología.

Respecto del control implementado, a pesar de tener un fundamento teórico desarrollado, su funcionamiento se basa más en la experimentación y calibración, por medio de ajustes de parámetros y curvas de comportamiento según las pruebas realizadas.

En conclusión, se construyó un vehículo funcional, con gran potencial pero aún con varios aspectos mejorables. El presente trabajo aborda dichos aspectos, continuando con el desarrollo del mismo vehículo, con el fin de permitir que pueda ser usado de forma más sencilla, segura y confiable.

1.3 OBJETIVOS GENERALES Y ESPECÍFICOS

Con el trabajo de esta memoria se persiguen los siguientes objetivos generales:

- Mejorar el desempeño electrónico del vehículo previamente diseñado, en cuanto a eficiencia, seguridad y confiabilidad.
- Aplicar una estrategia de control adecuada, que mejore la percepción del usuario y facilite el aprendizaje en la fase de conducción.
- Mejorar la Unidad de Medición Inercial, instalando un nuevo giróscopo y filtrando apropiadamente las señales de entrada.
- Dejar el vehículo convenientemente modificado para facilitar futuros desarrollos.

A partir de los objetivos generales mencionados anteriormente, se definen los siguientes objetivos específicos:

- Consolidar en un diseño definitivo las modificaciones, reparaciones y mejoras realizadas durante el trabajo anterior, las que permitieron hacer del vehículo un prototipo funcional.
- Incorporar medidas de protección que minimicen la posibilidad de fallas en la electrónica. En particular se debe evitar que los transistores de los puentes H reciban señales de encendido que puedan generar un cortocircuito.
- Utilizar técnicas para la minimización de ruido electrónico en las señales del sistema.
- Reducir los tiempos de conmutación de la electrónica de potencia, para así aumentar su eficiencia.

¹ Inertial Measurement Unit

- Construir una nueva unidad IMU independiente de la tarjeta de control, con instrumentos de mejor tecnología e interfaz digital.
- Implementar una estrategia de control proporcional derivativa para la inclinación del vehículo de manera más sistemática, capaz de mantener la estabilidad y permitir su operación de manera sencilla.
- Implementar técnicas de filtrado digital de las señales recibidas para mejorar la estimación de las variables físicas.
- Generar un software de control de fácil intervención, utilizando un esquema modular, que facilite trabajos futuros en cuanto a estrategia de control. Además de incorporar un sistema de comunicación adecuado para el monitoreo del sistema en tiempo real.

1.4 ESTRUCTURA DE LA MEMORIA

Este documento se estructura en 6 capítulos, cuya descripción detallada se indica a continuación:

Capítulo 1: Introducción

Presenta una visión general de lo que son los vehículos autobalanceados, sus características y la motivación para desarrollarlos. Se abordan posteriormente los antecedentes específicos de este trabajo, destacando los puntos importantes del desarrollo anterior. Finalmente se plantean los objetivos generales y específicos.

Capítulo 2: Fundamentos Técnicos y Teóricos

Este capítulo aborda los conceptos necesarios para comprender el funcionamiento del vehículo y los fundamentos técnicos que justifican algunas de las mejoras realizadas en él durante este trabajo.

Capítulo 3: Evaluación de la Situación Inicial y Propuesta de Mejoramiento

En este capítulo se realiza una exhaustiva evaluación del vehículo en su condición inicial, abordando aspectos mecánicos, electrónicos, de control y la percepción por parte de los usuarios. Se presenta una propuesta detallada de mejoras, motivada por la revisión realizada.

Capítulo 4: Diseño e Implementación de Mejoras

En este capítulo se describe de manera detallada cada una de las intervenciones y nuevos diseños realizados en el vehículo y sus sistemas asociados. Por su naturaleza, es el capítulo de mayor extensión y profundidad.

Capítulo 5: Pruebas, Resultados y Análisis del Vehículo Mejorado

Se evalúa nuevamente, de manera exhaustiva, el desempeño del vehículo luego de aplicados los cambios en su diseño. Se detallan además las especificaciones finales del equipo.

Capítulo 6: Conclusiones

Se analizan los resultados del trabajo desde un punto de vista integral, extrayendo las principales conclusiones de los resultados obtenidos. Se proponen además líneas de investigación y mejoras a realizar en futuros trabajos relacionados con el vehículo.

Capítulo 2: Fundamentos Técnicos y Teóricos

2.1 MOTOR DE CORRIENTE CONTINUA

El motor de corriente continua (C.C.) es la máquina eléctrica más antigua empleada en aplicaciones de potencia y tracción. Su sencillo principio de funcionamiento y gran versatilidad han permitido que siga vigente hasta nuestros días, a pesar de ser constructivamente más complejo que las máquinas de corriente alterna más modernas. Su velocidad fácilmente controlable, posibilidad de girar en ambos sentidos y capacidad de altos torques de partida, lo hacen ideal para aplicaciones de tracción.

El funcionamiento del motor de C.C. se basa en la fuerza generada por la interacción de un campo magnético inmóvil y uno generado por una bobina móvil, montada sobre un eje de rotación. La bobina móvil es alimentada a través de un sistema de escobillas y delgas para invertir la dirección de la corriente y, por consiguiente, el sentido del campo magnético generado, logrando que el torque resultante sea siempre favorable al sentido de giro (Figura 2.1a) como se menciona en Vargas 2006 [3]. En la Figura 2.1b se muestra la bobina dentro de un campo magnético fijo de dirección horizontal.

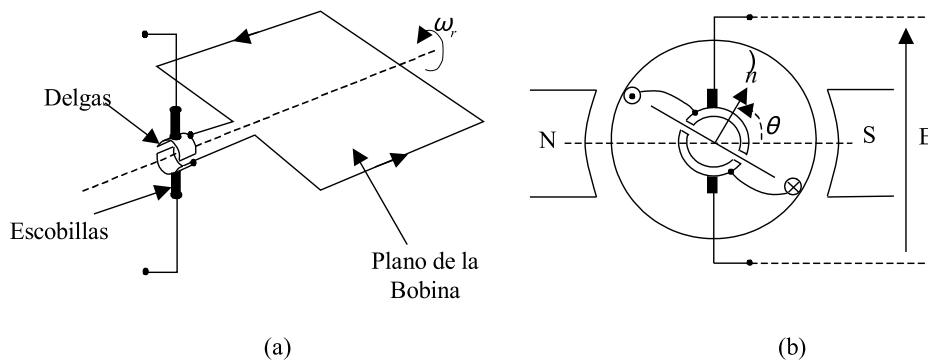


Figura 2.1: (a) Bobina elemental del motor de C.C. dispuesta sobre un eje de giro y alimentada a través de las escobillas. (b) Bobina montada en un rotor dentro de un campo magnético fijo cuya dirección es perpendicular al eje de giro.

El campo magnético de dirección fija generado por el estator puede ser producido por imanes permanentes, como en la Figura 2.1, o bien por otro enrollado.

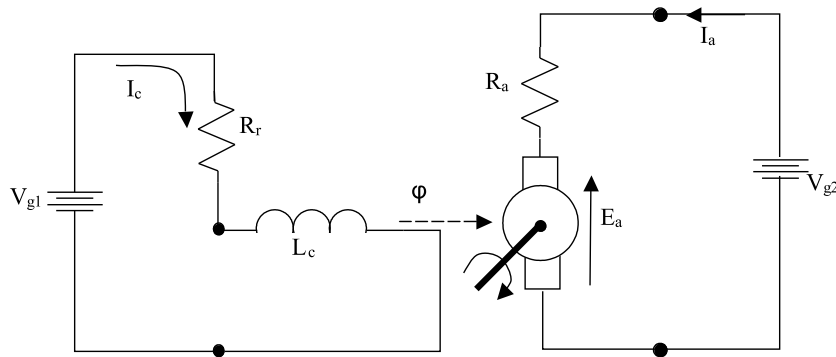


Figura 2.2: Circuito equivalente de un motor C.C.

El circuito equivalente del motor C.C., mostrado en la Figura 2.2, se divide en dos partes: El circuito de excitación (izquierda) que genera el campo magnético inmóvil al que se expone el rotor; y el circuito motriz (derecha) en el que se representa al rotor (o armadura) como una fuente de tensión E_a .

La interacción entre ambos circuitos queda descrita por las siguientes ecuaciones:

$$E_a = G \cdot \omega_r \cdot I_c \quad (2.1)$$

$$T = G \cdot I_c \cdot I_a \quad (2.2)$$

donde E_a es el voltaje de armadura, G es un parámetro de la máquina llamado inductancia rotacional, ω_r es la velocidad de giro del rotor y T es el torque generado. Finalmente I_c e I_a son las corrientes de excitación y de armadura, respectivamente.

Al ser R_a relativamente pequeña, se puede ver que la velocidad de giro depende fuertemente del voltaje de armadura E_a y de la corriente de excitación I_c . Sin embargo, para este trabajo se utilizan motores de imanes permanentes, por lo que el término I_c se considera constante.

El torque generado es directamente proporcional a la corriente de armadura, mientras que la presencia de R_a imposibilita un control directo y preciso de velocidad a través del voltaje aplicado a la armadura a pesar de su fuerte dependencia.

2.2 CONTROL DE VELOCIDAD DE MOTORES C.C.

Como se señaló anteriormente, existe una fuerte dependencia entre la velocidad del motor de corriente continua y el voltaje aplicado en sus terminales, sobre todo para el caso de imanes permanentes. Es por esto que, para controlar este tipo de motores, se busca construir fuentes de tensión variables continuamente.

La solución más sencilla es el uso de reóstatos para controlar el voltaje. Sin embargo, este componente limita fuertemente la corriente, además de disipar potencia innecesariamente, con la consiguiente alteración de las características de torque de la máquina.

Alternativamente, se pueden usar convertidores electrónicos para conseguir un control mucho más eficiente y versátil. Típicamente se utilizan circuitos reductores de voltaje o step-down para regular la velocidad de este tipo de motores, como el mostrado en la Figura 2.3, cuyo control de voltaje se realiza por medio de modulación de ancho de pulso (PWM) de la señal de encendido del transistor T1, el cual opera como interruptor, permitiendo el paso de corriente y cortándolo abruptamente a alta frecuencia, como se describe en Rashid 2001 [4].

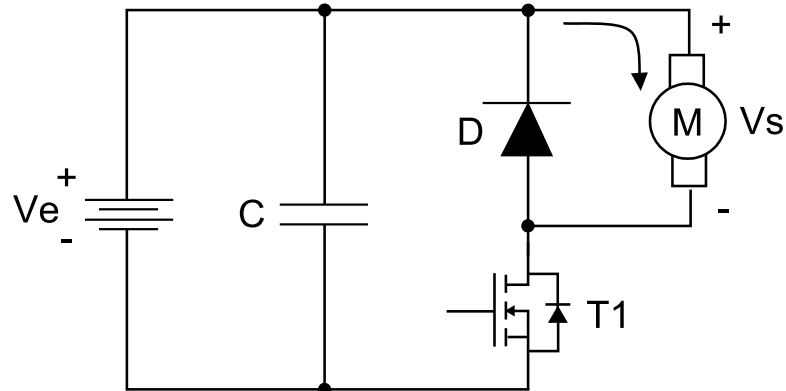


Figura 2.3: Circuito reductor de voltaje aplicado al control de velocidad de un motor C.C.

Dada la naturaleza inductiva de los enrollados del motor, es necesario incorporar un diodo en paralelo con éste para permitir la circulación de corriente durante el tiempo en que el transistor no conduce, así se evitan sobretensiones potencialmente peligrosas. Finalmente, el condensador es capaz de proveer altas corrientes por cortos períodos de tiempo para el arranque y además estabiliza el voltaje \$V_e\$ suprimiendo transientes.

La modulación de ancho de pulso consiste en el control del tiempo en que el transistor T1 conduce respecto al tiempo durante el cual está apagado para una frecuencia de conmutación fija. Se define el ciclo de trabajo como:

$$\delta = \frac{t_{on}}{t_{on} + t_{off}} = \frac{t_{on}}{T} \quad (2.3)$$

donde \$t_{on}\$ es el tiempo en que el transistor conduce y \$t_{off}\$ es el tiempo en que no conduce. \$T\$ es el período de conmutación.

Luego, el voltaje medio en el motor (\$V_s\$) está dado por:

$$V_s = \delta \cdot V_e \quad (2.4)$$

El control del ciclo de trabajo se puede realizar por medio de circuitos analógicos o digitales, permitiendo además la implementación de lazos de control para conseguir, por ejemplo, regular la corriente para obtener un torque deseado sin importar la velocidad.

2.3 PUENTE H

Circuitos como el de la Figura 2.3 permiten un control fluido y eficiente de la velocidad de un motor de corriente continua. Sin embargo, restringen el sentido de giro al proveer corriente únicamente en una dirección.

Para superar esta limitación, se utiliza un circuito llamado Convertidor de Puente Completo o Puente H, según lo presentado por Mohan 2003 [5], el cual utiliza la misma técnica de modulación de ancho de pulso (PWM) en una estructura de puente que permite la conducción en ambos sentidos, tal como se muestra en la Figura 2.4.

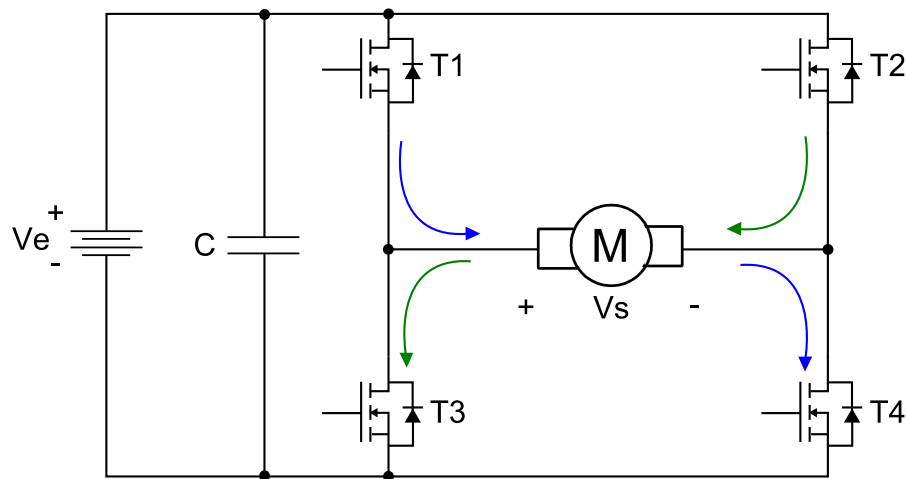


Figura 2.4: Circuito esquemático de un puente H.

En esta configuración, al encender T1 y T4 simultáneamente, se obtiene conducción en un sentido, mientras que al encender T2 y T3 simultáneamente se obtiene conducción en el sentido opuesto. Los diodos adosados a los transistores permiten la conducción cuando están todos los transistores abiertos.

Cabe destacar que encender simultáneamente T1 y T3, así como T2 y T4 producen un peligroso cortocircuito que debe ser evitado.

El uso de PWM en el encendido de los transistores permite controlar la velocidad del motor, tal como en el caso presentado en la sección 2.2.

2.4 TRANSISTORES MOSFET COMO INTERRUPTORES DE POTENCIA

Los transistores MOSFET² son utilizados masivamente como conmutadores gracias a sus características de tamaño, facilidad de uso y bajo consumo de energía, lo cual los convierte en el ladrillo constructor de los circuitos integrados digitales de hoy en día, estando presentes en prácticamente todos los dispositivos electrónicos, según lo mencionado por Brews 2000 [6]. Dichas características también los hacen idóneos para aplicaciones de potencia, en que se requiere gran capacidad de corriente y conmutaciones veloces.

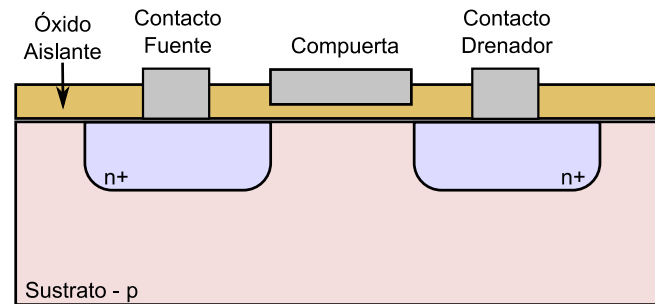


Figura 2.5: Representación de un transistor MOSFET canal N.

Un MOSFET es un dispositivo semiconductor de tres terminales denominados **fuelle** (*source*), **compuerta** (*gate*) y **drenador** (*drain*) como se describe en Brews 2000 [6]. La representación de la Figura 2.5 muestra la conformación general de un ejemplar **canal N**, el cual se construye sobre un sustrato semiconductor con dopaje tipo P, en que se crean, mediante difusión, dos zonas con dopaje n+. Sobre la región que se encuentra entre ellas se ubica la compuerta, un contacto eléctrico aislado del semiconductor por medio de una capa de óxido de silicio. Típicamente, el sustrato se cortocircuita a la fuente.

Al aplicar un voltaje positivo entre la fuente y la compuerta no se produce conducción entre ellos, pues la compuerta está físicamente aislada del semiconductor. Sin embargo, si se aplica una tensión positiva en la compuerta (respecto a la fuente, que está cortocircuitada con el sustrato), se produce una acumulación de electrones producto de la atracción eléctrica generada por esta diferencia de potencial. Si este voltaje es lo suficientemente grande, la acumulación de electrones forma un canal que une ambas zonas de tipo N, donde los portadores mayoritarios son los electrones, permitiendo así la conducción eléctrica entre el drenador y la fuente. Es claro que al quitar el voltaje entre la compuerta y la fuente (V_{GS}), la conducción desaparece inmediatamente al desaparecer el canal.

De lo anterior se deduce que el tamaño del canal dependerá fuertemente del nivel de tensión aplicado a la compuerta, ya que mientras mayor sea, más electrones habrá en el canal, condicionando así la corriente que puede circular. Además, existe un efecto capacitivo de influencia no menor en muchos casos, producto de la aislación con óxido de silicio, el cual puede ser crítico para alcanzar conmutaciones rápidas.

² Metal Oxide Semiconductor Field Effect Transistor

Estas características le dan al MOSFET su excelente desempeño en cuanto a velocidad de conmutación y bajas pérdidas, pero hacen necesario el estudio de los fenómenos involucrados para el diseño de un adecuado circuito de encendido.

2.4.1 Resistencia de Conducción

Al estar encendido, el MOSFET presenta un comportamiento resistivo entre el drenador y la fuente, el cual se define por el parámetro R_{DS} , cuyo valor varía según las condiciones de operación.

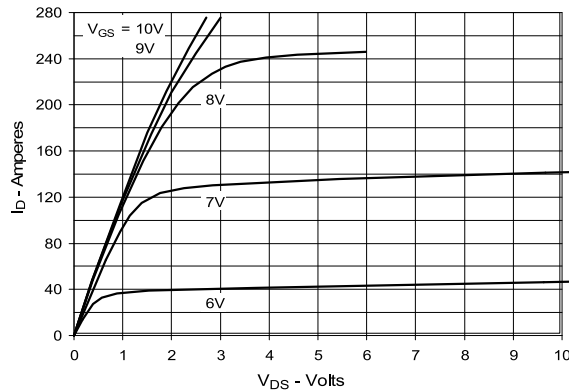


Figura 2.6: Características de salida del transistor IXTH130N10T [7]

La Figura 2.6 muestra la relación entre la corriente de drenador (I_D) y el voltaje entre el drenador y la fuente (V_{DS}) del transistor IXTH130N10T de IXYS Corporation [7], para diferentes valores de tensión en la compuerta (V_{GS}). La pendiente de la curva define la resistencia R_{DS} . Para un valor dado de V_{GS} se observa un comportamiento lineal, es decir resistivo, hasta el punto en que el canal formado se ocupa a plena capacidad, limitando la corriente y aumentando la potencia disipada por el transistor.

Debido a lo anterior, es fundamental la aplicación de una tensión de compuerta apropiada para la corriente que se requiere, de modo de minimizar las pérdidas de conducción.

Otro factor a considerar, para determinar las tensiones usadas en los disparos de un MOSFET, es el voltaje de umbral en la compuerta. Dicho umbral corresponde a la tensión mínima necesaria para formar el canal, por lo que una señal de apagado debe estar bajo este umbral y una de encendido sobre el mismo.

2.4.2 Velocidad de Conmutación

Las capacidades parásitas presentes entre la compuerta y los otros dos terminales del transistor condicionan la velocidad de encendido, ya que para alcanzar la diferencia de potencial capaz de formar el canal deseado, es necesario entregar una cierta cantidad de carga. El tiempo que tarde en acumularse dicha carga determinará el tiempo de conmutación del MOSFET.

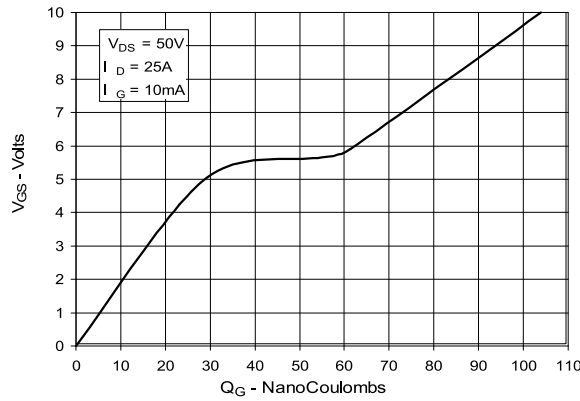


Figura 2.7: Carga de la compuerta del transistor IXTH130N10T [7]

En la Figura 2.7 se muestra la tensión de compuerta alcanzada según la cantidad de carga entregada para un transistor en particular. Dicho valor se vincula con la velocidad de conmutación a través de la siguiente relación:

$$\frac{Q_G}{\Delta t} = I_{con} \quad (2.5)$$

luego, para alcanzar cierto nivel de carga (Q_G) en un intervalo definido de tiempo (Δt), el circuito de encendido debe ser capaz de entregar una corriente de conmutación (I_{con}) tal que se cumpla la ecuación (2.5).

Una vez alcanzado el voltaje deseado en la compuerta, no se requiere más flujo de corriente para mantener el canal, exceptuando pequeñas corrientes de fuga, debido a la aislación eléctrica del óxido de silicio.

Para cumplir con estas especificaciones, existe en el mercado una gran variedad de circuitos de disparo o *drivers*, los cuales están diseñados para proporcionar estas altas corrientes por pequeños períodos y mantener la tensión de compuerta en un valor apropiado, según lo expuesto por Rashid 2001 [4].

2.4.3 Uso en Paralelo de Transistores MOSFET

En las aplicaciones de potencia, la utilización de transistores MOSFET en paralelo tiene por objetivo distribuir la corriente a través de ellos, reduciendo así las pérdidas de conducción, mejorando las características térmicas y permitiendo el uso de transistores más pequeños y económicos. Sin embargo, es necesario considerar algunos fenómenos que pueden presentarse con esta configuración, capaces de anular sus ventajas o incluso generar una falla importante.

El primer aspecto a tomar en cuenta es el balance de las características eléctricas, tanto de los semiconductores como del circuito del cual forman parte, puesto que, si bien la igualdad de voltajes está asegurada por la conexión en paralelo, esto no es necesariamente cierto durante los transientes, debido a las inductancias parásitas presentes en el circuito que pueden estar desbalanceadas [8].

Durante el apagado, un desbalance en las inductancias en serie de los respectivos drenadores puede producir diferencias de tensiones importantes, con el consiguiente riesgo de sobretensión en el drenador de uno de los transistores. Debido a lo anterior, es fundamental reducir a un mínimo las inductancias, tanto en los colectores como en las fuentes de los transistores.

Uno de los métodos efectivos para reducir estas inductancias es el uso de un bus laminado, compuesto por sucesivas capas de cobre aisladas entre sí, cada una de las cuales constituye un nodo del circuito de potencia, como se describe en Allocco 1998 [9]. A través de perforaciones correctamente ubicadas y alineadas, se conectan a estas placas los componentes electrónicos, reduciendo al mínimo la inductancia del nodo gracias a la gran superficie de cobre. Adicionalmente, el apilamiento de las láminas favorece un diseño compacto y robusto.

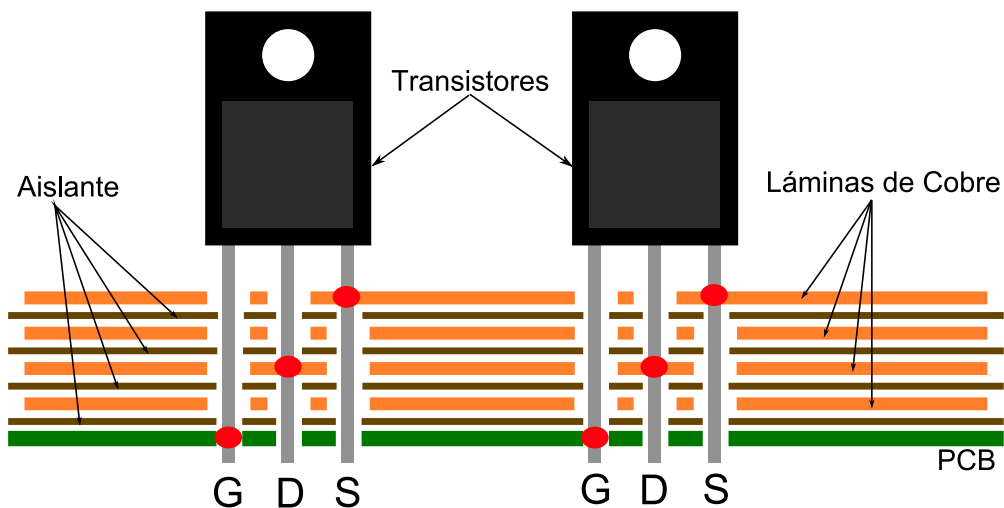


Figura 2.8: Diagrama de conexión para dos transistores en paralelo.

En la Figura 2.8 se muestra un corte transversal de las conexiones para dos MOSFET en paralelo. El número de láminas de cobre dependerá de los nodos directamente conectados a los transistores, en este caso cuatro, pues corresponde a un puente H cuyos nodos son los dos de alimentación y los dos de salida.

Aplicado a la construcción de un puente H, el bus laminado reduce además la inductancia entre el banco de condensadores y los transistores (ver Figura 2.4). Esto contribuye a minimizar las sobretensiones que se pueden generar producto de los cortes abruptos en la conducción de los MOSFET.

Con las inductancias parásitas debidamente manejadas, aún es posible que se presente un comportamiento anómalo al momento del encendido de los transistores. Esto se debe a que la conexión en paralelo puede generar un camino de baja impedancia para una frecuencia de resonancia, determinada por las características de los transistores y sus conexiones, tal como se muestra en la Figura 2.9 [10].

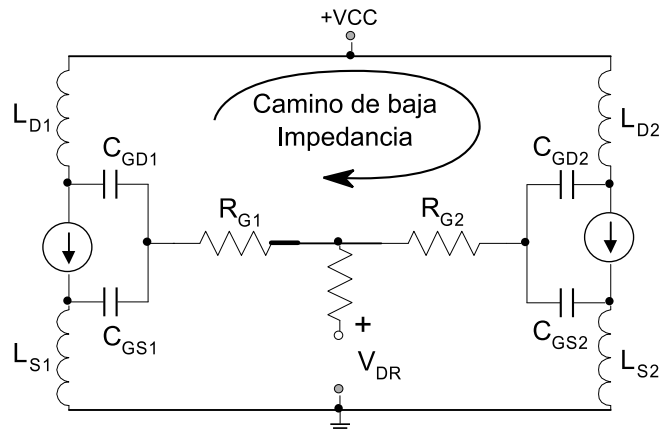


Figura 2.9: Circuito equivalente de dos MOSFET conectados en paralelo. Se muestra el camino de baja impedancia para valores pequeños de resistencia de compuerta.

La existencia de este camino de baja impedancia puede generar oscilaciones parásitas durante el encendido o el apagado, las cuales son de muy alta frecuencia, típicamente en el rango de los 50 MHz a 250 MHz. Dicha oscilación es inaceptable ya que puede causar sobretensiones en la compuerta, emisión de ruido de radio frecuencia, altas pérdidas de conmutación e incluso oscilaciones incontrolables que pueden derivar en la destrucción de los semiconductores [10].

Este fenómeno puede ser de naturaleza sumamente intermitente, dependiendo fuertemente de las condiciones del circuito del cual son parte los transistores, incluso las puntas de prueba pueden eliminarlo, dificultando su detección. Es por esta razón que resulta indispensable considerarlo al momento de diseñar los circuitos de disparo para dispositivos que utilicen esta configuración [10].

El incremento de la resistencia de compuerta para cada transistor (R_{G1} y R_{G2} en la Figura 2.9) amortigua eficazmente las oscilaciones. Desafortunadamente esto también reduce la velocidad de las conmutaciones, por lo que es posible que una resistencia capaz de eliminar las oscilaciones genere pérdidas de conmutación inaceptablemente altas.

El uso de un núcleo de ferrita, combinado con una resistencia en cada compuerta, es capaz de eliminar la oscilación minimizando las pérdidas de conmutación, ya que la impedancia del núcleo de ferrita es directamente proporcional a la frecuencia. El ancho de banda de la señal de encendido en la compuerta es de alrededor de 2MHz, mientras que las oscilaciones parásitas usualmente se encuentran entre los 50MHz y los 150MHz. Luego, la impedancia del núcleo de ferrita es de 25 a 125 veces mayor para las oscilaciones parásitas, que para la señal de encendido [10].

2.5 UNIDAD DE MEDICIÓN INERCIAL (IMU)

Una Unidad de Medición Inercial o IMU (por las siglas en inglés de *Inertial Measurement Unit*) es un dispositivo electrónico cuyo objetivo es obtener mediciones de velocidad, rotación y fuerzas gravitacionales en forma autónoma. Se utilizan como componentes fundamentales en los sistemas de navegación de barcos, aviones, helicópteros, misiles o cualquier móvil en que sea necesario estimar estas mediciones, sin la posibilidad de utilizar referencias externas o mediciones directas (Por ejemplo, en el caso en que no se cuente con odometría o dicha información sea insuficiente para estimar las variables de interés).

Típicamente, una IMU está compuesta por un conjunto de acelerómetros y giróscopos, que obtienen datos de uno o más ejes ortogonales (dependiendo de los requerimientos del sistema), enviándolos a algún sistema computarizado que realiza los cálculos necesarios para obtener las estimaciones de aceleración y velocidad de rotación requeridas.

En la Figura 2.10 se indican las mediciones que realiza una unidad IMU de un avión en vuelo. Se mide tanto la inclinación del avión, como la velocidad de rotación en cada uno de los ejes.

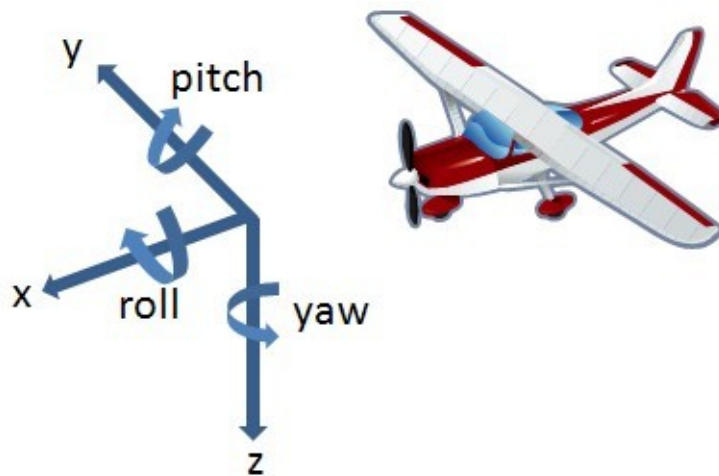


Figura 2.10: Mediciones de una unidad IMU de 3 ejes aplicada a un avión [11].

Debido a que el vehículo es un sistema que esencialmente pivota libremente en un eje horizontal, es indispensable contar con una unidad IMU para estimar dicho ángulo de rotación y aplicar una acción de control adecuada para mantener la estabilidad.

Capítulo 3: Evaluación Inicial y Propuesta de Mejoramiento

3.1 ESTADO INICIAL DEL VEHÍCULO.

Al momento de iniciar el presente trabajo, se contaba con el vehículo completamente armado y funcional, permitiendo evaluar su estado y así tener una línea base de comparación para los cambios que se hicieron posteriormente. Sólo fueron necesarias reparaciones menores y un pequeño mantenimiento para someterlo a un intensivo programa de pruebas. En la Figura 3.1 se muestra al vehículo en su estado inicial.



Figura 3.1: Estado inicial del vehículo.

3.1.1 Evaluación Técnica

En conjunto con el programa de pruebas con usuarios, al que fue sometido el vehículo, se profundizó en los aspectos técnicos relativos a la electrónica y la estrategia de control que inicialmente disponía el vehículo.

Aspectos Mecánicos

Los componentes electrónicos, las baterías y los motores del vehículo están contenidos en un esqueleto de pletina de acero soldado. Dicho esqueleto fue diseñado para quedar completamente dentro del diámetro de las ruedas, para así proteger los componentes electrónicos ante un eventual impacto.

Al inicio de este trabajo, el esqueleto de acero ya cumplía su función al brindar una estructura sólida y confiable para el montaje de los diversos componentes. En la fotografía de la Figura 3.2 se puede ver el estado inicial de la estructura de acero. Se quitaron las baterías y los motores del vehículo, con el objetivo de apreciar de mejor manera la electrónica en su interior.



Figura 3.2: Estado inicial del esqueleto de acero.

Se aprecia que el metal se encontraba sin pintura, de un color opaco, con visibles y notorios cordones de soldadura, lo cual indica que luego de la fabricación y ensamblaje no se alcanzó a dar tratamiento alguno al metal. Como consecuencia de lo anterior, también existían algunas puntas potencialmente cortantes y virutas resultantes de las perforaciones, lo cual constituía riesgo de lesiones para quien trabajase en él.

El control de dirección se realizaba mediante el giro de un manillar ubicado en la parte superior del mando de dirección. El peso de dicho mecanismo, sumado a su ubicación en la parte más alta del vehículo, inclinaba naturalmente todo el equipo hacia adelante, produciendo el avance espontáneo del vehículo cuando se encontraba encendido y sin pasajero, puesto que el controlador buscaba estabilizar el peso de la parte superior.

Tarjeta de Control

Toda la operación del vehículo era manejada por una tarjeta de control, la cual usa un DSP perteneciente a la familia C2000 de Texas Instruments como procesador central, específicamente el modelo TMS320F2808 [12]. Dicho dispositivo, orientado específicamente al control de motores y sistemas de potencia, posee módulos capaces de generar PWM por hardware, conversores análogo-digital, módulos de comunicaciones y otros periféricos que permiten realizar una gran variedad de tareas en tiempo real. La CPU es de punto fijo, con una frecuencia de reloj de 100MHz.

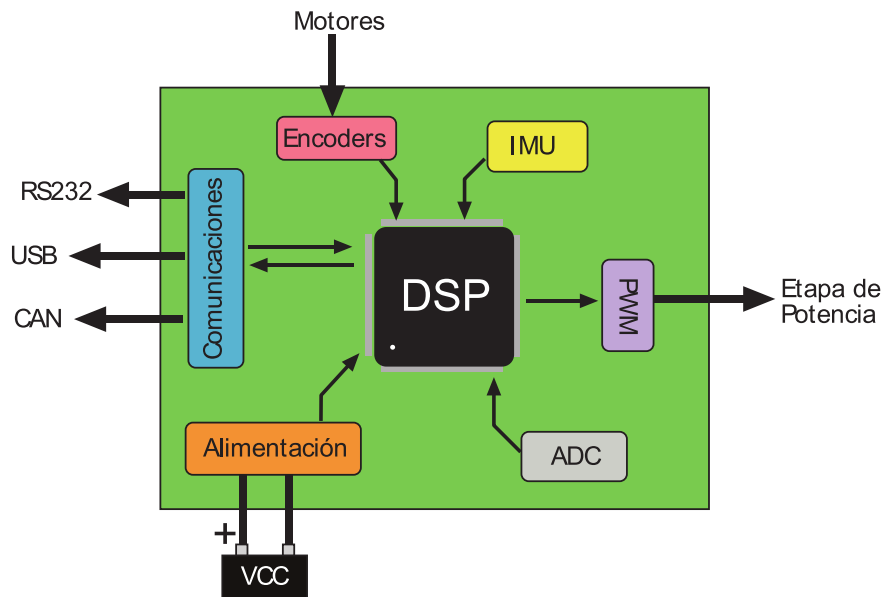


Figura 3.3: Diagrama de bloques de la tarjeta de control original [2].

La Figura 3.3 muestra un diagrama de bloques de la tarjeta de control original. En ella está el DSP antes mencionado, así como la unidad IMU (Inclinómetro y Giróscopo). También posee la electrónica necesaria para enviar los pulsos generados por los módulos PWM a los puentes H, diversos puertos de comunicaciones, entradas analógicas y de encoders para medir la velocidad de los motores. Destaca también una conexión USB realizada a través de un circuito integrado conversor USB a Serial, que permite una lectura directa desde un computador.

En general, la tarjeta de control cumplía su función sin grandes problemas. Sólo fue necesario reparar algunas soldaduras y verificar las conexiones para la realización de pruebas iniciales.

Puentes H

Los dos puentes H que controlaban la operación de los motores funcionaban correctamente al inicio de este trabajo. Sin embargo, este estado se logró luego de varias iteraciones posteriores al diseño inicial, durante el desarrollo del trabajo de Moreno [2], en las que se realizaron los siguientes cambios:

- Los transistores de la parte inferior del puente H (Figura 2.4) no contaban con drivers adecuados para su encendido, por lo que fueron agregados en un circuito adicional a la tarjeta electrónica.
- El diseño original de la tarjeta no consideraba los diodos supresores de transiente incorporados posteriormente. Éstos se encontraban soldados por encima de los componentes de la tarjeta y aislados con silicona.
- Las tarjetas no disponían de un circuito para medir la corriente entregada a los motores. En su lugar se instalaron sensores de corriente directamente a los cables, sin un soporte firme y con conexiones susceptibles al ruido electrónico inducido por la operación de los transistores.
- Cada puente H cuenta con una inductancia, cuyo objetivo es filtrar la corriente de entrada, que inicialmente se encontraba sujeta junto a la tarjeta electrónica conectada con cables.

Adicionalmente, se observó que los circuitos de disparo no incorporaban ningún tipo de protección para evitar un encendido accidental que podría generar un corto circuito, dejando el sistema expuesto a importantes fallas de existir algún problema de software o de conexiones.

Se estudió también el desempeño de los puentes H en cuanto a tiempos de conmutación, con el fin de determinar la necesidad de cambios en la electrónica de disparo.

Durante el tiempo que duran las conmutaciones, los transistores disipan gran cantidad de potencia puesto que, tanto tensiones como corrientes son mayores que cero. Es por este motivo que la reducción de este tiempo impacta sensiblemente en el comportamiento térmico del dispositivo.

Para estos puentes H se midió el voltaje compuerta-fuente en los MOSFET mientras los motores eran accionados. Los resultados se encuentran especificados en la Figura 3.4.

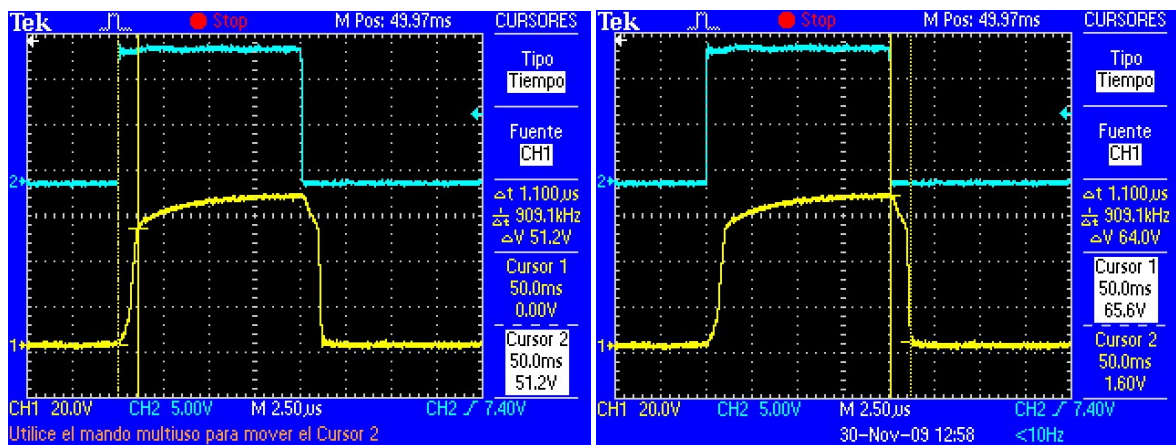


Figura 3.4: (Izquierda) Flanco de subida del pulso en la compuerta del transistor. (Derecha) Flanco de bajada del pulso en la compuerta del transistor.

En la Figura 3.4 se observa una captura de un pulso en la compuerta de uno de los transistores del puente H. La señal de arriba corresponde a los pulsos en el circuito lógico, mientras que la de abajo corresponde a la punta de prueba ubicada en el transistor. El tiempo que toma el encendido es el mismo que toma el apagado, de alrededor de $1,1\mu$ S. Dado que la frecuencia de la señal PWM es de 20kHz, las conmutaciones utilizan el 4,4% del tiempo de cada ciclo, valor que se desea reducir al mínimo posible, ya que es en este proceso en que los transistores disipan la mayor cantidad de potencia. Estos tiempos son factibles de reducir, puesto que en la hoja de datos de los transistores utilizados [13], se indican tiempos de encendido y apagado de 140nS y 100nS respectivamente.

Estrategia de Control

En el DSP se encontraba implementado un algoritmo de control proporcional derivativo (PD), siguiendo el enfoque planteado en el trabajo de Moreno [2].

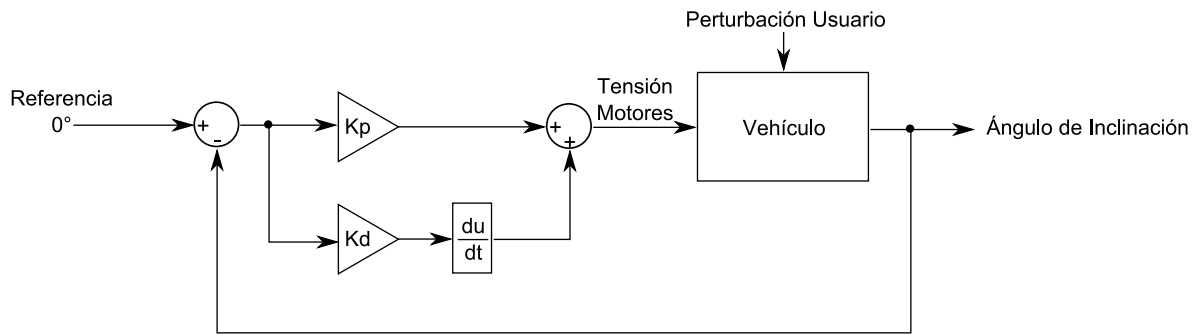


Figura 3.5: Esquema de control presente en el vehículo.

Como muestra la Figura 3.5, el controlador busca alcanzar la referencia de 0° de inclinación, aplicando una tensión a los motores, tal que reduzca el error entre el ángulo de inclinación medido y la referencia. Dicha tensión se calcula de manera proporcional a la inclinación y a su velocidad, con constantes determinadas convenientemente.

En el código, el controlador calcula su salida según el punto de operación, aplicando un modelo lineal para ángulos pequeños y aproximaciones de mayor orden para ángulos más grandes. Además, la derivada de la inclinación no es calculada, sino obtenida directamente desde el giróscopo, realizando los ajustes necesarios.

Cabe destacar que, según se plantea en el trabajo de Moreno [2], la acción de control aplicada al sistema debe ser un torque sobre las ruedas. Sin embargo, en este caso la acción de control es un voltaje aplicado a los motores por medio de los puentes H, el cual no se relaciona directamente con el torque, puesto que éste depende además de otras variables.

A pesar de lo anterior, la acción de control igualmente tiende a llevar el sistema a la referencia, aunque con una dinámica diferente a la modelada.

Aspectos Generales

Además de los aspectos detallados anteriormente, el vehículo contaba con una fuente conmutada de tipo *step-down*, que alimentaba los componentes electrónicos con una tensión estable a partir de las baterías, con suficiente capacidad de corriente y con la eficiencia necesaria para no subir su temperatura durante la operación.

La tarjeta electrónica encargada de comandar el visor LCD, montado en la parte superior del vehículo, también funcionaba correctamente y sólo requería un mecanismo más firme de sujeción al manubrio.

En términos de autonomía, el vehículo mostró un buen desempeño al funcionar por varias horas de forma continua, sin problemas. Constructivamente también se obtuvo un resultado satisfactorio, sin problemas mecánicos de ningún tipo.

3.1.2 Pruebas con Usuarios

Ya que el objetivo es conseguir que el vehículo sea apto para gente común, sin necesidad de experiencia o gran habilidad motriz, se invitó a 15 personas que no habían usado el vehículo anteriormente para que lo probaran. Sus impresiones fueron capturadas mediante un formulario en el que evaluaron con calificación del 1 al 10 distintos aspectos del vehículo. Los cuyos resultados promedio están contenidos en la Tabla 3.1:

Aspecto Evaluado	Calificación
Facilidad de Aprendizaje	4,6
Manejo Intuitivo	6,4
Sensación de Seguridad	3,3
Potencia	4,9
Velocidad	4,7
Calificación General	5,4

Tabla 3.1: Resultados de las encuestas aplicadas a usuarios novatos, utilizando el diseño inicial del vehículo. El detalle de las evaluaciones se encuentran en el Anexo B.

Como se puede ver, únicamente el manejo intuitivo y la calificación general fueron evaluados sobre la mitad de la escala, siendo la sensación de seguridad el aspecto más débil. Entre los comentarios recibidos está la dificultad para iniciar y detener la marcha, así como las sacudidas que experimenta el vehículo por momentos, las cuales casi en todos los casos desestabilizan a quien maneja. A lo mencionado se puede agregar falta de potencia, la sensación de que el sistema de control no responde con la vigorosidad necesaria, ya que el usuario se inclina y no tiene la suficiente fuerza para compensar.

Desde el punto de vista general, fue positivo el hecho de que se hizo funcionar el vehículo intensamente durante varias horas, con muchas personas, sin que las baterías se agotaran ni se soltaran piezas, lo que da muestra de la fortaleza estructural y buena construcción que se hizo en este primer diseño. La recarga de las baterías, luego de la jornada más intensa, se hizo en alrededor de 2 horas, con una fuente de laboratorio a poco menos de 3A de corriente constante y a los 14,4V de tensión constante que define el fabricante de las baterías para la carga cíclica.

3.2 MEJORAS A REALIZAR.

En este trabajo se postula que es posible mejorar de forma notable el desempeño del vehículo ya construido, aplicando modificaciones en la instrumentación, electrónica y sistema de control.

Se busca utilizar y aprovechar todos los conocimientos adquiridos en el diseño anterior, consolidándolos en un diseño definitivo que incorpore además las mejoras que se desean aplicar.

El alcance de la intervención propuesta involucra a todos los aspectos del vehículo, usando como base la configuración mecánica y la topología de la electrónica, pero generando nuevas piezas de hardware y reformulando completamente el software asociado.

Específicamente, las mejoras a realizar son las siguientes:

- Reemplazar los puentes H existentes por unos nuevos, con mejores características eléctricas y mayor seguridad, reemplazando drivers, agregando protecciones e integrando los componentes añadidos a los puentes H originales.
- Diseñar una unidad IMU nueva, capaz de utilizar múltiples instrumentos, para tener redundancia, en una tarjeta electrónica independiente a la de control, conectada mediante una interfaz digital.
- Adaptar el diseño de la tarjeta de control para acoplarse a los cambios realizados en el resto de la electrónica, incorporando medidas para minimizar el efecto del ruido inducido.
- Reemplazar el pesado manillar de la parte superior por un manubrio fijo y liviano, cambiando el sistema de dirección para poder controlar el vehículo con movimientos laterales del cuerpo.
- Mejorar la seguridad y la estética del vehículo, eliminando los bordes cortantes, aplicando un tratamiento adecuado a la estructura y agregando una cubierta para proteger a los componentes internos de la suciedad y los golpes directos.
- Reestructurar y reescribir el software de control, ajustándolo a las necesidades específicas de la aplicación, pero sin perder versatilidad y claridad en código, de manera de facilitar su ajuste e intervención posterior en trabajos futuros.
- Aplicar una estrategia de control adecuada, que permita un manejo más cómodo y estable.

Capítulo 4: Diseño e Implementación de las mejoras

En este capítulo se aborda de manera detallada los cambios y mejoras realizadas en el vehículo. Se explicitan las consideraciones de diseño y los criterios empleados en la definición de los nuevos sistemas, resaltando los aspectos más relevantes para alcanzar los objetivos planteados.

En términos generales, el vehículo es un sistema cuyo funcionamiento está definido por tres aspectos fundamentales: el mecánico, determinado por los componentes estructurales, mecanismos de accionamiento y soporte físico para los sistemas; el electrónico, compuesto por los circuitos encargados de accionar el vehículo y dar soporte al algoritmo de control; y finalmente el software, encargado de administrar los demás aspectos para conseguir un determinado comportamiento.

Las mejoras implementadas abarcan los tres aspectos fundamentales, sin embargo, las mayores intervenciones se realizaron en los aspectos electrónicos y de software.

4.1 MEJORAS EN EL DISEÑO MECÁNICO

En cuanto la parte mecánica, el vehículo fue intervenido para mejorar aspectos estéticos, prácticos y funcionales entre los que destacan: un nuevo sistema de dirección, reorganización de componentes electrónicos y mejoras en cuanto a facilidad de mantenimiento y seguridad de quien lo realiza.

Es por esto que se decidió pulir completamente el metal con una herramienta abrasiva, para eliminar las terminaciones en punta, los rastros de virutas y toda la suciedad remanente de las soldaduras donde fue posible (ver Figura 4.1). Luego de limpiar el metal, se aplicó una capa de pintura anticorrosiva negra para protegerlo, posteriormente se usó esmalte azul y una capa de laca acrílica para proteger la pintura y mejorar el aspecto.

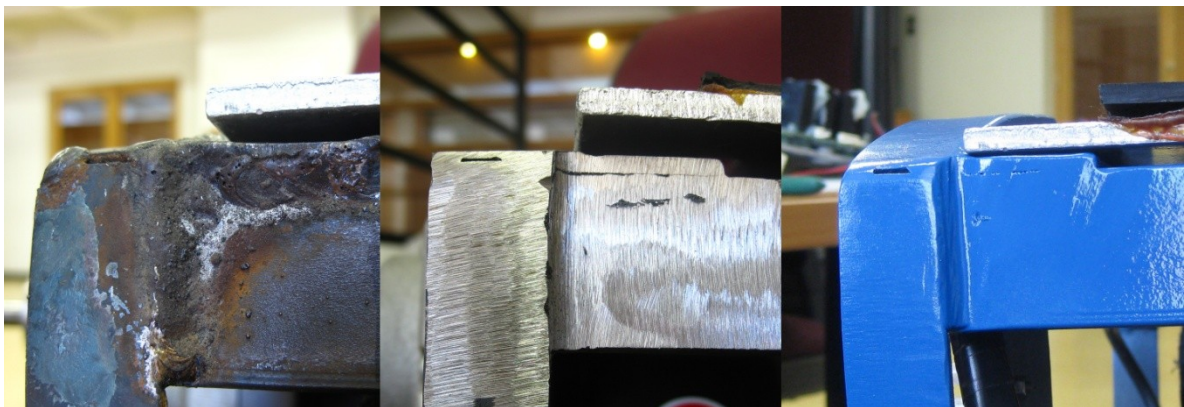


Figura 4.1: Tratamiento aplicado al esqueleto de acero del vehículo.

En la Figura 4.1 se observan las etapas del tratamiento descrito anteriormente. Con esto se consiguió un mejor aspecto y mucho más seguro de manipular al eliminar todos los bordes filosos.

4.1.1 Sistema de dirección

La intervención estructural realizada de mayor impacto en la experiencia del usuario es el cambio en el sistema de dirección. Originalmente, el usuario controlaba el giro del vehículo por medio de un manillar ubicado en la parte superior de una columna rígida sujeta a la base de acero.

Con el objetivo de hacer más natural la conducción y mejorar la estabilidad del vehículo sin pasajero, se reemplazó el manillar de la parte superior de la columna de dirección por un manubrio de bicicleta de aluminio mucho más liviano y sin partes móviles. Se reemplazó la fijación a la base de acero por una articulación que permite inclinar toda la columna de dirección hacia los lados, siguiendo la inclinación natural del usuario al efectuar un giro, tal como se muestra en la Figura 4.2.

Para realizar esta modificación se utilizó la pieza de aluminio existente para la fijación del mando de dirección, incorporándole un rodamiento, un resorte y el potenciómetro para medir la inclinación. Finalmente, se soldó a la estructura de acero un eje para acoplar el rodamiento junto con dos soportes verticales, destinados a aumentar la rigidez estructural y proporcionando los topes necesarios para el resorte de retorno.

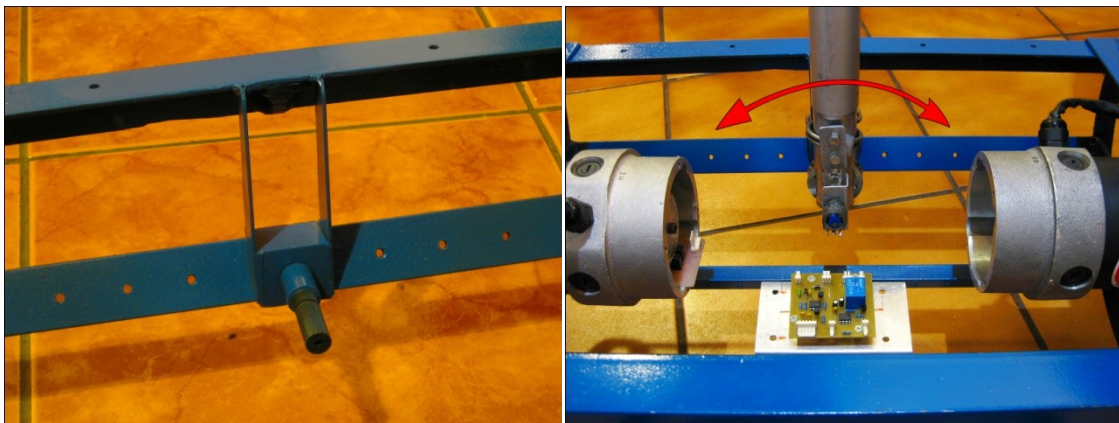


Figura 4.2: (Izquierda) Modificación a la estructura de acero para el montaje de la nueva dirección. (Derecha) Sistema ensamblado, se muestra la dirección de movimiento del mecanismo.

4.1.2 Montaje de componentes

Producto del rediseño de la electrónica del vehículo, se fabricaron nuevas tarjetas con distintos tamaños y elementos, haciendo necesario reevaluar su distribución física. Esto permitió también mejorar la confiabilidad y durabilidad al tener en cuenta aspectos como la reducción del ruido en las señales y los esfuerzos mecánicos a los que estas tarjetas están sometidas.

En el diseño original, tanto la electrónica de potencia como la de control se encontraban concentradas en la parte central de la base del vehículo, en un espacio de aproximadamente 12cm de ancho entre las baterías y los motores. Las tarjetas electrónicas se sostenían de la cubierta superior (plataforma para los pies del usuario) por medio de soportes de acrílico atornillados a la misma, tal como se puede ver en la Figura 4.3. El uso de largos cables hacia una bornera de distribución permitía retirar la cubierta y ubicarla a un lado del vehículo sin necesidad de desconectarla para realizar ajustes y mantenimiento.

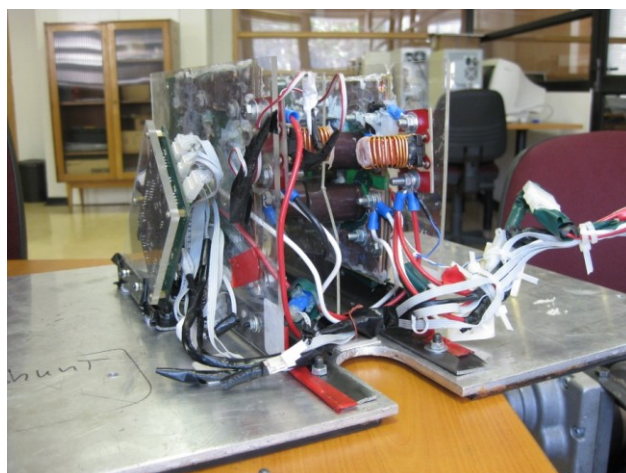


Figura 4.3: Disposición original de los componentes electrónicos.

Los puentes H son las tarjetas de mayor tamaño y fueron diseñadas para ocupar el mismo espacio que las anteriores. Se fabricó un nuevo sistema de montaje para ellas, fijándolas a la estructura de acero de la base en vez de la plataforma para los pies (ver Figura 4.4). Se utilizaron láminas de aluminio de 1,2 mm de espesor para los soportes, los cuales a su vez cumplen la función de pantalla del ruido electromagnético generado por las conmutaciones de los semiconductores. Con esta configuración, se evita también que los componentes y sus cables se muevan cada vez que se quita la cubierta superior, reduciendo la probabilidad de fallas por desconexiones, cortocircuitos o fatiga de material.

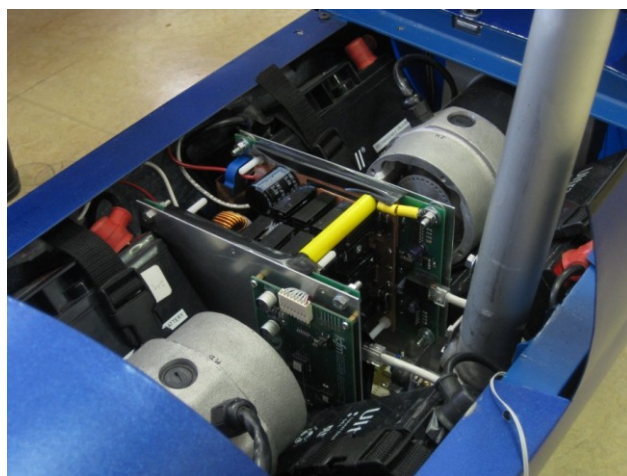


Figura 4.4: Montaje final de componentes electrónicos en la parte central del vehículo.

En la Figura 4.4 se aprecia la manera en que las nuevas tarjetas electrónicas quedaron dispuestas en los soportes de aluminio, los cuales no se ubicaron en el centro del espacio disponible, sino que se buscó llevar la electrónica hacia la parte trasera, con el fin de compensar el peso del mando de dirección, proporcionando además el espacio necesario para los componentes de su nuevo mecanismo. En la parte interior, entre las láminas, se ubicaron los puentes H, mientras que en la parte exterior sólo se ubicó la tarjeta con los instrumentos de la Unidad de Medición Inercial. Finalmente, bajo los puentes H, se ubicó una tarjeta de lecturas analógicas (visible en la Figura 4.2), cuya función se detallará más adelante.

La tarjeta de control se ubicó junto a la fuente de poder de la electrónica en la parte externa de la estructura principal, al lado derecho. Nuevamente se utilizaron láminas de aluminio para la superficie plana a la cual se atornillan los separadores que finalmente sostienen las tarjetas. El montaje general de los componentes se puede ver en la Figura 4.5.

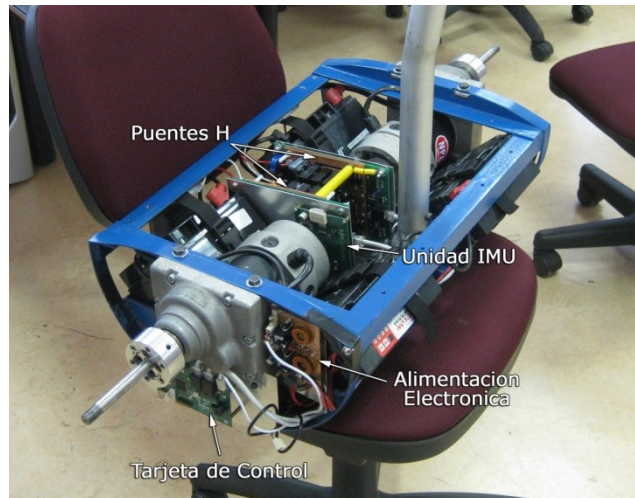


Figura 4.5: Vista general de la disposición final de los componentes internos del vehículo.

4.1.3 Cubierta Exterior

Con el fin de proteger los componentes electrónicos de golpes directos y la suciedad proveniente del exterior, se diseñó una cubierta protectora para toda la base del vehículo, la cual además mejora la apariencia general del equipo.

Dicha cubierta se elaboró a partir de una lámina de hojalata a la cual se le aplicó un proceso conocido como cilindrado, para ceñirse al contorno de la base de acero del vehículo. Este material, además de ser moldeable, es mucho más rígido y resistente que el aluminio.



Figura 4.6: Vehículo con cubierta protectora instalada.

Se decidió pintar la cubierta, tal como se observa en la Figura 4.6, siendo necesaria la aplicación de un tratamiento con ácido muriático a la hojalata para quitarle el recubrimiento de estaño y permitir, de esta manera, fijar la pintura adecuadamente.

Con el diseño de la cubierta, se buscó extender el ancho de la base para así proteger los componentes electrónicos montados en el exterior del marco de acero, logrando además una mejor apariencia. La extensión de la forma cilíndrica, hacia los lados de la parte superior de la base, demarca la zona donde se extendió la cubierta más allá de la estructura de acero y genera superficies suaves, minimizando el riesgo de lesiones por impacto en caso de que el usuario se caiga del vehículo.

En la parte superior del mando de dirección, se agregó una pieza de aluminio con un diseño y pintura consistentes con el diseño de la cubierta exterior, cuyo objetivo es el montaje del visor LCD que informa al usuario sobre el estado del vehículo y los botones de comando, tal como se indica en la Figura 4.7.



Figura 4.7: Perspectiva del usuario del visor LCD y los botones de comando del vehículo.

4.2 MEJORAS ELECTRÓNICAS

El proceso iterativo de implementación, prueba y ajuste del diseño electrónico del trabajo anterior, permitió que se alcanzara un prototipo funcional. Sin embargo, los constantes cambios al diseño original, las correcciones y componentes añadidos después de la fabricación de las tarjetas electrónicas, hicieron que las conexiones se volvieran cada vez más complejas y susceptibles a fallas. Esto, sumado a la incorporación de diversas mejoras en cada uno de los sistemas de control y accionamiento de motores, hizo necesario el rediseño de todas las tarjetas electrónicas involucradas.

Para el rediseño electrónico efectuado, se utilizó como base los circuitos y la topología del sistema planteados en la memoria de Leonardo Moreno [2], modificándola donde se consideró necesario para lograr un funcionamiento seguro y confiable, minimizando las conexiones requeridas y las posibles fuentes de falla.

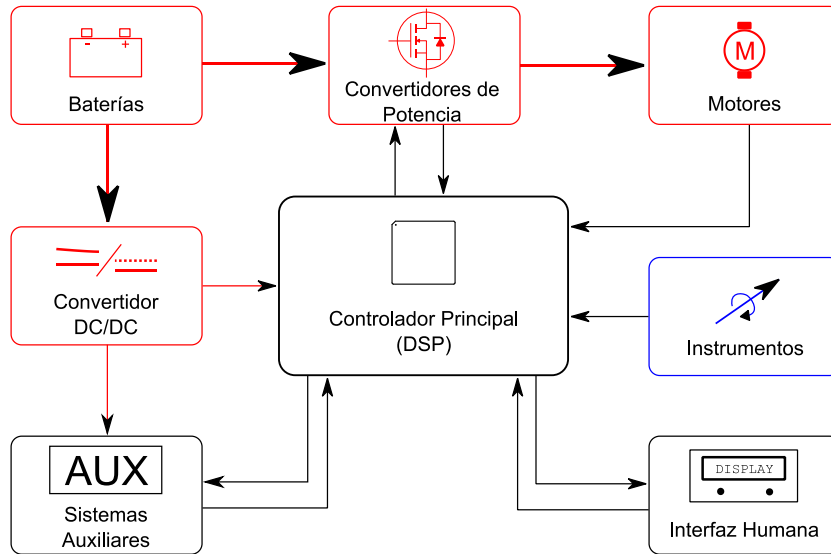


Figura 4.8: Diagrama de bloques de la electrónica de control.

La Figura 4.8 muestra un diagrama de bloques del sistema electrónico. Cada bloque se relaciona con una tarjeta en particular, con excepción de las baterías y los motores.

En esta configuración, todos los componentes están subordinados a la tarjeta de control, cuyo corazón es un DSP³, el cual utiliza la información extraída desde los instrumentos y los sistemas auxiliares para aplicar un determinado esquema de control, bajo la supervisión de una interfaz humana, cuya función es desplegar información y permitir la interacción del usuario con el sistema de control.

El DSP acciona a los convertidores de potencia constituidos por dos puentes H, uno por cada motor, los cuales toman la energía de las baterías y la suministran a los motores de forma controlada.

Las ruedas del vehículo están acopladas a los motores a través de una caja reductora, aumentando así el torque aplicado al sistema.

El diagrama también muestra un convertidor de corriente continua cuya función es alimentar toda la electrónica. Se trata de la misma fuente switching desarrollada en el trabajo anterior [2].

³ Digital Signal Processor

4.2.1 Modificación de la Electrónica de Potencia

Para el rediseño de los puentes H se utilizó como base el diseño existente, concentrando todos los componentes en una sola tarjeta con el mismo tamaño de las existentes, ya que los puentes H son los componentes electrónicos de mayor tamaño y se cuenta con el espacio justo para el montaje de ellos en el vehículo.

Electrónica de Disparo

El primer aspecto abordado en la modificación, fue el cambio en los drivers utilizados para encender los transistores. El diseño original utilizaba el integrado IR2117 [14], compuesto de un único driver capaz de generar una señal de disparo referenciada a un nodo flotante (distinto de la tierra del circuito). Con esto se resolvía el disparo de la parte superior de cada rama del puente H, ya que la tensión de compuerta está referenciada a una de las salidas del puente (ver Figura 2.4).

El IR2117, según su hoja de datos, posee una capacidad de corriente máxima de 200mA para el encendido y 420mA para el apagado, siendo estos valores los que limitan las velocidades de conmutación puesto que, a mayor corriente, más rápidamente se carga la capacidad de compuerta.

El nuevo diseño reemplaza este driver por el IR2110 [15], de la misma familia pero con mejores prestaciones. Este integrado implementa dos drivers diseñados específicamente para operar un medio puente, es decir, una de las ramas verticales del puente H. Posee un driver capaz de generar una señal de disparo referenciada a un nodo flotante (tal como el IR2117) y otro cuya referencia es la tierra del circuito. Cada driver puede entregar hasta 2A de corriente tanto al encendido como al apagado, posibilitando mejoras sustanciales en los tiempos de conmutación. Adicionalmente, la lógica del circuito integrado que incorpora ambos drivers, posee una interfaz de tensiones adaptables, desde 3,3V hasta los 20V, así como una entrada de disparo lógica capaz de deshabilitar completamente ambos drivers.

Para este diseño en particular, se utilizó lógica TTL con 5V para el nivel “alto”, facilitando así la implementación de la segunda mejora importante en el diseño del puente H: La protección contra disparo simultáneo.

Tal como se explicó en el Capítulo 2, de encenderse simultáneamente los transistores de una de las ramas verticales del puente H, se produce un cortocircuito de consecuencias destructivas. Para proteger al sistema de este peligro, se incorporó un sencillo circuito lógico compuesto de una compuerta AND cuyas entradas son las señales de disparo que entran al driver IR2110. Su salida se conecta directamente al pin de deshabilitación de éste como se muestra en la Figura 4.9. De esta manera, si por alguna razón se envían señales de disparo simultáneamente, la compuerta AND desactivará el driver, previniendo por hardware el riesgo de cortocircuito.

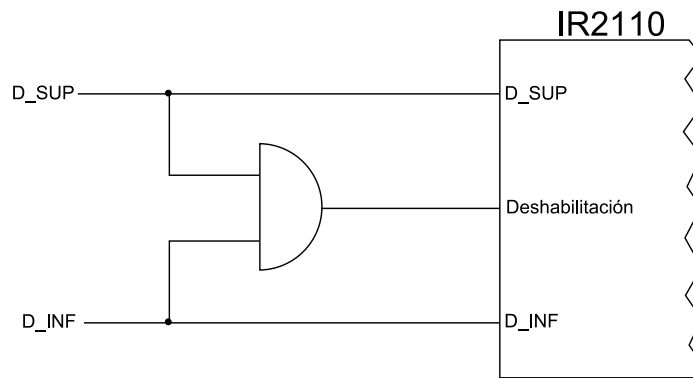


Figura 4.9: Protección contra disparo simultáneo.

Con el fin de asegurar la calidad de las señales de disparo entrantes al puente H, se utilizaron los aisladores digitales ADuM1400 [16], los cuales regeneran la señal proveniente de la tarjeta de control con mínimo ruido y alta velocidad además de proveer aislación eléctrica entre ambos circuitos, protegiendo a la tarjeta de control ante cualquier falla. El circuito integrado implementa cuatro canales de aislación digital ajustándose a los cuatro pulsos de encendido requeridos.

Una de las características de este puente H es la utilización de transistores en paralelo, con el fin de distribuir la corriente entre ellos y reducir la resistencia de conducción, mejorando así las características térmicas y la capacidad de corriente del mismo.

Debido a que uno de los objetivos de las modificaciones es obtener conmutaciones rápidas, se redujo considerablemente la resistencia de compuerta, generando las condiciones propicias para el fenómeno de oscilaciones parásitas descrito en el Capítulo 2. Es por esta razón que se agregó en cada compuerta un núcleo de ferrita, quedando en serie con una resistencia de 15Ω , aumentando la impedancia para las oscilaciones de alta frecuencia pero permitiendo conmutaciones rápidas.

Lectura de Corriente

Además de la funcionalidad propia del puente H, se incorporó en esta tarjeta un transductor de corriente de efecto Hall, capaz de medir con precisión corrientes continuas y variantes en el tiempo para obtener la corriente instantánea sobre cada motor, posibilitando la implementación de un lazo de control.

Dicho sensor de corriente está diseñado para entregar una tensión continua entre 0V y 5V, dependiendo linealmente de la medición con un nivel central de 2,5V que indica los 0A, por lo cual puede medir tanto corrientes positivas como negativas.

Debido a que este transductor se encuentra a distancia considerable de la tarjeta de control y expuesta a altos niveles de ruido eléctrico generados por los transistores del puente H, se incorporó junto al mismo un circuito generador de frecuencia controlada por voltaje o VCO⁴ LM331 [17], el cual genera una onda cuadrada cuya frecuencia depende linealmente del voltaje de entrada que, en

⁴ Voltage-Controlled Oscillator

este caso, es la salida del sensor de corriente. De esta manera, la lectura de corriente se convierte en un tren de pulsos con niveles lógicos mucho menos susceptibles a perturbaciones por ruido eléctrico, con posibilidad de aplicar filtrados más agresivos y otras técnicas de reducción de ruido sin alterar el valor de la lectura.

Desde el punto de vista del diseño de la tarjeta, además de cuidar el tamaño, se prestó atención en simplificar el conexionado, por lo que se incorporaron todas las líneas necesarias para alimentar la electrónica, recibir los pulsos y enviar la lectura de corriente en un único conector del tipo RJ-45, el cual posee ocho conexiones eléctricas.

Bus Laminado

El montaje sobre la tarjeta de los componentes de potencia (MOSFETs, condensadores, inductancia, supresores de transiente, etc.) se hizo sobre un bus laminado (ver Sección 2.4.3) diseñado y construido para minimizar las inductancias parásitas en este circuito. Su construcción se realizó en base al tamaño deseado de la tarjeta, utilizando láminas de cobre de 0,5mm de espesor intercaladas con láminas de un aislante del mismo espesor llamado presspan, utilizado ampliamente en la confección de transformadores por su alta aislación, resistencia a altas temperaturas y naturaleza auto-extinguible. En la Figura 4.10 se muestra el montaje final de los componentes en el puente H.

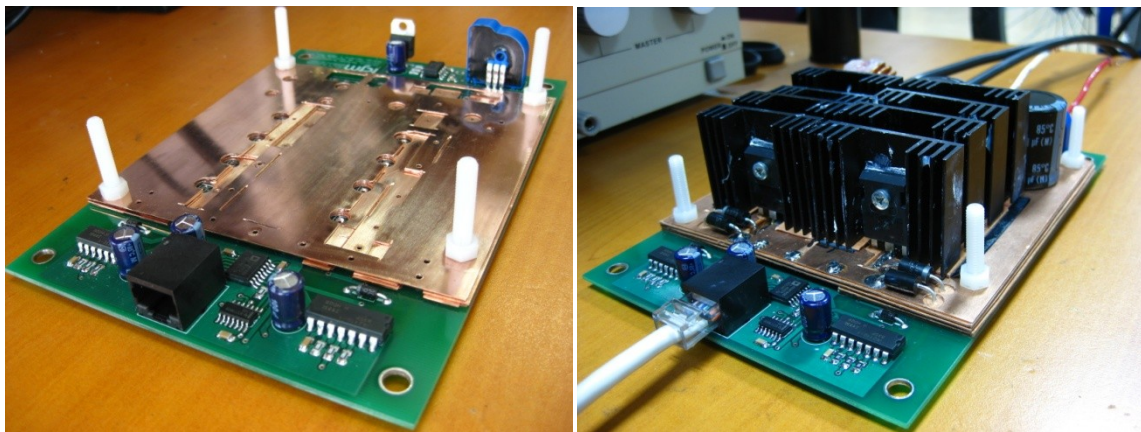


Figura 4.10: (Izquierda) Puente H con el bus plano ensamblado pero sin los componentes de potencia. (Derecha) Puente H completamente armado y funcional.

4.2.2 Cambios en el Sistema de Instrumentación

Para la Unidad de Medición Inercial (IMU) se diseñó una tarjeta electrónica independiente, separando esta parte crítica del sistema de la tarjeta de control e incorporando el nuevo giróscopo de mejor calidad.

Los instrumentos utilizados en esta tarjeta son: Inclinómetro ADIS16203 [18] y Giróscopo ADIS16250 [19], ambos fabricados por la empresa Analog Devices⁵. Su comunicación es por medio de un bus SPI⁶, el cual no implementa direccionamiento en su trama de datos, por lo que la selección del instrumento a leer debe hacerse por medio de señales de habilitación o deshabilitación.

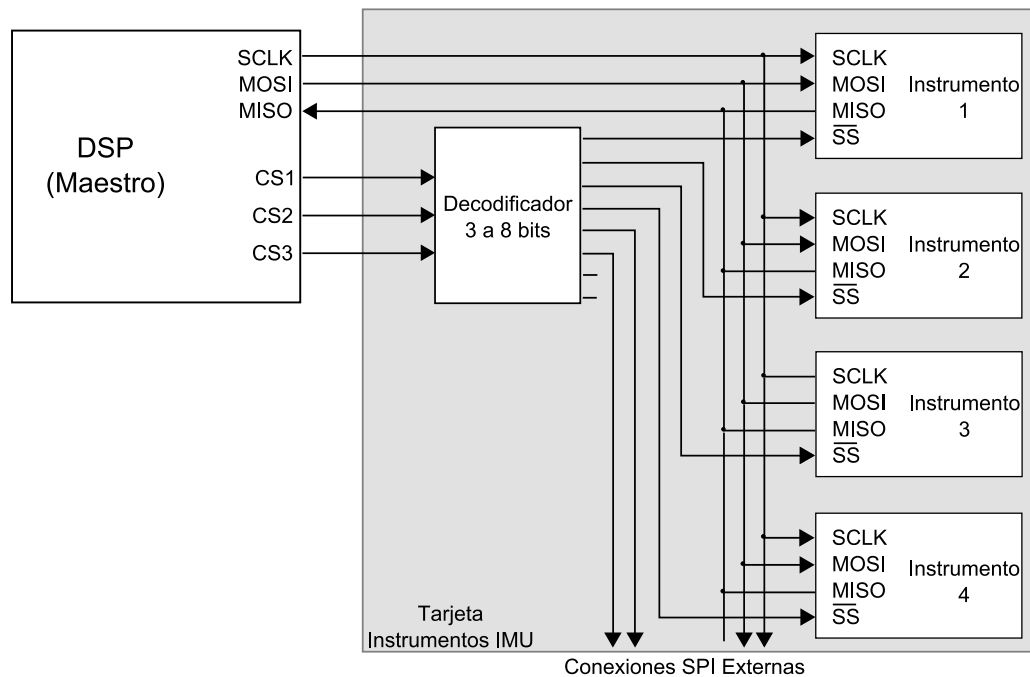


Figura 4.11: Diagrama lógico de la tarjeta de instrumentos IMU.

En la Figura 4.11 se muestra la manera en que se resuelve el direccionamiento al utilizar un decodificador de 3 a 8 bits. La tarjeta fue diseñada para integrar dos acelerómetros (ADIS16203) y dos giróscopos (ADIS16250), así como dos conexiones SPI para una eventual futura expansión. El DSP es el encargado de seleccionar el instrumento o salida SPI adecuada de la tarjeta, por medio de tres bits de dirección. El diseño final de esta tarjeta se muestra en la Figura 4.12.

⁵ <http://www.analog.com/>

⁶ Serial Peripheral Interface

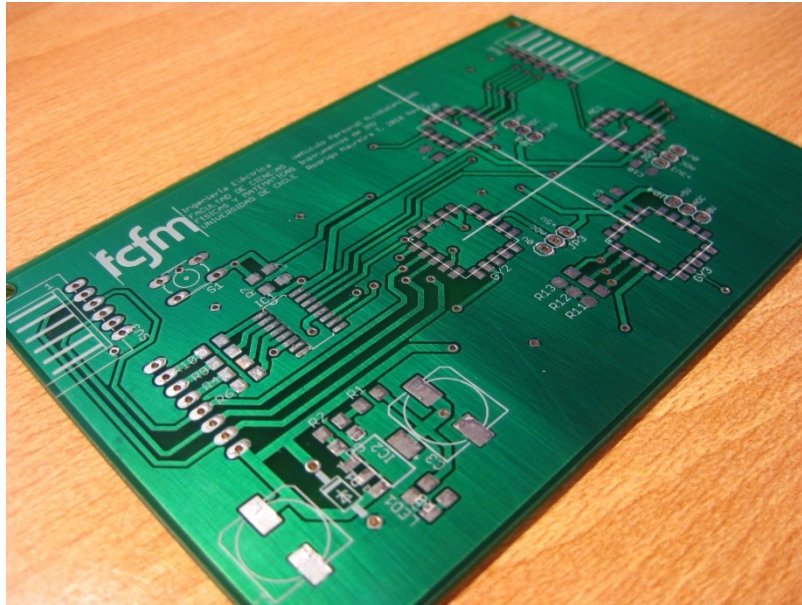


Figura 4.12: Tarjeta de instrumentos IMU previa al montaje de sus componentes.

4.2.3 Tarjeta de control (DSP)

La tarjeta de control fue rediseñada principalmente para adaptarse a la nueva configuración de componentes, utilizando un número mínimo de conectores y reemplazando completamente la etapa de envío de señales PWM a los puentes H.

La tarjeta de control original contenía parte de la electrónica de disparo, con una configuración de dos inversores en cascada, cuyo propósito era elevar la tensión de los pulsos enviados hasta los 15V, tensión utilizada para encender los transistores de los puentes H. Debido a que los nuevos puentes H incorporan toda la electrónica de disparo, la configuración de doble inversor se vuelve innecesaria. Adicionalmente, se buscó aprovechar la utilización de los aisladores digitales ADuM1400 [16] en los puentes H, instalando aisladores del mismo modelo en la tarjeta de control, aprovechando así las características de este dispositivo como interfaz para buses de datos industriales con excelente desempeño en ambientes ruidosos.

Se agregó al diseño un pequeño altavoz, capaz de emitir tonos de alerta, con el objetivo de informar al usuario sobre el estado del vehículo o fallas, sin necesidad de que éste deba mirar la pantalla instalada en la parte superior.

En la Figura 4.13 se muestra la tarjeta de control en su diseño final, con los componentes instalados. Adicionalmente, se indican en la Tabla 4.1 sus especificaciones.

Ítem	Cantidad y/o comentarios
DSP Texas Instruments TMS320F2808	Unidad principal de procesamiento, controla el funcionamiento general del vehículo e implementa el algoritmo de control.
Salidas a puentes H	2 conectores RJ-45 que incorporan 4 pulsos PWM, alimentación de 5V y 15V además de una línea para recibir la lectura de corriente.
Conector SPI para IMU	1 conector de 8 pines para la tarjeta de instrumentos IMU, incluye alimentación (5V), bus SPI y 3 bits de dirección.
Puerto serie	1 conector con alimentación para el circuito de operación del visor LCD y las conexiones seriales con niveles lógicos de 3,3V. El puerto RS-232 está compartido con el USB
Conector USB	1 conector asociado al puerto serie del DSP a través del convertor USB a RS-232 modelo FT232R.
Altavoz de alerta	Dispositivo integrado a la tarjeta, capaz de emitir sonidos de alerta al usuario.
Conexión JTAG	Puerto de programación y depuración para el DSP mediante un dispositivo JTAG.
Encoders	2 conectores de recepción de pulsos desde los encoders instalados en los motores, cuyo objetivo es determinar la velocidad de las ruedas.
Entradas analógicas	8 canales analógicos de entrada, para tensiones entre 0V y 3,3V proveniente de circuitos activos. (posee baja impedancia $\sim 1k\Omega$)
Conexión SPI-B	Conexión SPI auxiliar.
GPIO	2 entradas y salidas de propósito general, se pueden configurar para encender LEDs integrados en la tarjeta o para conectarlos al exterior.

Tabla 4.1: Especificaciones de la Tarjeta de Control.

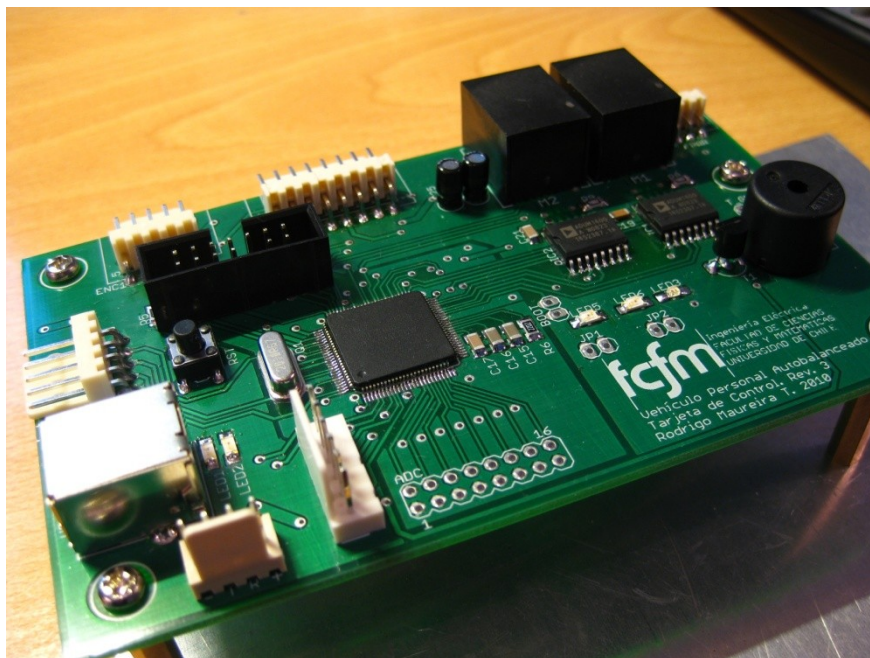


Figura 4.13: Tarjeta de Control

4.2.4 Electrónica auxiliar

Además de las tarjetas electrónicas ya mencionadas, se diseñó una nueva, cuya función principal es obtener mediciones analógicas del sistema.

Esta tarjeta surge de la necesidad de capturar la posición del mando de dirección, que es medida por medio de un potenciómetro instalado en la articulación asociada, al interior de la base del vehículo. Esto es diferente de la implementación anterior que capturaba este dato desde la parte superior del manillar, que fue reemplazado. En su diseño se incorporaron lecturas adicionales como el estado de las baterías y un sensor de presión ubicado en la plataforma para los pies, cuyo objetivo es informar al controlador sobre la presencia del usuario.

Al momento del diseño de esta tarjeta, se consideró la posibilidad de incluir un contactor para conectar y desconectar la alimentación de los puentes H de forma automática, para así resguardar la seguridad del sistema. Para ello, la tarjeta posee un relé capaz de operar un contactor, con un circuito de accionamiento que utiliza un amplificador operacional para generar un retardo en la orden de encendido, de modo de evitar el encendido del contactor por efecto de algún pulso no deseado o ruido electrónico. En el diseño final no fue necesario instalar dicho contactor, pero se construyó el circuito de accionamiento, quedando disponible para su futura utilización.

Todas las mediciones analógicas implementan un filtro pasa bajos pasivo en la entrada, para reducir el ruido. Adicionalmente, cada señal pasa por un circuito convertor de tensión a corriente, para aumentar la robustez de la señal. Un esquema simplificado de este circuito se muestra en la Figura 4.14.

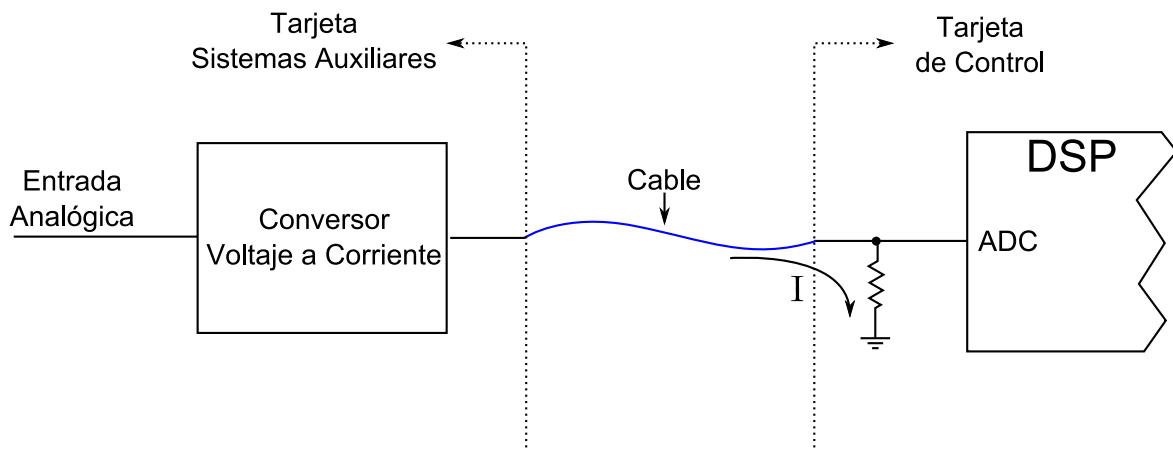


Figura 4.14: Esquema del circuito utilizado para las lecturas analógicas.

De esta manera, se fuerza la circulación de una corriente desde la fuente analógica hasta el pin del DSP con la entrada correspondiente a través del cable de conexión, minimizando el impacto de las pequeñas corrientes inducidas por un eventual ruido.

4.3 SOFTWARE DSP

En esta sección se presenta el desarrollo realizado para el firmware del vehículo, encargado del control general del mismo. La necesidad de replantear el software que ejecuta el DSP es consecuencia de los cambios en el hardware descritos anteriormente y nuevos requisitos que se plantearon como objetivos de este desarrollo.

A diferencia de lo sucedido con los demás aspectos intervenidos del vehículo, el software fue diseñado e implementado completamente desde cero. Esto se realizó a través de un planteamiento sistemático y riguroso del problema, un proceso de desarrollo en el cual las necesidades del sistema fueron traducidas en requisitos de software, éstos transformados en un diseño y el diseño implementado en código.

Con este planteamiento se buscó generar un código funcional y lógicamente comprensible, facilitando su manipulación durante el proceso iterativo de desarrollo y permitiendo la utilización del vehículo como plataforma para aplicar diversas estrategias de control en trabajos futuros.

4.3.1 Requisitos del software

El objetivo principal del software implementado, es controlar un sistema dinámico naturalmente inestable, tomando como entradas las mediciones de los instrumentos y actuando sobre los motores a través de los convertidores de potencia construidos para tal fin.

Además del objetivo principal, debe cumplir una serie de tareas secundarias tales como:

- Realizar control de dirección del vehículo a través de la lectura de los sistemas auxiliares que proporcionan la referencia de dirección.
- Mantener al usuario informado del estado del vehículo, a través de un sistema de interfaz humana, provisto de un visor LCD. Emitir además, alertas sonoras en caso de falla, o falla inminente.
- Recibir diversos datos relativos al estado del vehículo como la carga de las baterías, presencia de usuario, etc.

Debido a la posible necesidad de reemplazar el hardware de control (DSP), durante el proceso de desarrollo, el software se diseñó para cumplir además con un nivel de modularidad tal que sus bloques puedan ser adaptados fácilmente a otro hardware, para lo cual fue necesaria la implementación de una capa de abstracción de hardware.

Teniendo en cuenta lo anterior, se definen los requisitos generales del software:

- Separar las funciones de control de las relacionadas con los periféricos, implementando una capa de abstracción de hardware para ello.
- Capacidad de controlar el vehículo, implementando una estrategia adecuada.

- Lectura en tiempo real la corriente consumida por cada motor para cerrar el lazo de control de corriente.
- Lectura de los instrumentos de la unidad IMU en tiempo real, mediante una conexión SPI.
- Debe implementar un sistema de comunicación por medio de comandos, que permita mantener actualizada la información del visor LCD, recibir órdenes del usuario y la conexión a un sistema de supervisión en un computador.
- Control de los sistemas auxiliares, mediante la lectura de los conversores ADC⁷ y salidas de propósito general.
- Generación de alertas sonoras al usuario mediante un altavoz incorporado en la tarjeta de control.
- Control de los convertidores de potencia mediante el uso de PWM.

Para satisfacer estos requisitos, se implementó un diseño modular que consta de dos niveles: los módulos de alto nivel y módulos de abstracción de hardware.

Los primeros ejecutan tareas lógicas para la administración y el control de los sistemas del vehículo, mientras que los módulos de abstracción de hardware se comunican directamente con los dispositivos periféricos para llevar a cabo las tareas ordenadas por los módulos de alto nivel.

La comunicación entre los módulos se realiza mediante funciones dispuestas para tal fin, estableciendo una interfaz de uso para cada uno en términos de lectura/escritura de valores y ejecución de órdenes, asegurando además la consistencia de los datos en tiempo de ejecución al evitar la intervención directa sobre las variables internas.

4.3.2 Administración de Tareas

El software implementado se ejecuta directamente en la CPU del DSP, sin la intervención de un sistema operativo, por lo que se desarrolló una estrategia que permitiera la ejecución de todos los módulos en forma concurrente, administrando el tiempo que la CPU dedica a cada uno.

Dicha estrategia se basa en el uso de un ciclo principal que itera de forma indefinida cuando el sistema está en régimen permanente, realizando secuencialmente la ejecución de cada módulo para luego actualizar una máquina de estados general en cada iteración.

⁷ Analog-to-Digital Converter

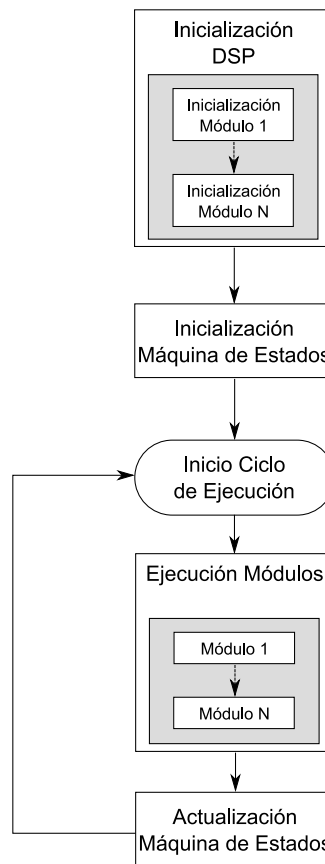


Figura 4.15: Diagrama de flujo general del programa.

En la Figura 4.15 se muestra el diagrama de flujo general del software implementado. Al iniciarse el sistema, comienza la inicialización de cada uno de los módulos a través del llamado a una función de inicialización presente en ellos. Allí se establecen los valores iniciales de las variables y las configuraciones de los periféricos que se utilizarán. Una vez inicializados los módulos, se inicializan las variables del módulo principal, que ejecuta una máquina de estados encargada de administrar los sistemas en su conjunto. Posteriormente se ingresa al ciclo principal que actualiza los módulos llamando la función de ejecución implementada en ellos, la cual realiza sus propios procesos.

Este enfoque posee la ventaja de ser sencillo de comprender e implementar, además de evitar los problemas de sincronización propios de la concurrencia en sistemas operativos ya que, si bien se ejecutan varias tareas de forma concurrente, cada una se realiza mediante actualizaciones secuenciales en las que cada módulo hace uso exclusivo de la CPU sin interrupciones.

Sin embargo, para el éxito de esta estrategia es fundamental cuidar la fluidez de las funciones de ejecución de cada módulo, ya que cualquier ciclo que espere algún evento o pueda caer en ejecución indefinida, detendrá el sistema completo sin posibilidad de recuperación.

Es por este motivo que se prestó especial atención en anular cualquier posibilidad de bloqueo del sistema, debiendo elaborar algoritmos más sofisticados en tareas que involucran la espera de alguna señal, desde algún periférico, como los puertos de comunicación.

El software implementado cuenta con 11 módulos en total, los cuales son:

- HAL Timer
- HAL Main
- HAL Instrumentos
- HAL PWM
- HAL Corriente
- HAL ADC
- HAL GPIO
- HAL Serial
- Módulo de Comandos
- Módulo de Control
- Módulo Principal

Sus funciones específicas y el detalle de su implementación se describen en las siguientes secciones.

4.3.3 HAL Timer

Este módulo tiene por objetivo administrar el temporizador del sistema. Permite controlar además el momento de ejecución de los demás módulos.

Para ello maneja la única interrupción de todo el sistema, cuya función es incrementar un contador de 32 bits. Durante su proceso de inicialización configura uno de los temporizadores del sistema para generar una interrupción cada 100 μ S.

En este módulo, se mantiene un registro interno con el valor del temporizador al momento de la última ejecución de cada uno de los demás módulos. De esta manera, cuando llega el momento de ejecutar nuevamente un módulo, se activa una señal que es leída externamente para concretar su ejecución. Posteriormente, se notifica esta acción para registrar el nuevo valor del temporizador y repetir el proceso. No todos los módulos requieren la misma resolución temporal. Por ejemplo, el módulo que realiza el control de corriente se ejecuta cada 500 μ S, mientras que el módulo de comando, menos crítico para el sistema, se ejecuta cada 10mS.

La interfaz de este módulo permite leer el valor del temporizador y acceder a la función que realiza diferencias entre dos marcas de tiempo, la cual maneja adecuadamente el caso de un desbordamiento en el contador de 32 bits. Con esto es posible medir intervalos de tiempo fuera de este módulo de manera sencilla.

4.3.4 HAL Main

La función de este módulo es agrupar dentro de sí los módulos de abstracción de hardware, organizando su ejecución y simplificando el código del flujo principal del programa.

En su proceso de inicialización se configuran los aspectos esenciales del hardware del DSP y se llama a las funciones de inicialización de todos los módulos de abstracción de hardware.

En régimen permanente, se ejecuta una función que ordena la actualización de los módulos de abstracción de hardware, verificando (en los casos que corresponda) si ha llegado el momento de su ejecución consultando al módulo *HAL Timer* y notificándolo de la manera descrita anteriormente.

4.3.5 HAL Instrumentos

El objetivo de este módulo es leer la información desde los instrumentos de la IMU a través de una de las interfaces SPI del DSP, aplicar los procesos de filtrado adecuados y poner a disposición del módulo de control, el ángulo de inclinación y su derivada en unidades adecuadas.

Para cumplir con estos requerimientos, es necesario ejecutar una rutina de manejo de la interfaz SPI en conjunto con los algoritmos de filtrado y la máquina de estados que lee secuencialmente cada instrumento, ya que se requiere acceder a cuatro dispositivos diferentes a través de un mismo bus, mediante un sistema de direccionamiento incorporado en el diseño electrónico.

Esto se logró aplicando el mismo enfoque de administración de tareas del sistema completo, de manera anidada en este módulo. En consecuencia, se creó un sub módulo encargado de manejar la interfaz SPI; otro sub módulo encargado de recibir las lecturas de los instrumentos y aplicar los filtrados correspondientes; y finalmente un sub-módulo encargado de ejecutar la máquina de estados, solicitando la lectura de un instrumento diferente de manera secuencial, cada cierto intervalo de tiempo determinado por la configuración del módulo *HAL Timer*.

En el cálculo de los valores que son puestos a disposición del módulo de control se aplican dos clases de filtrado:

- Promedio móvil: Para minimizar el efecto que puedan tener eventuales datos incorrectos, producto del ruido electromagnético en el bus de comunicación u otro motivo. Se realiza un promedio de las últimas lecturas obtenidas en una ventana de tiempo determinada. Este tipo de filtrado suaviza la señal de salida pero genera retardos en la respuesta del sistema, por lo que se escogieron tamaños de ventana relativamente pequeños.
- Limitación de derivada: Puesto que las lecturas provienen de un sistema dinámico que posee una cierta inercia, se toma como hipótesis que las variables físicas no pueden variar arbitrariamente entre una muestra y otra. De esta manera, si se aplica un limitador adecuado a la diferencia entre la lectura que se recibe y la anterior, se pueden filtrar valores inválidos sin sacrificar mayormente el tiempo de respuesta del controlador.

Además de los procesos de filtrado, el módulo de lectura de instrumentos implementa una alarma que detiene el vehículo si la lectura del giróscopo es excesivamente alta. Esto sucede habitualmente con movimientos violentos propios de un controlador inestable o producto de un choque del vehículo.

4.3.6 HAL PWM

Este módulo tiene por objetivo manejar las salidas PWM que controlan los convertidores de potencia del vehículo.

Debido a que la generación de los pulsos se realiza completamente por hardware, la implementación de este módulo es sencilla. Únicamente contiene las funciones de inicialización, tres funciones de comando y una de consulta de estado. No requiere de una tarea que se ejecute continuamente, pues basta con actualizar la configuración del periférico del DSP, cada vez que sea necesario.

La función de accionamiento de los motores recibe como parámetros el motor que se desea accionar (izquierda o derecha), la dirección de movimiento (adelante o atrás) y el ciclo de trabajo multiplicado por 10 para mayor resolución, de modo que se pueda ajustar en pasos de 0,1%. En su ejecución, esta función verifica que el ciclo de trabajo esté dentro de los rangos aceptables y aplica un escalamiento para reducir la zona muerta en torno al cero, pues los motores no responden a ciclos de trabajo muy pequeños.

Posee además dos funciones de comando que encienden o detienen completamente los pulsos. Éstas son usadas para iniciar el funcionamiento del vehículo y para detenerlo en caso de operación normal. En caso de falla, la función de detención actúa de forma inmediata.

Una última función entrega el estado del módulo para verificar si se encuentra operando o detenido.

4.3.7 HAL Corriente

La función de hardware de este módulo es recoger las lecturas de corriente desde los puentes H, sin embargo, también implementa un lazo de control de corriente que pone a disposición del módulo de control para accionar el vehículo por medio de referencias de torque en los motores.

Desde el punto de vista de hardware, este módulo configura y utiliza los periféricos de captura del DSP, los cuales registran internamente los instantes en que la señal de entrada experimenta flancos de bajada o de subida. De esta manera, se puede obtener directamente el período de la señal de entrada (un tren de pulsos generado en la tarjeta de los puentes H, cuya frecuencia es proporcional a la corriente en cada motor), con un alto nivel de precisión, facilitando el cálculo de la corriente en los motores.

Gracias a la linealidad del VCO que genera el tren de pulsos (ver Sección 4.2.1), únicamente fue necesario un proceso de calibración para determinar la pendiente de la recta que relaciona la frecuencia en Hertz con la corriente en Amperes. Para obtener la constante de desplazamiento y así caracterizar completamente esta función, el módulo toma muestras de corriente durante un segundo en la inicialización con los motores apagados, utilizando ese valor como los 0A, consiguiendo así lecturas sin corrimientos.

Al igual que en el módulo HAL Instrumentos, se realizan dos tipos de filtrado de la señal de corriente, uno por el promedio de un número determinado de muestras y otro limitando la velocidad con que la lectura de corriente puede cambiar, suponiendo que la dinámica del sistema es lo suficientemente lenta. Esto último no es necesariamente cierto puesto que no se trata de una variable mecánica sino eléctrica. Sin embargo, se comprobó empíricamente que este filtrado mejora el desempeño del control de corriente al omitir transientes eléctricos que no dicen relación con el comportamiento mecánico del sistema, haciéndolo más suave y estable.

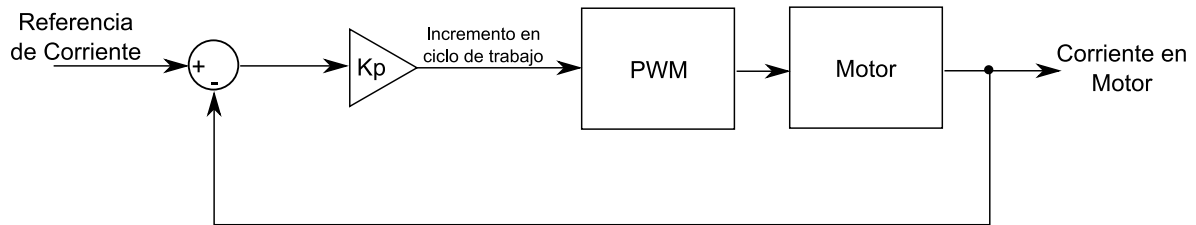


Figura 4.16: Esquema de control de corriente.

Puesto que la corriente en los motores no sólo depende del ciclo de trabajo, se implementó un controlador de corriente proporcional, cuya salida es la variación del ciclo de trabajo respecto al anterior, tal como se observa en la Figura 4.16.

Inicialmente se planteó la posibilidad de utilizar un control proporcional integrativo, sin embargo, esto generó algunos problemas de inestabilidad ya que la referencia entregada a este controlador por el módulo de control de inclinación varía rápidamente, dificultando que el error acumulado se mantenga acotado.

Adicionalmente se observó, durante el proceso de puesta en marcha, que cuando el vehículo desarrollaba una velocidad tal que llevara la corriente a su límite de 25A en los motores (dado por el rango de medición de los sensores de corriente), el controlador era incapaz de compensar, haciendo que el usuario se inclinara hacia adelante, forzando su bajada y la detención del vehículo.

Por esta razón, se aplicó una modificación a la respuesta del controlador de corriente, de manera que la compensación sea más vigorosa antes que se alcance el límite de 25A, forzando la reducción de la inclinación del usuario antes de que se requieran torques mayores a los límites para mantener la estabilidad. En la Figura 4.17 se muestra la relación entre la corriente solicitada por el controlador de inclinación y la referencia efectiva que utiliza el controlador de corriente. Dicha referencia es castigada al pasar de $\pm 20A$, con una pendiente de 2 en la recta. Para los valores de corriente inferiores, no se aplica escalamiento alguno a la referencia (pendiente 1).

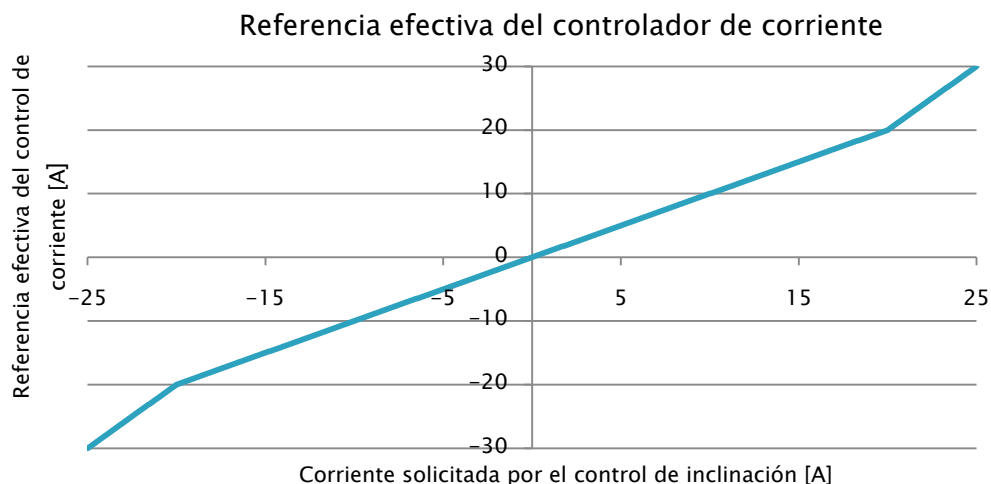


Figura 4.17: Referencia efectiva del controlador de corriente.

La medida resultó ser efectiva, permitiendo la operación del vehículo a velocidades constantes por tramos largos, sin sobrepasar los límites sobre los cuales no es capaz de compensar.

Producto de la incorporación interna del lazo de control de corriente en este módulo, su interfaz resulta de estructura similar a la del módulo HAL PWM, es decir, posee una función cuya entrada es la corriente positiva deseada en Amperes, el motor que se desea accionar y la dirección de giro esperada del motor.

4.3.8 HAL ADC

Este módulo de abstracción de hardware, maneja las entradas analógicas del sistema, obtiene las lecturas del mando de dirección, el estado de las baterías y la presión del sensor ubicado en la plataforma para los pies.

En la función de ejecución de este módulo, se leen los valores desde los conversores análogo digital y se aplican filtrados de promedio. Se accede a estos valores a través de funciones específicas para tal fin, aplicando las conversiones necesarias. La tensión de las baterías es ponderada para generar la medición en décimas de Volt y la lectura del sistema de dirección es limitada para evitar que el vehículo se des controle ante una eventual falla.

4.3.9 HAL GPIO

El módulo HAL GPIO es el encargado de administrar las entradas y salidas de propósito general GPIO⁸ del DSP.

Para este trabajo, únicamente se utilizó la salida de propósito general encargada de excitar el altavoz integrado en la tarjeta de control. Los tonos emitidos son de una frecuencia de 5kHz, determinada por la configuración fijada en el módulo HAL Timer que ejecuta este módulo cada

⁸ General Purpose Input/Output

100 μ S, permitiendo que en cada paso se cambie de estado un pin específico (de alto a bajo y viceversa) completando un ciclo completo en dos pasos.

El módulo cuenta con una función que ordena la emisión de un tono, recibe la duración del tono en décimas de mili segundo y si se debe repetir indefinidamente. Esta última opción genera tonos intermitentes como el usado para la alarma de batería baja.

4.3.10 HAL Serial

En este módulo se sustenta la comunicación del controlador del vehículo con la pantalla LCD y con el software de supervisión en un computador. Su función básica es manejar el periférico de comunicación serial del DSP.

El protocolo de comunicación desarrollado para el vehículo se basa en comandos contenidos en un paquete de datos de 7 bytes, los cuales son interpretados y respondidos por el DSP. En la Tabla 4.2 se muestra una descripción de la estructura de cada comando.

Byte N°	Descripción
1	Código de operación
2	Identificador del paquete
3	Byte de información 1
4	Byte de información 2
5	Byte de información 3
6	Byte de información 4
7	Fin de paquete (255)

Tabla 4.2: Estructura del paquete de datos que conforma cada comando

Cada comando posee un código de operación que determina su función, un identificador único para detectar una eventual pérdida de paquetes y 4 bytes destinados al envío y recepción de datos.

Para la administración del puerto serie del DSP se utiliza un enfoque similar al del módulo HAL Instrumentos, con rutinas que evitan cualquier posibilidad de bloqueo utilizando variables marcadores para indicar la ocurrencia de algún evento en vez de esperar por ellos con el sistema detenido.

La recepción de los comandos desde el exterior se hace leyendo byte a byte la información recibida desde el puerto serie hasta completar un comando completo, el cual es almacenado en una cola que puede guardar hasta 4 comandos a la espera de su procesamiento. La interfaz del módulo provee una función que consulta si existe un comando a la espera de ser procesado y otra que lee el primer comando de la cola. Así mismo, implementa también una función que envía las respuestas de los comandos atendidos, eliminándolos de la cola. Este mecanismo conduce a que sólo un comando pueda ser atendido a la vez, evitando ambigüedades en los demás módulos. Si se reciben comandos con la cola llena, serán ignorados.

4.3.11 Módulo de Comandos

Este módulo de alto nivel tiene por objetivo interpretar y ejecutar los comandos recibidos desde el módulo HAL Serial. Para ello implementa una serie de máquinas de estado que, ayudadas por la estructura de los comandos, distribuyen su decodificación y procesamiento en varios ciclos de ejecución. De esta manera, un único comando tardará varios ciclos de ejecución del programa principal en ser completamente contestado, distribuyendo temporalmente la carga de esta tarea de menor prioridad en la CPU del DSP.

El Módulo de Comandos puede realizar 3 tipos de tareas determinadas por el código de operación del comando recibido: las tareas de lectura de un parámetro desde el vehículo, su escritura o la ejecución de una orden. Se discriminan interpretando los primeros dos bits del código de operación tal como se muestra en la Tabla 4.3.

Código de Operación	Tipo de Acción
00XXXXXX	No especificada
01XXXXXX	Escribir Parámetro
10XXXXXX	Leer Parámetro
11XXXXXX	Ejecutar Orden

Tabla 4.3: Decodificación básica del código de operación.

Además del procesamiento de comandos individuales, el módulo posee un modo de flujo para ciertos comandos de lectura, en el cual un cliente solicita al DSP el envío de uno o más parámetros de forma continua, permitiendo monitorear en tiempo real variables como las lecturas de los instrumentos, corriente en los motores, etc. En este modo, el Módulo de Comandos enviará en cada ejecución un valor por el puerto serie hasta recibir un nuevo comando, el cual automáticamente detendrá el flujo, cualquiera que este sea.

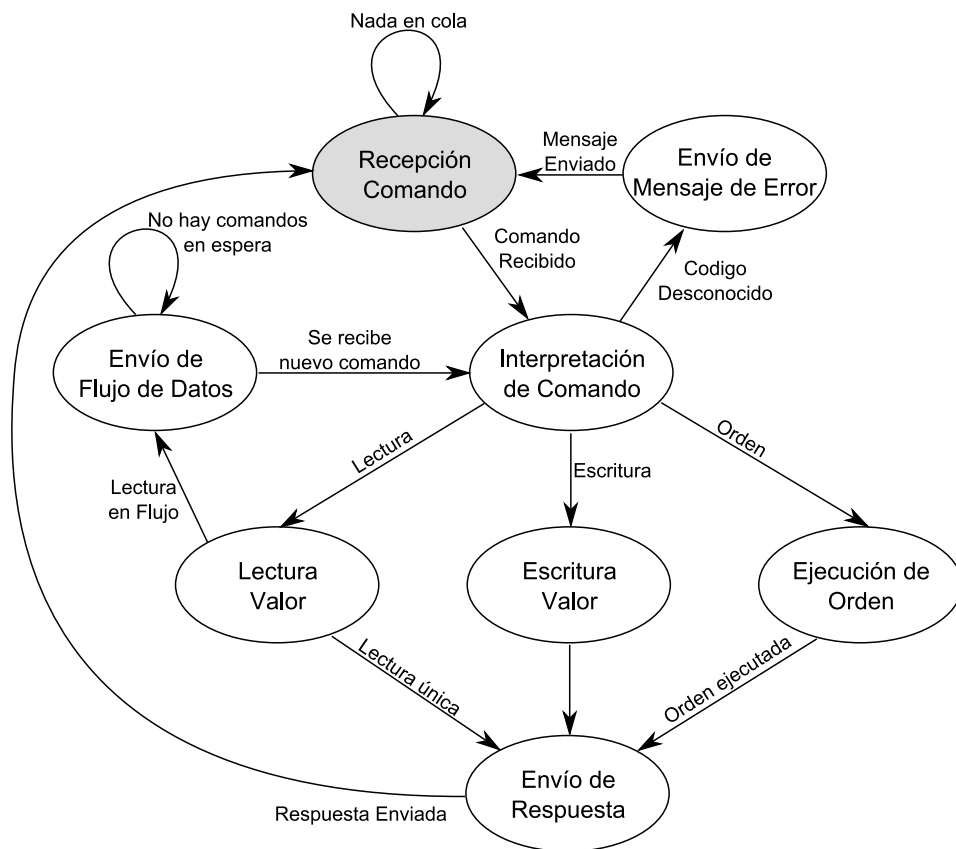


Figura 4.18: Diagrama de estados del Módulo de Comandos.

En la Figura 4.18 se muestra el diagrama de estados del Módulo de Comandos de forma simplificada. El estado inicial es el de recepción que consulta al módulo HAL Serial por la llegada de algún comando. Luego de la recepción se realiza la decodificación y se deriva a la función correspondiente que terminará de decodificar el comando, realizando la acción específica y devolviendo siempre una respuesta por el puerto serie con el mismo código de operación e identificador para indicar al cliente que su solicitud ha sido atendida. En cada proceso de interpretación es posible pasar al estado de envío de mensaje de error si no se reconoce el código de operación.

4.3.12 Módulo de Control

Este es el módulo de mayor relevancia en la operación del vehículo, su objetivo es aplicar una estrategia de control que lo mantenga estable, permitiendo que un usuario se desplace con comodidad. Para ello se sustenta en todos los sistemas previamente descritos, abstrayéndose de sus funciones específicas.

En esencia, se aplica una estrategia de control proporcional derivativa con el ángulo de inclinación del vehículo como entrada y el torque aplicado a las ruedas como salida. Sin embargo, las condiciones reales del sistema hacen necesaria la incorporación de bloques adicionales en este esquema.

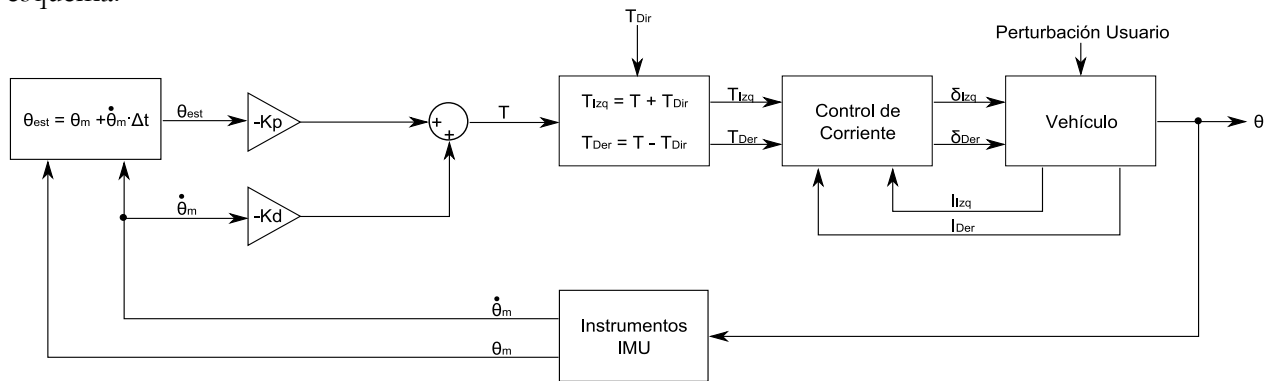


Figura 4.19: Esquema general de control del vehículo

En la Figura 4.19 se muestra el esquema general de control implementado en este módulo, el cual busca llegar a la referencia de 0° en el ángulo de inclinación θ . Las variables involucradas se indican en la Tabla 4.4.

Variable	Descripción
θ	Ángulo de inclinación real del vehículo.
θ_m	Ángulo de inclinación medido por los instrumentos.
$\dot{\theta}_m$	Velocidad de inclinación medida por los instrumentos.
Δt	Intervalo de tiempo de la proyección para la estimación de la inclinación (200mS).
θ_{est}	Estimación del ángulo de inclinación.
K_p	Constante proporcional del controlador PD.
K_d	Constante derivativa del controlador PD.
T	Torque que debe aplicarse a las ruedas para la acción de compensación.
T_{dir}	Diferencia de torques entre ambas ruedas, determinado por el sistema de dirección para maniobrar el vehículo.
T_{izq}	Torque que debe aplicarse a la rueda izquierda.
T_{der}	Torque que debe aplicarse a la rueda derecha.
δ_{izq}	Ciclo de trabajo aplicado al motor izquierdo por el controlador de corriente.
δ_{der}	Ciclo de trabajo aplicado al motor derecho por el controlador de corriente.
I_{izq}	Lectura de corriente del motor izquierdo.
I_{der}	Lectura de corriente del motor derecho.

Tabla 4.4: Variables utilizadas por el algoritmo de control.

La unidad IMU obtiene directamente el ángulo de inclinación (θ_m) y su derivada ($\dot{\theta}_m$) desde los instrumentos en tiempo real. Sin embargo, la lectura de θ_m posee un retardo de aproximadamente 200mS respecto a la lectura del giróscopo ($\dot{\theta}_m$), lo cual hizo necesario aplicar una operación de compensación para estimar el valor real de la inclinación a partir de la lectura obtenida y la proyección dada por su derivada. De esta manera, la estimación utilizada tiene la forma:

$$\theta_{est} = \theta_m + \dot{\theta}_m \cdot \Delta t \quad (4.1)$$

donde θ_{est} es la estimación obtenida y Δt es el intervalo de tiempo de la proyección.

A continuación, el valor del ángulo estimado (θ_{est}) y su derivada ($\dot{\theta}_m$) son ponderados por las constantes del controlador K_p y K_d con signo negativo, para así realizar la acción de compensación y llegar a la referencia de 0° para θ . El valor resultante de esta operación y la suma de los valores parciales, es un torque que debe ser aplicado a ambas ruedas por igual.

Para maniobrar el vehículo, se obtiene un valor proporcional a la posición del mando de dirección que se traduce en una diferencia de torques en los motores, la cual hará girar el vehículo hacia la izquierda o hacia la derecha, permitiendo incluso el giro sobre su propio eje. Dicha diferencia se incorpora al valor de torque obtenido por el algoritmo de control principal de la siguiente manera:

$$T_{Izq} = T + T_{dir} \quad (4.2)$$

$$T_{Der} = T - T_{dir} \quad (4.3)$$

donde T_{Izq} es el torque para el motor izquierdo, T_{Der} es el torque para el motor derecho, T es el torque obtenido por la etapa anterior y T_{dir} es el torque obtenido desde el sistema de dirección. Si T_{dir} es positivo, se aplicará más torque al motor izquierdo que al derecho causando que el vehículo doble hacia la derecha. Si este valor es negativo, se obtiene el efecto contrario.

Los torques resultantes se convierten en las referencias para el módulo HAL Corriente, el cual obtiene las corrientes que circulan por los motores y aplica un lazo de control proporcional como se muestra en la Figura 4.16. El uso de torques como referencias de corriente, para este lazo de control, se justifica por la relación existente entre la corriente por el motor y el torque generado, descritos en la ecuación (2.2), ya que los motores empleados son de imanes permanentes.

Se supone en todo momento la hipótesis de linealidad del sistema a controlar. Sin embargo, se exploró también la posibilidad de usar aproximaciones de segundo orden para las funciones trigonométricas asociadas a los ángulos de inclinación, sin resultados perceptibles por el usuario. Esto se justifica además por el hecho de que los ángulos de inclinación siempre son pequeños (menores a 10°), estando constantemente en torno al punto de operación, es decir, en posición vertical ($\theta = 0^\circ$). Por esta razón se conservó el controlador lineal, quedando comentadas las líneas de código que incorporan la aproximación.

Para el esquema de control descrito, se utilizan dos juegos de parámetros, sintonizados empíricamente para optimizar la estabilidad y desempeño del vehículo, cuando hay un usuario sobre la plataforma y otro set para cuando este no está presente.

La necesidad de discriminar ambos estados surge del hecho de que, constructivamente el vehículo es un sistema dinámico estable cuando no hay un usuario sobre él, puesto que su centro de masa se ubica bajo el eje de pivote de la base. Sin embargo, al subir una persona, el centro de masa cambia y se ubica sobre este eje, creando una situación de péndulo invertido. Es por esta razón que los parámetros que mejor desempeño logran con un pasajero abordo, desestabilizan al vehículo sin usuario.

El cambio de parámetros se realiza al superar un determinado umbral en la lectura del sensor de fuerza, ubicado en la plataforma para los pies.

4.3.13 Módulo Principal

Este módulo corresponde a la función principal que ejecuta el programa del DSP. Administra el estado del vehículo y ordena la ejecución de los demás módulos como se indica en la sección 4.3.2. Puesto que las labores de inicialización y ejecución de los módulos están encapsuladas en cada uno de ellos, el código es sumamente sencillo y tiene la siguiente forma:

```
void main(void)
{
    hal_inicializar_dsp();

    mod_comando_inicializar();
    mod_control_inicializar();

    funcionamiento(); //Se pone en marcha el programa
}
```

La función `hal_inicializar_dsp` inicializa todos los módulos de abstracción de hardware y deja el sistema configurado para su uso. Luego se inicializan los módulos de alto nivel, para finalmente hacer un llamado a una función que contiene la operación en régimen permanente del programa.

Esta última función, al igual que muchas de las funciones críticas del sistema, posee una directiva de compilación justo encima de ella en el código, que las clasifica para luego ser cargadas en la memoria RAM del DSP, lo cual mejora su velocidad de ejecución, puesto que la memoria Flash (donde se aloja el programa), posee mayor latencia, reduciendo notoriamente el rendimiento en labores complicadas como el lazo de control de corriente o la lectura de los instrumentos IMU.

La orden de copia de estas funciones de la memoria Flash del DSP hacia la RAM se hace durante la inicialización del DSP, en la función `hal_inicializar_dsp`.

En régimen permanente, el vehículo puede estar en uno de cuatro estados denominados: Iniciando, Detenido, Operando y Falla. Al momento del encendido, el sistema se encuentra en el estado Iniciando, el cual espera dos segundos, permitiendo así que todas las lecturas entren en régimen permanente, para luego fijar los parámetros predeterminados del controlador, dando paso entonces al estado Detenido.

En el estado Detenido, el sistema espera la recepción de un comando de encendido desde el exterior. Al ocurrir este evento, enciende el módulo HAL PWM permitiendo que los motores reciban energía y pasa al estado Operando, del cual sólo saldrá ante una orden de apagado o la activación de alguna alarma.

En todo momento se monitorea el estado de las baterías, emitiendo un tono de alerta si la tensión baja de cierto umbral, para indicar al usuario que las baterías están prontas a agotarse y pasa a estado de falla si baja de un segundo umbral. Por otro lado, las características del circuito analógico utilizado para enviar la señal al DSP hacen que se envíen 3,3V cuando la tensión medida es cero, permitiendo pasar inmediatamente a estado de falla si se detecta la desconexión del circuito de potencia o falla del fusible

4.4 SOFTWARE DE MONITOREO

La interfaz por comandos desarrollada para el vehículo, permite el acceso a información interna en tiempo de ejecución y es indispensable para la operación del mismo, al recibir los comandos de encendido y apagado desde los controles ubicados en la parte superior del vehículo. Sin embargo, el objetivo de este sistema de comunicación es permitir el acceso a un software de monitoreo que se ejecuta desde un computador, el cual tiene la capacidad de mostrar en tiempo real parámetros del vehículo a un operador, e incluso registrarlos en archivos para su posterior análisis.

Este programa fue realizado bajo el entorno de desarrollo de software Delphi⁹, el cual se especializa en la programación visual, permitiendo la generación de interfaces gráficas de manera sencilla utilizando el lenguaje de programación Object Pascal. El programa compilado es un archivo ejecutable para sistemas operativos Windows. En la Figura 4.20 se muestra la interfaz gráfica del programa de monitoreo al momento de iniciarse.

⁹ <http://www.embarcadero.com/products/delphi>

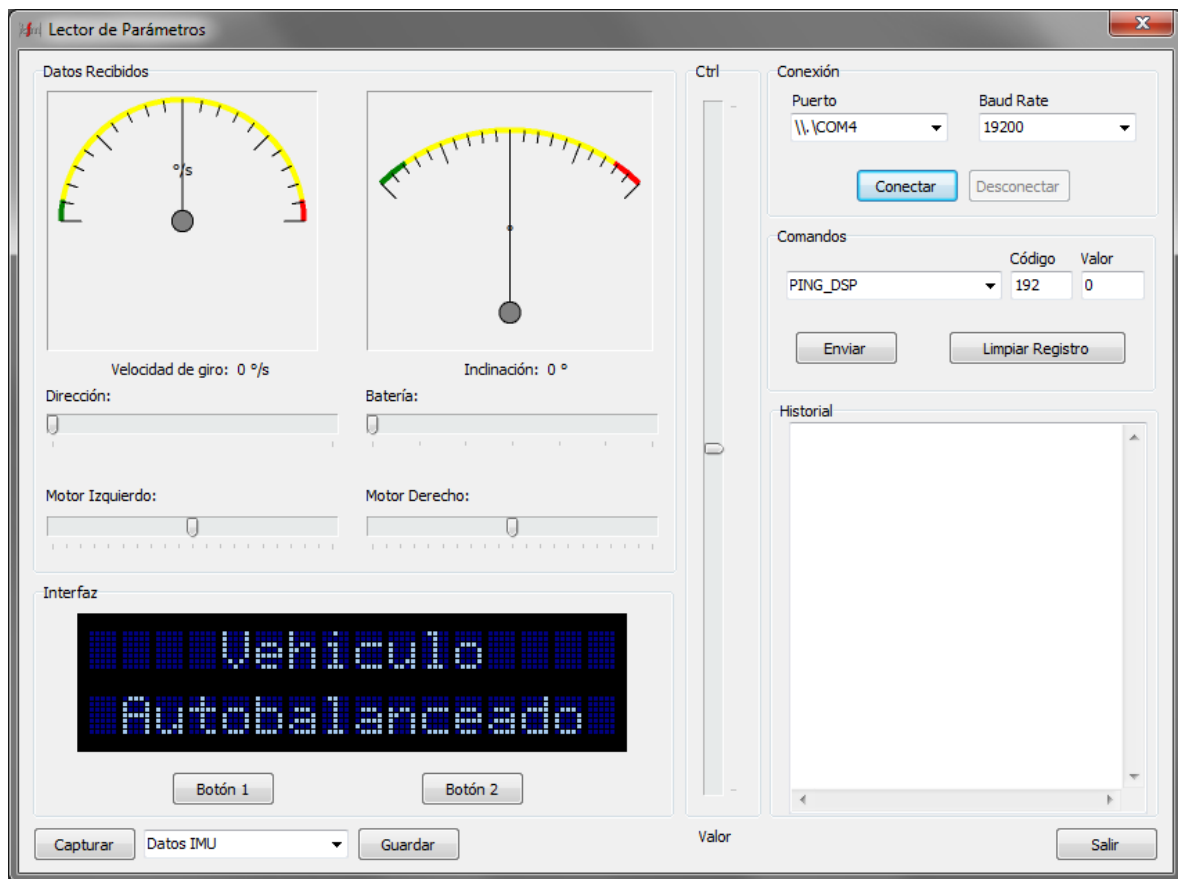


Figura 4.20: Interfaz gráfica del software de monitoreo del vehículo.

El programa se comunica a través de una interfaz RS-232, cuyo hardware es el convertor a USB incorporado en la tarjeta de control, por lo que únicamente se requiere un puerto USB disponible en un computador con plataforma Windows para utilizarlo.

La interfaz gráfica posee un área llamada “**Datos Recibidos**”, que muestra automáticamente el valor de los parámetros allí incluidos al recibir la información desde el vehículo. De esta manera, si se envía al vehículo el comando que solicita el flujo de datos del módulo IMU, automáticamente los indicadores de la parte superior mostrarán la inclinación y la velocidad de inclinación en tiempo real y el valor numérico de las lecturas. Lo mismo sucede con cualquiera de los otros datos que se solicite. El indicador vertical, ubicado al costado derecho de esta área, permite visualizar el valor de la acción de control calculada por el vehículo en tiempo real, si es que se solicita esa información (ver Figura 4.20).

La sección “**Interfaz**” posee una representación de los controles disponibles en la parte superior del mando de dirección en el vehículo, siendo posible emular su funcionamiento.

En la sección “**Comandos**” se pueden enviar comandos arbitrarios. Es posible elegir comandos predefinidos de una lista, pero siempre se permite escribir los códigos manualmente, al igual que el valor enviado.

Este software posee también un historial de los comandos enviados, que también registra algunos de los comandos recibidos.

Los comandos implementados en el vehículo se muestran en la Tabla 4.5:

Nombre de comando	Código decimal	Efecto sobre el vehículo
PING_DSP	192	Este comando no ejecuta ninguna acción excepto generar la devolución del mismo. Su objetivo es verificar conectividad con el controlador.
TRIP_MOTOR	193	Apaga de inmediato los motores. Es el comando de detención del vehículo.
UNLOCK_MOTOR	194	El controlador esperará que el vehículo se encuentre detenido y vertical para encender efectivamente los motores.
FIJAR_INCL_NULA	196	Envía una orden a los inclinómetros, para que éstos fijen su ángulo 0° en la posición actual.
DETENER_FLUJOS	197	Detiene los flujos de datos si los hubiera.
LEER_INCLINOMETROS	129	Envía la lectura de inclinación al software de monitoreo.
LEER_GIROSCOPOS	130	Envía la lectura de velocidad de inclinación al software de monitoreo.
LEER_CORRIENTE_D	131	Envía la lectura de corriente del motor derecho al software de monitoreo.
LEER_CORRIENTE_I	132	Envía la lectura de corriente del motor izquierdo al software de monitoreo.
FLUJO_INCLINOMETROS	133	Envía un flujo de datos con los valores en tiempo real de inclinación al software de monitoreo.
FLUJO_GIROSCOPOS	134	Envía un flujo de datos con los valores en tiempo real de velocidad de inclinación al software de monitoreo.
FLUJO_CORRIENTE_MD	135	Envía un flujo de datos con la corriente en el motor derecho al software de monitoreo.
FLUJO_CORRIENTE_MI	136	Envía un flujo de datos con la corriente en el motor izquierdo al software de monitoreo.
FLUJO_IMU	137	Envía un flujo de datos con los datos tanto de inclinación como de velocidad de inclinación al software de monitoreo.
FLUJO_ADC	138	Envía un flujo de datos con el valor obtenido del sistema de dirección, la lectura de las baterías y la presión en el sensor de fuerza, al software de monitoreo.
FLUJO_CONTROL	141	Envía un flujo de datos con la salida del algoritmo de control PD al software de monitoreo.
SET_MD_F_DUTY	65	Fija el ciclo de trabajo del motor derecho hacia adelante. (Si está operando el controlador, éste sobre escribirá este valor)

SET_MD_B_DUTY	66	Fija el ciclo de trabajo del motor derecho hacia atrás. (Si está operando el controlador, éste sobre escribirá este valor)
SET_MI_F_DUTY	67	Fija el ciclo de trabajo del motor izquierdo hacia adelante. (Si está operando el controlador, éste sobre escribirá este valor)
SET_MI_B_DUTY	68	Fija el ciclo de trabajo del motor izquierdo hacia atrás. (Si está operando el controlador, éste sobre escribirá este valor)
SET_CTRL_KP	69	Fija la constante proporcional del controlador de inclinación para cuando hay un usuario sobre la plataforma.
SET_CTRL_KD	70	Fija la constante derivativa del controlador de inclinación para cuando hay un usuario sobre la plataforma.

Tabla 4.5: Descripción de comandos del vehículo.

Finalmente, los controles de la parte inferior izquierda de la interfaz gráfica (ver Figura 4.20) permiten almacenar flujos de datos solicitados al vehículo. Una vez que se detiene la captura, se puede generar un archivo de texto separado por tabulaciones con los valores recibidos durante el período de captura.

Capítulo 5: Pruebas, Resultados y Análisis del Vehículo Mejorado

El vehículo terminado fue sometido a un intenso período de pruebas, en el cual estuvo expuesto a movimientos violentos, largos tiempos de operación, funcionamiento en superficies irregulares e incluso algunos golpes que pusieron a prueba, tanto el montaje de los componentes electrónicos, como a la estructura mecánica del vehículo.

En este capítulo se abordan en detalle los resultados técnicos y de percepción de los cambios realizados, desde un punto de vista comparativo, respecto al estado inicial del vehículo, verificándose que éste consiguió operar sin necesidad de modificaciones adicionales a las ya expuestas, con un comportamiento confiable.

La nueva organización de los componentes, que se mantuvo fiel al diseño planteado en este trabajo, permite un armado y desarmado completo del vehículo en alrededor de 3 horas. En este tiempo se puede pasar de un vehículo totalmente funcional a la estructura de acero de la base vacía, con los componentes separados y viceversa. Esto resulta sumamente útil para las labores de diagnóstico y mantenimiento.

5.1 ASPECTOS MECÁNICOS

El diseño estructural del vehículo no sufrió mayores modificaciones, por lo que sus características de firmeza y robustez ante condiciones difíciles se mantuvieron consecuentemente. Sin embargo, la modificación realizada al sistema de dirección sufrió una falla durante el período de pruebas, al ceder la soldadura que une el eje sobre el cual pivota el mando de dirección, a la estructura de acero de la base. Dicha situación se solucionó soldando nuevamente la pieza a la estructura, con un tipo diferente de soldadura, aportando más material a la unión en busca de fortalecerla.

A pesar de lo anterior, el nuevo mando de dirección funcionó correctamente luego de algunos ajustes en el sistema de montaje del potenciómetro que mide la posición del mando de dirección. El movimiento lateral permite maniobrar el vehículo con movimientos naturales de inclinación del cuerpo, tal como se esperaba.

La cubierta exterior del vehículo también jugó un rol importante al prevenir el ingreso de suciedad al interior, protegiendo también los componentes internos de los golpes, así como a los usuarios, puesto que su forma redonda, sin puntas filosas, evita cualquier tipo de lesión por impacto. Cabe destacar que en ningún caso el vehículo golpeó con fuerza a un usuario luego de perder el equilibrio, puesto que el proceso de aprendizaje siempre se realizó a velocidades reducidas y bajo supervisión. Además, se tomaron medidas de apagado automático del vehículo ante detección de golpes o ángulos de inclinación extremos, tal como se describió anteriormente, las cuales funcionaron de manera adecuada.

5.2 MEJORAS ELECTRÓNICAS

El énfasis en la reducción de conexiones, sumado a que las nuevas tarjetas electrónicas prácticamente no sufrieron modificaciones luego de su armado (incorporando en ellas todas las funciones requeridas) permiten un funcionamiento predecible. Se puede suponer que los componentes operan sin mayores riesgos mientras se mantengan dentro de las condiciones para las que fueron diseñados. Los componentes electrónicos cumplieron con sus objetivos satisfactoriamente, sin ninguna falla atribuible al diseño, tanto durante el proceso de armado como el de puesta en marcha. Sólo fue necesario hacer correcciones menores, propias del diseño de prototipos, como agregar condensadores para mejorar el filtrado del ruido en las señales de corriente o la alteración de los valores de ciertos componentes para mejorar su desempeño.

El funcionamiento del puente H sufrió dos importantes cambios en su diseño: la nueva electrónica de disparo y la protección contra cortocircuitos.

Desde el punto de vista del encendido de los transistores, efectivamente se consiguió una mejora significativa en los tiempos de conmutación, tal como lo muestra la Figura 5.1. En ella se presenta de forma comparativa las formas de onda del puente H original (izquierda) y del nuevo (derecha) utilizando la misma escala temporal y de amplitud (señal en amarillo).

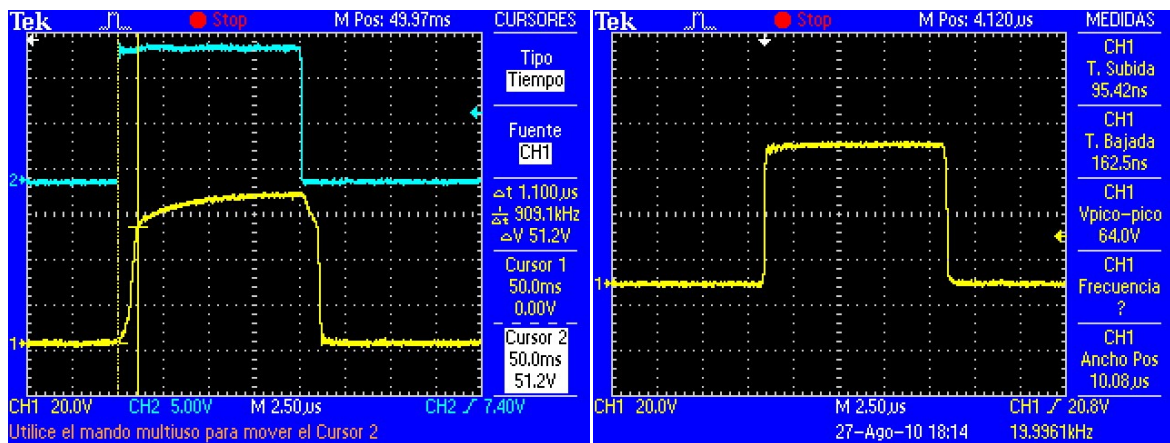


Figura 5.1: (Izquierda) Pulso de encendido en la compuerta de un transistor en el puente H original. (Derecha) Pulso de encendido en la compuerta de un transistor en el nuevo puente H.

La medición de los tiempos de conmutación brinda los resultados cuantitativos de esta mejora, pasando de $1,1\mu\text{S}$ (Figura 3.4) a 100nS (Figura 5.2) en el flanco de subida y de $1,1\mu\text{S}$ (Figura 3.4) a 44nS (Figura 5.2) en el flanco de bajada, con esto los transistores reducen en un factor de 10 el tiempo que se encuentran en zona activa, con la consiguiente reducción de las pérdidas.

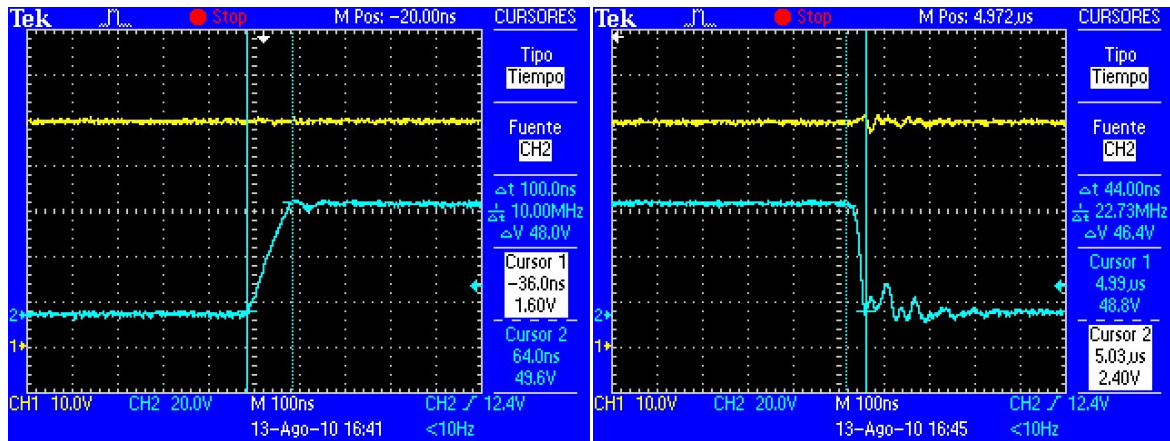


Figura 5.2: (Izquierda) En celeste se muestra el flanco de subida del pulso de encendido en la compuerta del transistor. (Derecha) Se muestra el flanco de bajada del pulso de encendido.

Para el caso de transistores operados como interruptor, la potencia disipada por ellos (P_m) puede ser estimada como:

$$P_m = \underbrace{\frac{V_{OFF} \cdot I_{ON}}{T} \cdot \left(\frac{t_{ON} + t_{OFF}}{6} \right)}_{\text{Comutación}} + \underbrace{\frac{V_{ON} \cdot I_{ON}}{T} \cdot (T_S - t_{OFF} - t_{ON})}_{\text{Conducción}} \quad (5.1)$$

donde el sumando de la izquierda representa las pérdidas totales por conmutación, mientras que el sumando de la derecha representa las pérdidas de conducción en un ciclo de conmutación. V_{OFF} es el voltaje entre drenador y fuente del MOSFET apagado, V_{ON} es el voltaje entre drenador y fuente para el transistor encendido, I_{ON} corresponde a la corriente de conducción, T es el período de la señal de encendido, T_S el tiempo durante el cual el transistor conduce. Finalmente, t_{ON} y t_{OFF} corresponden a la duración de los flancos de encendido y apagado respectivamente [4]. Se han despreciado las pérdidas por corrientes de fuga durante el estado de apagado del transistor, ya que éste puede ser considerado como un circuito abierto.

Puesto que en este caso particular las corrientes y el ciclo de trabajo varían ampliamente, no es posible realizar una estimación precisa de las pérdidas en régimen permanente. Sin embargo, la ecuación (5.1) muestra claramente que la reducción de los tiempos de conmutación a una décima parte de los originales, reduce en igual medida las pérdidas por conmutación en cada transistor.

Tampoco se observaron fenómenos de oscilación producto de la conexión en paralelo de los transistores. Los flancos en el osciloscopio se aprecian relativamente limpios. La temperatura en los disipadores de los transistores tampoco se elevó significativamente por sobre la temperatura ambiente, incluso luego de largos periodos de exigencia.

La protección contra disparo simultáneo también funcionó como estaba previsto. La velocidad de las compuertas lógicas dispuestas en las entradas de los drivers que accionan los transistores, permite prevenir un encendido simultáneo de los transistores de consecuencias destructivas, como lo muestra la Figura 5.3, donde los canales 1 y 2 del osciloscopio corresponden a

la tensión aplicada directamente a las compuertas de los MOSFET (salidas del driver), mientras que la señal inferior (canal 3 del osciloscopio) corresponde al tren de pulsos de entrada.

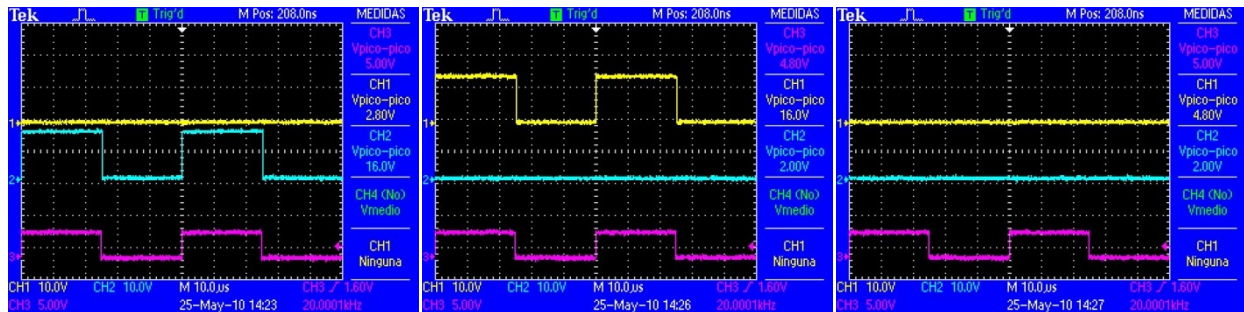


Figura 5.3: (Izquierda) Tren de pulsos aplicado a la una de las entradas del driver. (Centro) Tren de pulsos aplicado a la otra entrada del driver. (Derecha) Tren de pulsos aplicado a ambas entradas simultáneamente.

Luego de numerosas pruebas que confirmaban el buen funcionamiento de esta protección, se conectaron los puentes H a las baterías con un fusible de protección, el cual no fue necesario remover, puesto que el circuito resultó ser lo suficientemente seguro como para permanecer conectado. Luego de muchos ciclos de encendido y apagado no se presentaron fallas, tal como estaba previsto.

Las mediciones de corriente resultaron ser adecuadas y, si bien el circuito VCO utilizado minimiza las posibilidades de alteración en la transmisión de esta lectura, fue necesario emplear cubiertas de cobre a modo de blindaje, puesto que el VCO en sí es sensible al ruido electromagnético generado por los puentes H y los motores.

La robustez de la electrónica de potencia, así como la de control y sus sistemas de montaje, quedó demostrada en el intenso período de pruebas en el que el vehículo sufrió sacudidas e incluso golpes de relativa violencia, que en ningún caso generaron fallas en la electrónica. Los golpes fueron mayormente de las ruedas contra paredes o muro, durante el proceso de aprendizaje, en los momentos en que el usuario baja del vehículo en movimiento ante la inminencia del impacto, sin poder detenerlo oportunamente.

5.3 EVALUACIÓN DE LOS INSTRUMENTOS

Los instrumentos del circuito IMU funcionaron tal como se esperaba. El cambio más relevante es el mejor desempeño del nuevo giróscopo, de la misma tecnología que el inclinómetro. La naturaleza digital de ambos instrumentos los hace inmunes al ruido eléctrico generado por los puentes H, lo cual sumado a que éstos se encuentran dentro de una estructura semi cerrada de aluminio, reduce a un mínimo las probabilidades de contaminación de las señales.

Ambos instrumentos proporcionaron mediciones acordes con lo esperado y con la calidad necesaria para la correcta operación del algoritmo de control. Sin embargo, se detectó un retardo importante en la medición del inclinómetro respecto a la del giróscopo, retardo que probablemente jugó un rol importante también en las dificultades para controlar el vehículo presentadas en el trabajo anterior [2].

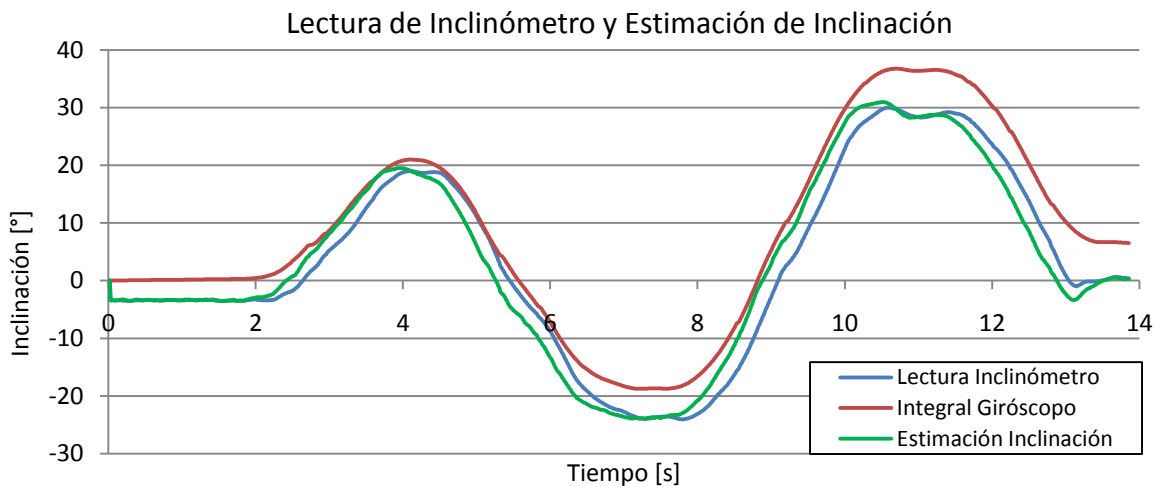


Figura 5.4: Captura de datos de los instrumentos y compensación de retardo

En la Figura 5.4 se muestra una captura de datos obtenida de los instrumentos, en la cual se movió el vehículo hacia adelante y atrás con los motores apagados para registrar así las lecturas sin perturbaciones. La línea en azul corresponde a los datos obtenidos directamente desde el inclinómetro, mientras que la línea roja a la integral numérica de los datos obtenidos desde el giróscopo, poniendo en evidencia la consistencia entre las lecturas de ambos instrumentos.

Se aprecia que la integral de la lectura del giróscopo posee un error acumulativo propio de este tipo de sensores. Es evidente también el retardo antes mencionado entre la lectura entregada por el inclinómetro y la estimación generada a partir del valor del giróscopo.

En verde se muestra la estimación realizada por el controlador, de la inclinación instantánea a partir del valor actual y la proyección de la derivada en 200mS, que es el valor del retardo, reduciendo de manera efectiva este desfase entre las lecturas usadas por el algoritmo de control.

5.4 ESTABILIDAD Y CONTROL

La estrategia de control implementada permite un funcionamiento cómodo para un usuario experimentado, sin embargo, aún presenta algunos problemas que dificultan el proceso de aprendizaje de usuarios novatos.

La utilización del lazo de control de corriente, en cascada con el controlador de inclinación, demostró ser efectivo en reducir las sacudidas incontrolables que se presentaban, en la versión anterior del vehículo, bajo determinadas circunstancias, ya que asegura que la corriente se mantiene dentro de ciertos límites. Esto también suaviza la respuesta ante los movimientos del usuario, evitando compensaciones demasiado vigorosas.

A pesar de lo anterior, un usuario con experiencia puede efectivamente mantenerse fijo en un lugar sin perder el equilibrio por tiempo indefinido, a diferencia de lo sucedido con la versión anterior que requería de constante movimiento para no perder la estabilidad. Es posible también hacer giros con radio cero con el vehículo detenido, así como tomar curvas cerradas a baja velocidad.

Especialmente crítico es el comportamiento del vehículo en planos inclinados, puesto que, al carecer de un lazo de control de velocidad, la compensación se vuelve asimétrica en estos casos, aumentando el riesgo de caídas. No obstante de lo anterior, es posible subir y bajar pequeñas pendientes teniendo el cuidado de abordarlas a muy baja velocidad y con una postura totalmente erguida, lo cual también requiere cierto entrenamiento.

La operación sobre superficies irregulares aumenta significativamente el grado de dificultad para el usuario, puesto que las ruedas del vehículo no están diseñadas para esta aplicación. Los 52 kg. de peso del vehículo, al ser mucho mayor que el peso de una bicicleta (para la cual están diseñadas las ruedas), hace que éstas sean incapaces de amortiguar eficazmente todas las irregularidades, al poseer un pequeño volumen de aire para tal fin.

5.5 EXPERIENCIA DEL USUARIO

Como parte de la evaluación del vehículo luego de las mejoras realizadas, se invitó nuevamente a usuarios novatos a probar el dispositivo. Se aplicaron un total de 13 encuestas de usuarios que no habían utilizado el vehículo antes, evaluándose los mismos aspectos de la Tabla 3.1, con el propósito de comparar la opinión sobre el vehículo mejorado. Cada aspecto es evaluado en una escala de 1 a 10, donde 10 corresponde a un funcionamiento óptimo, mientras que 1 a un funcionamiento insuficiente

Los resultados de aplicar la encuesta mostrada en el Anexo B, promediando las evaluaciones de cada aspecto, se reúnen en la Tabla 5.1:

Aspecto Evaluado	Calificación del Vehículo Mejorado	Cambio en Evaluación*
Facilidad de Aprendizaje	7,2	+2,5
Manejo Intuitivo	7,5	+1,2
Sensación de Seguridad	6,2	+3,2
Potencia	8,4	+3,6
Velocidad	8,4	+3,7
Calificación General	8,0	+2,7

Tabla 5.1: Tabla de la evaluación realizada por usuarios novatos, luego de aplicadas las mejoras.
* Calculado con respecto a los resultados de la Tabla 3.1.

En los resultados presentados en la Tabla 5.1, destaca el hecho de que los usuarios otorgaron mayores puntuaciones en todos los aspectos evaluados de manera espontánea, sin haber conocido el vehículo en su estado inicial, ni haber intentado operarlo antes, lo cual indica que el desempeño general es efectivamente mejor luego de la aplicación de las modificaciones.

Los aspectos que mayores alzas tuvieron fueron la sensación de seguridad, la potencia y velocidad, lo cual da cuenta de un sistema de control que es capaz de responder con mayor vigorosidad ante las perturbaciones generadas por el usuario, compensándolas de manera más efectiva. Producto de lo anterior se obtiene una mayor sensación de seguridad, al reducir la sensación de caída inminente.

Por otro lado, el manejo intuitivo, si bien presentó mejora, no fue sustancial, lo cual puede indicar que los cambios realizados en el sistema de dirección, si bien hacen más natural la operación del vehículo, no constituyen un punto crítico en la experiencia del usuario.

El tiempo de aprendizaje osciló entre 3 y 16 minutos en los casos medidos (con un promedio de 9,8 minutos), representando una mejora respecto a la situación inicial, estimada en 30 minutos. Sin embargo, aún depende significativamente de la habilidad del usuario, presentándose casos en que la

estabilidad se consiguió casi de forma inmediata, mientras otros usuarios lograban mantener el equilibrio con dificultad aún luego de 10 minutos de entrenamiento.

Adicionalmente, se invitó a los usuarios que realizaron las pruebas del vehículo en su estado inicial, para que evaluaran el vehículo mejorado llenando la misma encuesta anterior. Fue posible realizar 8 pruebas, cuyos resultados se resumen en la Tabla 5.2.

Aspecto Evaluado	Calificación del Vehículo Mejorado	Cambio en Evaluación*
Facilidad de Aprendizaje	8,5	+4,3
Manejo Intuitivo	8,6	+2,1
Sensación de Seguridad	7,5	+4,4
Potencia	8,3	+3,8
Velocidad	7,8	+4,0
Calificación General	8,3	+2,8

Tabla 5.2: Resumen de la evaluación realizada por 8 de los usuarios que probaron el vehículo en su estado inicial, luego de aplicadas las mejoras.

* Calculado respecto a la evaluación de los mismos usuarios, como se muestra en la tabla comparativa del Anexo B

En términos generales, los resultados son consistentes con los mostrados en la Tabla 5.1, aunque en este caso la evaluación es ligeramente superior, con mejoras sustanciales en cuanto a sensación de seguridad y facilidad de aprendizaje. Esto último se refleja en el tiempo requerido para maniobrar con comodidad el vehículo, ya que en promedio sólo fueron necesarios 5,9 minutos para conseguirlo (ver Anexo B).

Finalmente, a pesar de que los resultados indican que hubo mejoras perceptibles, aún la sensación de inseguridad continúa siendo un problema importante, lo cual dificulta el aprendizaje y la experiencia del usuario en general.

5.6 ESPECIFICACIONES FINALES DEL VEHÍCULO

El vehículo mejorado en su estado final posee las siguientes características:

Característica	Valor y/o Descripción
Alto	135 cm
Ancho	90 cm
Diámetro de las ruedas	20"
Peso	52 kg
Velocidad Máxima	4,7 km/h
Radio de giro mínimo	0m (Puede girar sobre su propio eje)
Autonomía	2 horas de funcionamiento continuo aprox.
Alimentación	4 baterías de 12V, 12Ah.
Duración de Carga	3 horas alimentando las baterías con una fuente de 58V limitada a 3A de corriente máxima.

Capítulo 6: Conclusiones

6.1 CONCLUSIONES

En esta Memoria se han presentado los detalles correspondientes a una segunda versión funcional del prototipo de vehículo autobalanceado, desarrollado durante el año 2009 e informado en [2]. El vehículo fue intervenido en su estructura mecánica, sistemas electrónicos y software asociado, manteniendo en términos gruesos su conformación general, consolidando el trabajo desarrollado anteriormente y generando soluciones nuevas para cumplir de mejor manera los objetivos planteados originalmente para el vehículo.

El vehículo rediseñado es totalmente funcional, permite transportar a un pasajero de manera eficaz entre dos puntos, sin mayores dificultades una vez que éste se ha entrenado lo suficiente. Las encuestas aplicadas a usuarios indican que, en términos generales, la nueva versión representa una mejora perceptible respecto al vehículo en su condición inicial. Sin embargo, aún es necesario un proceso de aprendizaje tal vez más lento de lo esperado, requiriéndose entre 6 y 10 minutos para conseguir un control básico del vehículo, el cual continúa siendo dependiente, en alguna medida, de las habilidades de cada usuario.

La estrategia de control implementada se hizo en base a la teoría desarrollada en el trabajo anterior, con un ajuste de parámetros que se realizó de manera empírica a través de sucesivas pruebas.

Desde el punto de vista electrónico, el rediseño de sus componentes cumplió satisfactoriamente los objetivos planteados, haciendo del vehículo un equipo confiable con comportamiento predecible. Es posible operarlo suponiendo que los componentes funcionan bien y que no es necesario tener precauciones excesivas.

Los convertidores de potencia mejoraron también su seguridad y desempeño, aumentando su eficiencia, evitando el riesgo de recalentamiento de los transistores al ser operado durante un largo tiempo de manera continua. Constructivamente, estos puentes H están preparados para manejar altas potencias con un mínimo de pérdidas y recalentamiento, gracias a su estructura de bus laminado.

La nueva disposición de las tarjetas electrónicas, el énfasis en la reducción de las conexiones necesarias y la ausencia de componentes fuera de las tarjetas, permite un comportamiento sólido ante golpes y condiciones difíciles, facilitando también las labores de mantenimiento.

Mecánicamente, el vehículo mantiene las buenas características de su predecesor, incorporando además un nuevo mecanismo de dirección para maniobrarlo de manera más cómoda y natural. Sin embargo, los grandes esfuerzos mecánicos a los que está sometido este mecanismo hacen de éste un punto potencialmente vulnerable. Por otra parte, la incorporación de una cubierta protectora de hojalata cilíndrica, protege efectivamente tanto a los componentes internos como al usuario de golpes directos, brinda un aspecto más armónico y estéticamente más agradable.

La nueva unidad IMU diseñada, se comportó como se esperaba, proporcionando mediciones de mejor calidad al controlador, básicamente gracias a la incorporación de un nuevo giróscopo de mejor tecnología que el usado anteriormente. La redundancia de componentes y su interfaz totalmente digital, evita cualquier contaminación por ruido de las señales.

El nuevo software implementado resultó ser completamente funcional, con una estructura jerarquizada que facilita su lectura y comprensión, así como su intervención. El sistema modular de implementación permite efectivamente separar las tareas, haciendo posible el desarrollo de un aspecto específico, sin alterar las complejas interacciones con el resto del programa. Esta característica busca facilitar trabajos futuros en cuanto a técnicas de control más complejas que las ya implementadas, suponiendo que el resto del sistema funciona correctamente.

La incorporación de un protocolo de comandos, permitió también acceder al estado del vehículo en tiempo real, comandar su funcionamiento y monitorear variables críticas, facilitando en gran medida el proceso de sintonización de controladores y diagnóstico de problemas.

En conclusión, se consiguió una segunda versión del vehículo con mejores prestaciones y además una plataforma de desarrollo para continuar mejorando las técnicas de control empleadas, preparando el camino para futuras implementaciones capaces de conseguir un funcionamiento óptimo. A la luz de lo anterior, es posible asegurar que los objetivos planteados inicialmente para esta Memoria han sido total y satisfactoriamente cumplidos.

6.2 TRABAJO FUTURO

Las mejoras realizadas en este trabajo dan paso a dos posibles líneas de continuación para el proyecto: El estudio de estrategias de control aplicadas a la plataforma existente y la optimización del diseño mecánico y electrónico.

En el diseño actual, aún queda pendiente la incorporación al sistema de la información entregada por los encoders instalados en los motores, lo cual permitiría explorar el efecto de la implementación de un control de velocidad sobre las ruedas. También queda propuesta la construcción de un cargador de baterías para independizar completamente al vehículo de los equipos de laboratorio. Otra mejora interesante desde el punto de vista electrónico, es la instalación de una interfaz inalámbrica de comunicación para facilitar la captura de datos del sistema en movimiento por un computador fijo, lo cual puede hacerse a través de la interfaz RS-232 incorporada, usando un módulo Bluetooth, ZigBee u otro similar.

Puesto que ya se ha resuelto la construcción mecánica, la electrónica de potencia, de control y el software que administra el sistema para el vehículo existente, futuras investigaciones pueden enfocarse en una modelación más rigurosa del sistema con el fin de generar estrategias de control de mejor desempeño, las cuales no sólo pueden limitarse al control de inclinación, sino que también al de corriente, cuyo comportamiento afecta significativamente el desempeño general. De la misma manera, se pueden revisar y perfeccionar los algoritmos de filtrado de las señales para obtener respuestas más precisas.

Respecto a la segunda línea de investigación propuesta para la continuación del proyecto, el rediseño completo del vehículo, utilizando como base el trabajo ya realizado, puede ser una alternativa atractiva para conseguir un equipo de mejores características, que efectivamente esté preparado para su uso por usuarios comunes en condiciones normales.

El vehículo actual posee un ancho similar al de una silla de ruedas, lo cual le impide pasar por algunas puertas e incluso por las rampas para discapacitados más angostas. Las cuatro baterías de plomo significan además un enorme peso que los motores deben mover, contribuyendo también al insatisfactorio desempeño en planos inclinados, los cuales sólo pueden ser subidos a muy bajas velocidades y con pendientes pequeñas.

Sin duda una reducción en el tamaño y el peso total serán una ventaja en un nuevo diseño. El peso de las baterías podría reducirse a una tercera parte, de reemplazarlas por tecnología de ion de litio, cuya densidad de energía es alrededor de tres veces la de las baterías de plomo actuales [20]. Dicha reducción facilitaría el uso de motores de menor potencia ya que la masa a mover se reduce, los cuales a su vez podrían ser accionados por puentes H de menor tamaño que, por ejemplo, no hagan uso de dos transistores en paralelo, sino que uno solo por interruptor, reduciendo de manera importante el tamaño de estas tarjetas. Todo lo anterior manteniendo y quizás mejorando la autonomía del vehículo actual, además de disminuir el costo de fabricación del vehículo.

Cambios tan significativos como los comentados, también impactarán de alguna manera en el desempeño del controlador, siendo éste uno de los aspectos más relevantes a analizar.

Una reestructuración profunda del diseño mecánico del vehículo, como la que se propone, debe ser sustentada por un estudio acabado del diseño propuesto, buscando la optimización de los recursos y aplicando los conocimientos adquiridos en un prototipo cuyos componentes han sido sobredimensionados, pero que funciona de manera confiable.

Referencias

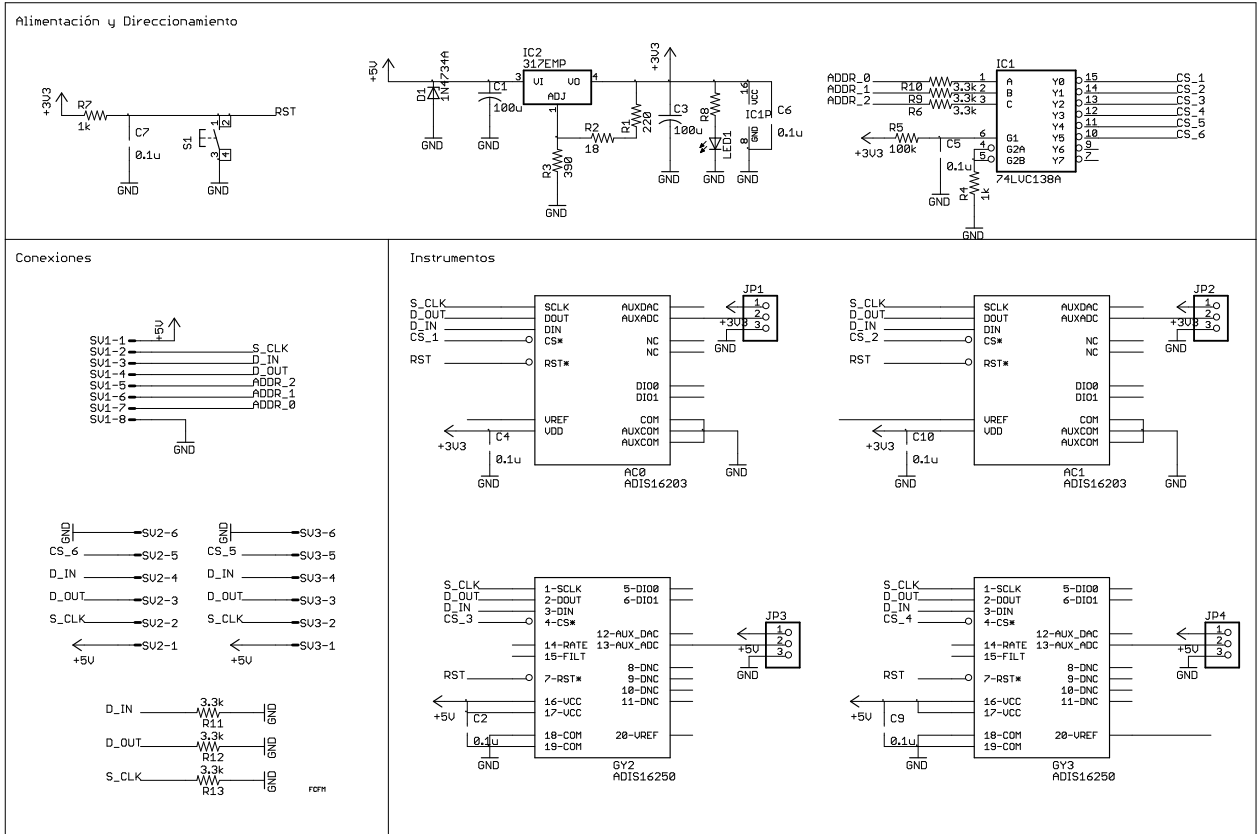
- [1] Segway Inc., "Segway - SPAIN." [En línea] [Citado el: 20 de 11 de 2010.] <http://www.segway.es/faq.asp>.
- [2] Moreno B., Leonardo., *Diseño e Implementación de Vehículo Autobalanceado Sobre Dos Ruedas*. 2009. Memoria para optar al Título de Ingeniero Civil Electricista. Departamento de Ingeniería Eléctrica, Universidad de Chile.
- [3] Vargas, Luis., "Máquinas de Corriente Continua.". Universidad de Chile Departamento de Ingeniería Eléctrica. *Conversión Electromecánica de la Energía*. 2006.
- [4] Rashid, Muhammad H., "Power Electronics Handbook.". Academic Press, 2001.
- [5] Mohan, Ned., *First Course on Power Electronics and Drives*. MNPERE, 2003.
- [6] Brews, John R., "The Metal-Oxide Semiconductor Field-Effect Transistor (MOSFET)." *The Electrical Engineering Handbook*. CRC Press LLC, 2000.
- [7] IXYS Corporation., *IXTH130N10T TrenchMV™ Power MOSFET*. 2008. Hoja de datos. IXYS REF: T_130N10T(V3)07-29-08-A.
- [8] International Rectifier., *Paralleling HEXFET® Power MOSFETs*. Application Note AN-941.
- [9] Allocco, James M., "Laminated Bus Bars for Power System Interconnects." IEEE Colloquium on. Londres, 1998.
- [10] Advanced Power Technology®, *Eliminating Parasitic Oscillation between Parallel MOSFETs*. 2004. Application Note APT-0402 Rev A.
- [11] National Instruments., Inertial Measurement Unit - Developer Zone. [En línea] [Citado el: 10 de 12 de 2010.] <http://zone.ni.com/devzone/cda/tut/p/id/8163>.
- [12] Texas Instruments., C2000™ 32-bit Real-time MCUs - 28x Fixed-point Series - TMS320F2808 - TI.com. [En línea] [Citado el: 20 de 11 de 2010.] <http://focus.ti.com/docs/prod/folders/print/tms320f2808.html>.
- [13] International Rectifier., *Automotive MOSFET IRFP2907ZPbF*. 2004. Hoja de Datos. PD - 95480.

- [14] Rectifier, International., *IR2117(A)/IR2118(S)&(PbF) Single Channel Driver*. 2007. Hoja de datos. No. PD60146 Rev O.
- [15] International Rectifier., *IR2110(S)/IR2113(S)&(PbF) High and Low Side Driver*. 2004. Hoja de datos. No. PD60147 Rev.T.
- [16] Analog Devices., *Quad-Channel Digital Isolators ADuM1400/ADuM1401/ADuM1402*. 2008. Hoja de datos. Rev. G.
- [17] National Semiconductor., *LM231A/LM231/LM331A/LM331 Precision Voltage-to-Frequency Converters*. 2006. Hoja de Datos. DS005680.
- [18] Analog Devices., *Programmable 360° Inclinometer ADIS13203*. 2006. Hoja de datos. Rev. 0.
- [19] Analog Devices., *Programmable Low Power Gyroscope ADIS16250/ADIS16255*. 2008. Hoja de datos. Rev. C.
- [20] Battery University., What's the best battery? [En línea] [Citado el: 10 de 11 de 2010.] http://batteryuniversity.com/learn/article/whats_the_best_battery.
- [21] Barkhordarian, Vrej., *Power MOSFET Basics*. Application Note AN-1084, International Rectifier.

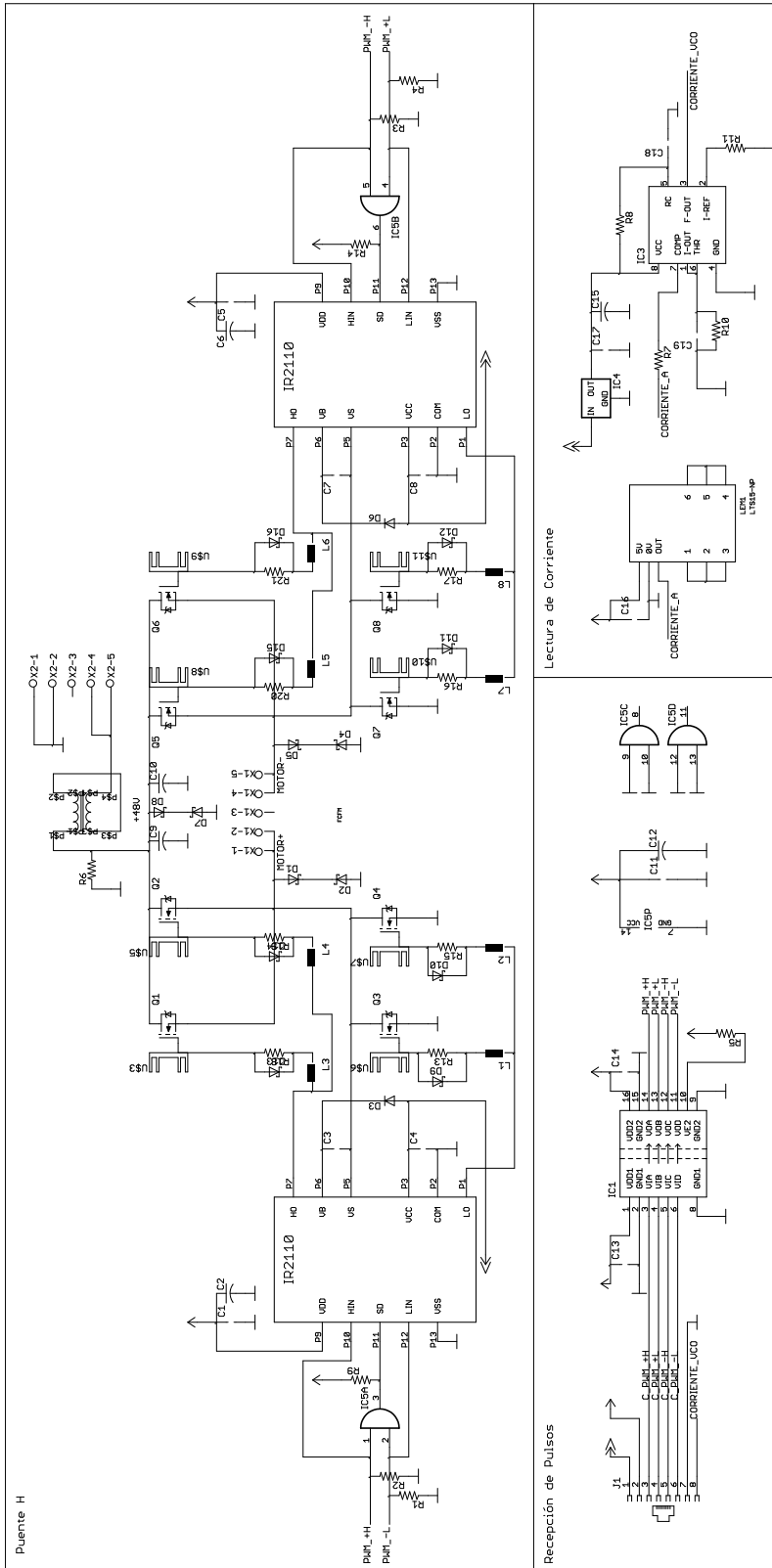
Anexo A: Circuitos Esquemáticos

En el presente anexo, se detallan los circuitos esquemáticos de las tarjetas electrónicas diseñadas para durante el desarrollo de este trabajo. Para confeccionarlos se utilizó el software EAGLE. Los archivos fuente se incluyen en el disco adjunto (Anexo D).

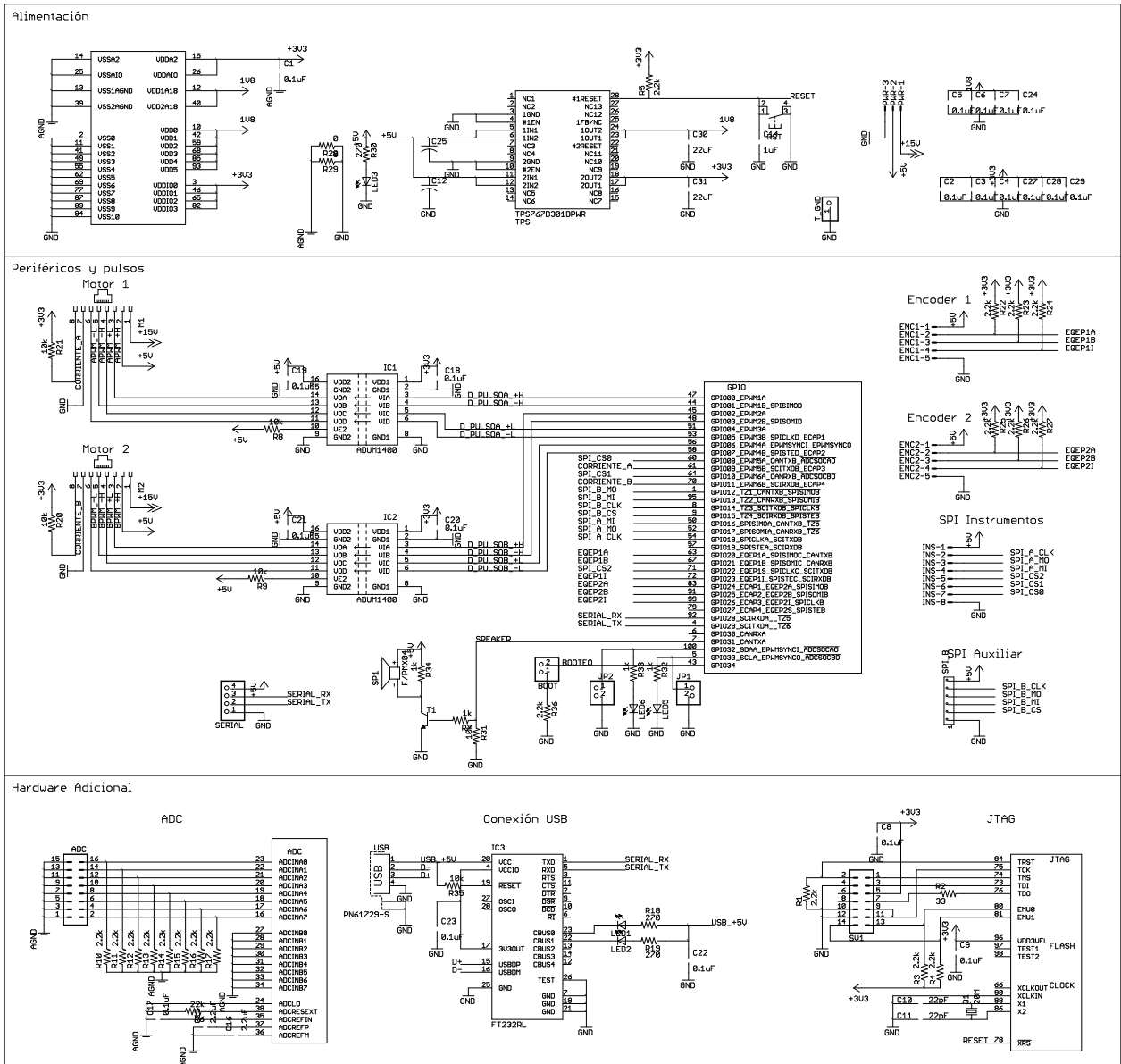
TARJETA UNIDAD IMU



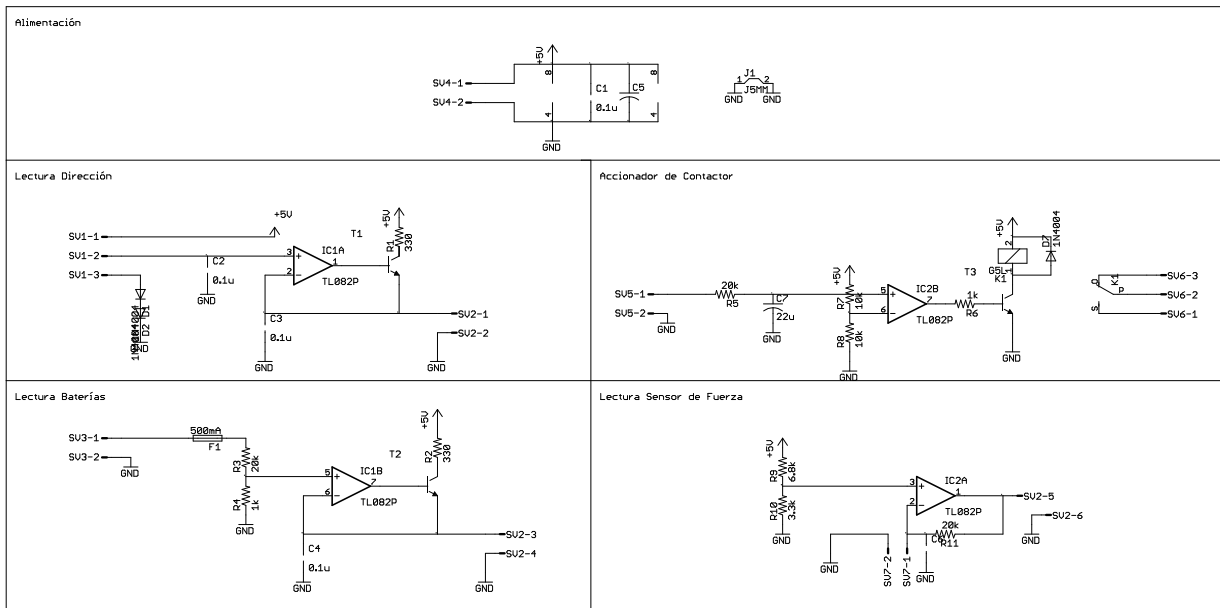
TARJETA PUENTE H



TARJETA DE CONTROL



TARJETA DE LECTURAS ANALÓGICAS



Anexo B: Encuestas

En el presente anexo, se describe la metodología aplicada para la realización de las encuestas a usuarios, así como el detalle de los datos obtenidos.

Tanto para las pruebas iniciales, como para las pruebas con el vehículo mejorado, se utilizó el mismo formato de encuesta. En las tablas con el detalle de los resultados, se omite la información de contacto de los usuarios.

FORMATO DE ENCUESTA

Evaluación de Desempeño	
Nombre: _____	E-Mail: _____
Fecha: _____	
Tiempo de Aprendizaje: _____	
El tiempo de aprendizaje se considera desde el momento en que el usuario se sube al vehículo hasta cuando considere que se siente cómodo al manejarlo sin ayuda.	
Los siguientes puntos se evalúan de 1 a 10. Una calificación de 10 corresponde al funcionamiento óptimo, mientras que 1 a un desempeño insuficiente.	
<u>Ítem</u>	<u>Evaluación</u>
Facilidad de aprendizaje:	_____
Manejo intuitivo:	_____
Sensación de seguridad:	_____
Potencia:	_____
Velocidad:	_____
Calificación general:	_____

EVALUACIÓN INICIAL CON USUARIOS NOVATOS

Al inicio del presente trabajo, se aplicaron encuestas a 14 usuarios antes de realizar cualquier intervención en el vehículo, utilizando el formato presentado anteriormente. Durante la realización de estas pruebas, no fue posible medir el tiempo efectivo de aprendizaje puesto que ninguno de los usuarios logró manejarlo cómodamente sin ayuda durante el tiempo que estuvieron dispuestos a intentarlo. Ante esta situación, se les solicitó dar una estimación de cuánto tiempo tardarían en alcanzar el control esperado por lo que, si bien los datos se incluyen en la tabla, no fueron tomados en cuenta para el análisis de los datos.

N°	Fecha	Nombre	Tiempo de Aprendizaje* (min)	Facilidad de Aprendizaje	Manejo Intuitivo	Sensación de Seguridad	Potencia	Velocidad	Calificación General
1	02/10/2009	Emmanuel Vrignaud	30	4	4	3	3	6	4
2	02/10/2009	Javier Urbina Fuentes	25	6	9	4	6	2	6
3	02/10/2009	Javier Moya	240	3	7	2	2	2	6
4	02/10/2009	Inés Otárola	15	4	7	3	2	2	5
5	02/10/2009	Vader Johnson	120	5	5	3	7	6	6
6	02/10/2009	Rodrigo Gutierrez	60	6	4	3	4	5	6
7	02/10/2009	Rodrigo Alarcón Reyes	40	4	7	4	7	5	6
8	02/10/2009	Cristóbal Tapia Camú	5	6	7	2	5	4	5
9	02/10/2009	Fabián Serradell Díaz	20	3	9	2	3	3	4
15	02/10/2009	Rodrigo Maureira Tenorio	50	3	4	2	5	4	4
10	06/10/2009	Nestor Gallegos	5	6	9	4	3	2	5
11	06/10/2009	Natalia Rojas	10	7	8	5	7	9	7
12	06/10/2009	Carlos Tampier	-	3	5	4	5	7	5
13	06/10/2009	Carolina Otárola	-	5	4	2	7	8	5
14	06/10/2009	Guillermo Campusano	5	5	6	2	6	6	6
Promedios			48,1	4,7	6,3	3,0	4,8	4,7	5,3

* Los datos sólo corresponden a estimaciones y no fueron utilizados en el análisis de los resultados. Los usuarios que no entregaron estimación se marcan con un guión.

EVALUACIÓN DEL VEHÍCULO MODIFICADO CON USUARIOS NOVATOS

Para este caso, se aplicó la misma metodología anterior, con la diferencia de que en la mayoría de los casos sí se obtuvo una medición de tiempo entre que el usuario comienza el entrenamiento hasta que se siente cómodo operándolo.

N°	Fecha	Nombre	Tiempo de Aprendizaje (min)	Facilidad de Aprendizaje	Manejo Intuitivo	Sensación de Seguridad	Potencia	Velocidad	Calificación General
1	30/10/2010	Elena Cifuentes	10*	5	5	4	7	8	6
2	30/10/2010	Gabriel Maureira	10	8	9	9	7	8	9
3	30/10/2010	Mauricio Maureira C.	20*	7	7	6	10	8	8
4	30/10/2010	Patricia Maureira L.	3	8	10	8	10	8	8
5	30/10/2010	Mauricio Maureira L.	20*	5	5	4	6	7	6
6	30/10/2010	Hector Maureira L.	30*	8	9	6	7	8	8
7	06/10/2010	Agustín Pudadanas	16	6	8	6	8	5	7
8	05/10/2010	Enrique Guerrero	20*	7	10	7	10	10	9
9	06/10/2010	Diego Valenzuela Iturra	9	10	5	8	10	10	8
10	06/10/2010	Juan Andrés Muñoz	15	8	8	7	8	10	9
11	06/10/2010	Gonzalo Mondaca	6	8	8	5	7	7	8
12	06/10/2010	Cristian Retamal	7	7	8	6	10	10	9
13	06/10/2010	Américo Jerez	12	7	6	5	9	10	9
Promedios			9,8	7,2	7,5	6,2	8,4	8,4	8,0

* Los usuarios marcados con asterisco no completaron el tiempo de práctica indicado, se les solicitó dar una estimación. Los demás valores corresponden a tiempos medidos. El promedio final se calculó únicamente en base a los tiempos medidos.

EVALUACIÓN COMPARATIVA DEL VEHÍCULO CON EL MISMO GRUPO DE USUARIOS

De las 14 personas encuestadas inicialmente, se logró que 8 de ellas probaran el vehículo mejorado y volvieran a llenar la encuesta con el fin de comparar su percepción antes y después de los cambios.

En esta oportunidad, todos los usuarios lograron manejar cómodamente el vehículo, siendo posible medir los tiempos reales de aprendizaje. Debido a las dificultades para medir este parámetro en las pruebas iniciales, sólo se presentan en la tabla los tiempos para las pruebas con el vehículo mejorado.

N°	Nombre	Tiempo de Aprendizaje (min)	Facilidad de Aprendizaje		Manejo Intuitivo		Sensación de Seguridad		Potencia		Velocidad		Calificación General	
		F	I	F	I	F	I	F	I	F	I	F	I	F
1	Javier Urbina Fuentes	6	6	8	9	9	4	7	6	10	2	8	6	8
2	Javier Moya	8	3	8	7	10	2	7	2	10	2	10	6	9
3	Inés Otárola	3	4	9	7	9	3	8	2	7	2	7	5	8
4	Vader Johnson	6	5	8	5	7	3	8	7	7	6	7	6	8
5	Rodrigo Gutierrez	3	6	8	4	7	3	8	4	6	5	5	6	7,7
6	Fabián Serradell Díaz	3	3	10	9	10	2	8	3	10	3	10	4	9
7	Rodrigo Maureira Tenorio	8	3	8	4	9	2	6	5	8	4	7	4	8
8	Rodrigo Ordóñez	10	4	9	7	8	6	8	7	8	6	8	6,5	8,5
	Promedio	5,9	4,3	8,5	6,5	8,6	3,1	7,5	4,5	8,3	3,8	7,8	5,4	8,3

En la tabla, las columnas marcadas con "I" corresponden a la evaluación del vehículo en su condición inicial, mientras que "F" marca las puntuaciones asignadas al vehículo mejorado.

Anexo C: Códigos Fuente

En el presente anexo, muestra el código fuente del software desarrollado. En primer lugar, se presenta el software de la Tarjeta de Control, en el cual, cada módulo está conformado por un archivo diferente, con un encabezado en el que se describe brevemente su función.

Posteriormente, se presentan los códigos del software implementado en la tarjeta controladora de la pantalla LCD y el Software de Monitoreo. Cada uno de estos programas está contenido en un solo archivo.

SOFTWARE DSP (TARJETA DE CONTROL)

```
1. //#####
2. //Archivo: principal.c
3. //Titulo: Modulo Principal
4. //Descripcion: Cumple la funcion de sistema operativo, inicializa los
5. //                perifericos y comunica entre si los modulos de alto nivel.
6. //                Administra el hilo principal de ejecución.
7. //
8. //
9. //Version:
10. //2010-11-19: v1.5 - Versión Operativa. Rodrigo Maureira.
11. //#####
12.
13. //Librerias del sistema
14. #include "..\Include\header\DSP280x_Device.h" // Headers para dispositivo
15. #include "..\Include\header\DSP280x_DefDisp.h" // Definiciones de Dispositivo
16. #include "hal\hal_main.h" //Capa de Abstraccion de Hardware
17. #include "hal\hal_timer.h"//Módulo HAL de control del tiempo
18. #include "hal\hal_adc.h" //Módulo HAL de entradas analógicas
19. #include "hal\hal_gpio.h" //Módulo HAL de salidas de propósito general
20. #include "hal\hal_instrumentos.h" //Módulo HAL de lectura de instrumentos
21. #include "mod_comando.h" //Modulo de comando
22. #include "mod_control.h"
23.
24. void funcionamiento(void);
25.
26. typedef enum
27. {
28.     ESTADO_FALLA = 0,
29.     ESTADO_INICIANDO = 1,
30.     ESTADO_DETENIDO = 2,
31.     ESTADO_OPERANDO = 3
32. } estado_vehiculo;
33.
34. typedef enum
35. {
36.     A_NINGUNA = 0,
37.     A_BATERIA_BAJA= 1,
38.     A_FUSIBLE = 2,
39.     A_INCLINACION = 3
40. } tipo_alarma;
41.
42.
43. void main(void)
44. {
45.     hal_inicializar_dsp();
46.
47.     mod_comando_inicializar();
48.     mod_control_inicializar();
49.
50.     funcionamiento(); //Se pone en marcha el programa
51. }
52.
53.
54. #pragma CODE_SECTION(funcionamiento, "ramfuncs");
55. void funcionamiento(void)
56. {
57.     estado_vehiculo estado;
58.     Uint32 marca_tiempo;
59.     int16 nivel_bateria;
```

```

60.     tipo_alarma alarma;
61.
62.     estado = ESTADO_INICIANDO;
63.     marca_tiempo = hal_timer_get_time();
64.     alarma = A_NINGUNA;
65.
66.     for(;;)
67.     {
68.     ejecutar_hal();
69.     nivel_bateria = hal_adc_get_bateria();
70.
71.     if (hal_timer_trigger(MOD_COMANDO) == TRUE)
72.     {
73.         mod_comando_ejecutar();
74.         hal_timer_mod_triggered(MOD_COMANDO);
75.     }
76.     if (hal_timer_trigger(MOD_CONTROL) == TRUE)
77.     {
78.         mod_control_ejecutar();
79.         hal_timer_mod_triggered(MOD_CONTROL);
80.     }
81.
82.     switch (estado) {
83.     case ESTADO_INICIANDO: if (hal_timer_diff(marca_tiempo, hal_timer_get_time()) > 20000)
84.     {
85.         //Parámetros para vehiculo con pasajero
86.         mod_control_set_var(KP, 350); //260
87.         mod_control_set_var(KD, 260); //250
88.         //Parámetros para vehiculo sin pasajero
89.         mod_control_set_var(KP_S, 80);
90.         mod_control_set_var(KD_S, 45);
91.         estado = ESTADO_DETENIDO;
92.         hal_gpio_emitir_tono(D_TONO_ALERTA, FALSE);
93.     }
94.     break;
95.
96.     case ESTADO_DETENIDO:   if (get_orden_inicio() == TRUE) //Se solicita encendido
97.     {
98.     menos centrado
99.         {
100.             hal_gpio_emitir_tono(D_TONO_INICIO, FALSE);
101.             hal_pwm_start();
102.             estado = ESTADO_OPERANDO;
103.         }
104.
105.     }
106.     if (nivel_bateria > 600) //Fusible quemado
107.     {
108.         set_orden_inicio(FALSE); //Cancela el encendido
109.         hal_gpio_emitir_tono(D_TONO_INICIO, TRUE);
110.         estado = ESTADO_FALLA;
111.         alarma = A_FUSIBLE;
112.     }
113.     break;
114.
115.     case ESTADO_OPERANDO: if (get_orden_inicio() == FALSE)
116.     {
117.         hal_pwm_trip();
118.         estado = ESTADO_DETENIDO;
119.     }
120.     if (nivel_bateria < 450) //Batería baja
121.     {
122.         hal_gpio_emitir_tono(D_TONO_ALERTA, TRUE);
123.         if (nivel_bateria < 430) //Se acabó la batería
124.         {
125.             hal_pwm_trip();
126.             set_orden_inicio(FALSE); //Cancela el encendido
127.             hal_gpio_emitir_tono(D_TONO_INICIO, FALSE);
128.             estado = ESTADO_FALLA;
129.             alarma = A_BATERIA_BAJA;
130.         }
131.     } else {
132.         if (nivel_bateria > 600) //Fusible quemado
133.         {
134.             hal_pwm_trip();
135.             set_orden_inicio(FALSE); //Cancela el encendido
136.             hal_gpio_emitir_tono(D_TONO_INICIO, TRUE);
137.             estado = ESTADO_FALLA;
138.             alarma = A_FUSIBLE;
139.         }
140.     }
141.     break;
142.

```

```

143.     case ESTADO_FALLA:         hal_pwm_trip();
144.     bateria                    if ((alarma == A_BATERIA_BAJA) && (nivel_bateria > 480)) //Carga mínima de
                                   {
145.                                   alarma = A_NINGUNA;
146.                                   hal_gpio_silenciar_tono();
147.                                   estado = ESTADO_DETENIDO;
148.                                   }
149.                                   if ((alarma == A_FUSIBLE) && (nivel_bateria < 600)) //Se conectó fusible
150.                                   {
151.                                   alarma = A_NINGUNA;
152.                                   hal_gpio_silenciar_tono();
153.                                   estado = ESTADO_DETENIDO;
154.                                   }
155.                                   break;
156.
157.     default: estado = ESTADO_FALLA;
158.             hal_pwm_trip();
159.             break;
160.
161. }
162. }
163. }

```

```

1. //#####
2. //Archivo: hal_main.c
3. //Titulo: Modulo de alto nivel de la capa de abstraccion de hardware
4. //Descripcion: Contiene las funciones necesarias para la inicializacion, uso
5. //              y administracion de preifericos para los modulos de alto nivel.
6. //
7. //
8. //Version:
9. //2010-07-12: v0.1 - Creacion de Archivo. Rodrigo Maureira.
10. //#####
11.
12. //Librerias del sistema
13. #include "..\..\Include\header\DSP280x_Device.h" // Headers para dispositivo
14. #include "..\..\Include\header\DSP280x_DefDisp.h" // Definiciones de Dispositivo
15.
16. //Módulos HAL
17. #include "hal_timer.h"
18. #include "hal_serial.h"
19. #include "hal_instrumentos.h"
20. #include "hal_gpio.h"
21. #include "hal_pwm.h"
22. #include "hal_corriente.h"
23.
24. void MemCopy(UInt16 *SourceAddr, UInt16* SourceEndAddr, UInt16* DestAddr)
25. {
26.     while(SourceAddr < SourceEndAddr)
27.     {
28.         *DestAddr++ = *SourceAddr++;
29.     }
30.     return;
31. }
32.
33. #pragma CODE_SECTION(absoluto, "ramfuncs");
34. int32 absoluto(int32 valor)
35. {
36.     if (valor < 0)
37.         return -1*valor;
38.     else return valor;
39. }
40.
41. void hal_inicializar_dsp(void){
42.     InitSysCtrl();
43.
44.     //Copiamos las funciones a RAM
45.     MemCopy(&RamfuncsLoadStart, &RamfuncsLoadEnd, &RamfuncsRunStart);
46.     Init_hal_adc();
47.     Init_hal_timer();
48.     Init_hal_serial();
49.     Init_hal_instrumentos();
50.     Init_hal_gpio();
51.     Init_hal_pwm();
52.     Init_hal_corriente();
53. }
54.
55. #pragma CODE_SECTION(ejecutar_hal, "ramfuncs");
56. void ejecutar_hal(void)
57. {
58.     //Modulos HAL de ejecucion continua
59.     hal_timer_ejecutar();
60.     hal_serial_ejecutar();
61.     // hal_pwm_ejecutar(); //--> La función está vacía
62.     hal_instrumentos_ejecutar();
63.
64.     //Modulos HAL de ejecucion discreta
65.     if (hal_timer_trigger(MOD_HAL_GPIO) == TRUE)
66.     {
67.         hal_gpio_ejecutar();
68.         hal_timer_mod_triggered(MOD_HAL_GPIO);
69.     }
70.
71.     if (hal_timer_trigger(MOD_HAL_CORRIENTE) == TRUE)
72.     {
73.         hal_corriente_ejecutar();
74.         hal_timer_mod_triggered(MOD_HAL_CORRIENTE);
75.     }
76.
77.     if (hal_timer_trigger(MOD_HAL_ADC) == TRUE)
78.     {
79.         hal_adc_ejecutar();
80.         hal_timer_mod_triggered(MOD_HAL_ADC);
81.     }
82. }
83.

```

```

1. //#####
2. //Archivo: hal_instrumentos.c
3. //Titulo: Módulo HAL Instrumentos
4. //Descripcion: Módulo administrador del puerto SPI-A, obtiene datos desde la
5. //          tarjeta de instrumentos.
6. //
7. //Version:
8. //2010-07-19: v1.0 - Creación del Archivo
9. //#####
10.
11. //Librerías del sistema
12. #include "..\..\Include\header\DSP280x_Device.h" // Headers para dispositivo
13. #include "..\..\Include\header\DSP280x_DefDisp.h" // Definiciones de Dispositivo
14.
15. #include "hal_instrumentos.h"
16. #include "hal_main.h"
17. #include "hal_timer.h"
18. #include "hal_pwm.h"
19. #include "hal_gpio.h"
20. #include "..\mod_comando.h"
21.
22. //Definiciones Internas
23. #define INC_SUPPLY_OUT 0x03FF
24. #define INC_AUX_ADC 0x09FF
25. #define INC_TEMP_OUT 0x0BFF
26. #define INC_INCL_OUT 0x0DFF
27. #define INC_INCL_180_OUT 0x0FFF
28. #define INC_NULL 0x18FF
29.
30. #define STATUS 0x3CFF
31.
32. #define GY_ENDURANCE 0x01FF
33. #define GY_SUPPLY_OUT 0x03FF
34. #define GY_GYRO_OUT 0x05FF
35. #define GY_AUX_ADC 0x0BFF
36. #define GY_TEMP_OUT 0x0DFF
37. #define GY_ANGL_OUT 0x0FFF
38.
39. #define MASK14 0x3FFF
40. #define MASK12 0x0FFF
41.
42. int vect_inclinacion[MUESTRAS_INCL];
43. int32 suma_inclinacion;
44. int vect_vel_giro[MUESTRAS_GYRO];
45. int32 suma_vel_giro;
46. int d_inc; //derivada de la inclinación
47. int d_giro; //derivada de la velocidad de giro
48. char indice_inclinacion;
49. char indice_vel_giro;
50.
51. boolean_t leyendo_instrumento;
52. char inst_a_leer;
53.
54. boolean_t *dato_recibido;
55. boolean_t spia_ocupado;
56. boolean_t enviar_dato;
57. Uint16 dato_e;
58. Uint16 *dato_r;
59. char instrumento_activo;
60.
61. //Prototipos de funciones internas
62. void Seleccionar_Instrumento(char inst); //Selecciona el instrumento a leer
63. void sumar_inclinacion(Uint16 incl); //agrega una lectura al buffer de inclinacion
64. void sumar_vel_giro(Uint16 gyro); //agrega una lectura al buffer de velocidad de giro
65.
66. boolean_t leer_instrumento(char instrumento); //Ordena la lectura de un instrumento
67. void lectura_instrumento(void); //Ejecuta la lectura de instrumentos
68.
69. void enviar_orden_a_instrumento(char id, Uint16 orden);
70. int leer_giroscopto(char id); // lee el giroscopo indicado
71. int leer_inclinometro(char id); //lee el inclinometro indicado
72.
73. void manejar_spi_a(void);
74. boolean_t transmitir_dato(Uint16 dato, char instrumento, Uint16 *lugar_recepcion, boolean_t
    *flag_listo);
75.
76. void ejecutar_maquina_de_estados(void); //Alterna las lecturas de los instrumentos
77.
78. void Init_hal_instrumentos(void)
79. {
80.     int i;
81.
82.     //Inicializar variables internas
83.

```

```

84. //Se inicializan los buffers de captura
85. for (i=0; i<MUESTRAS_INCL; i++)
86. {
87.     vect_inclinacion[i] = 0;
88. }
89. suma_inclinacion = 0;
90.
91. for (i=0; i<MUESTRAS_GYRO; i++)
92. {
93.     vect_vel_giro[i] = 0;
94. }
95. suma_vel_giro = 0;
96.
97. indice_inclinacion = 0;
98. indice_vel_giro = 0;
99. d_inc = 0;
100. d_giro = 0;
101.
102. //Inicializo variables de manejo de SPI-A
103. dato_recibido = &spia_ocupado; //Solo por iniciar los punteros con una direccion valida
104. dato_r = &dato_e; //Solo por iniciar los punteros con una direccion valida
105. spia_ocupado = FALSE;
106. enviar_dato = FALSE;
107. dato_e = 0;
108. instrumento_activo = 7; //Valor que deshabilita todos los instrumentos
109.
110. leyendo_instrumento = FALSE;
111. inst_a_leer = 7;
112.
113.
114. EALLOW;
115. // Habilitar salidas GPIO para dirección de SPI Instrumentos
116. GpioCtrlRegs.GPAPUD.bit.GPIO8 = 0; // Activar Pull up en GPIO8
117. GpioDataRegs.GPASET.bit.GPIO8 = 0; // Poner un cero en ese pin
118. GpioCtrlRegs.GPAMUX1.bit.GPIO8 = 0; // GPIO8 = GPIO8
119. GpioCtrlRegs.GPADIR.bit.GPIO8 = 1; // GPIO8 = output
120.
121. GpioCtrlRegs.GPAPUD.bit.GPIO10 = 0; // Activar Pull up en GPIO10
122. GpioDataRegs.GPASET.bit.GPIO10 = 0; // Poner un cero en ese pin
123. GpioCtrlRegs.GPAMUX1.bit.GPIO10 = 0; // GPIO10 = GPIO10
124. GpioCtrlRegs.GPADIR.bit.GPIO10 = 1; // GPIO10 = output
125.
126. GpioCtrlRegs.GPAPUD.bit.GPIO22 = 0; // Activar Pull up en GPIO22
127. GpioDataRegs.GPASET.bit.GPIO22 = 0; // Poner un cero en ese pin
128. GpioCtrlRegs.GPAMUX2.bit.GPIO22 = 0; // GPIO22 = GPIO22
129. GpioCtrlRegs.GPADIR.bit.GPIO22 = 1; // GPIO22 = output
130.
131. // Configuración de SPI-A (Instrumentos) GPIO16 - GPIO19
132. GpioCtrlRegs.GPAPUD.bit.GPIO16 = 0; // Activar Pull up en GPIO16
133. GpioCtrlRegs.GPAPUD.bit.GPIO17 = 0; // Activar Pull up en GPIO17
134. GpioCtrlRegs.GPAPUD.bit.GPIO18 = 0; // Activar Pull up en GPIO18
135. GpioCtrlRegs.GPAPUD.bit.GPIO19 = 0; // Activar Pull up en GPIO19
136. GpioCtrlRegs.GPAQSEL2.bit.GPIO16 = 3; // entrada asincrona
137. GpioCtrlRegs.GPAQSEL2.bit.GPIO17 = 3; // entrada asincrona
138. GpioCtrlRegs.GPAQSEL2.bit.GPIO18 = 3; // entrada asincrona
139. GpioCtrlRegs.GPAQSEL2.bit.GPIO19 = 3; // entrada asincrona
140. GpioCtrlRegs.GPAMUX2.bit.GPIO16 = 1; // GPIO16 = SPICLKA
141. GpioCtrlRegs.GPAMUX2.bit.GPIO17 = 1; // GPIO17 = SPISOMIA
142. GpioCtrlRegs.GPAMUX2.bit.GPIO18 = 1; // GPIO18 = SPICLKA
143. GpioCtrlRegs.GPAMUX2.bit.GPIO19 = 1; // GPIO19 = SPISTEA
144. EDIS;
145.
146. Seleccionar_Instrumento(7); //Ningun instrumento seleccionado
147.
148. SpiaRegs.SPIFFRX.bit.RXFFIENA = 0; //Deshabilito el FIFO de lectura
149. SpiaRegs.SPIFFTX.bit.TXFFIENA = 0; //Deshabilito el FIFO de escritura
150.
151.
152. /* SPI Configuration Control Register */
153. SpiaRegs.SPICCR.bit.SPISWRESET = 0; // Mientras se configura el puerto se mantiene
    resetado
154. SpiaRegs.SPICCR.bit.CLKPOLARITY = 1;
155. SpiaRegs.SPICCR.bit.SPILBK = 0; // Loopback desactivado
156. SpiaRegs.SPICCR.bit.SPICHR = 0xF; // 16 bits
157.
158. /* SPI Operation Control Register */
159. SpiaRegs.SPICTL.bit.OVERRUNINTENA = 0;
160. SpiaRegs.SPICTL.bit.CLK_PHASE = 0;
161. SpiaRegs.SPICTL.bit.MASTER_SLAVE = 1; // Master mode
162. SpiaRegs.SPICTL.bit.TALK = 1;
163. SpiaRegs.SPICTL.bit.SPIINTENA = 0; // Sin interrupciones
164.
165. /* SPI Baud Rate Register */
166. SpiaRegs.SPIBRR = 0x00F0; //0x000F;

```



```

167.
168.     /* SPI Priority Control Register */
169.     SpiaRegs.SPIPRI.bit.FREE = 1;                               // Breakpoints desde CCS no interrumpen comunicación
170.
171.     SpiaRegs.SPICCR.bit.SPISWRESET = 1;                       // Se reactiva el puerto una vez configurado
172.
173. }
174.
175. #pragma CODE_SECTION(manejar_spi_a, "ramfuncs");
176. void manejar_spi_a(void)
177. {
178.     static Uint32 hora = 0;
179.
180.     //Si hay algo que enviar, se envia
181.     if (enviar_dato == TRUE)
182.     {
183.         Seleccionar_Instrumento(instrumento_activo);
184.         hora = hal_timer_get_time();
185.
186.         SpiaRegs.SPITXBUF = dato_e;
187.
188.         enviar_dato = FALSE;
189.         spia_ocupado = TRUE;
190.     }
191.
192.     if (spia_ocupado)
193.     {
194.         //Si hay algo que recibir, se recibe
195.         if (SpiaRegs.SPISTS.bit.INT_FLAG == 1)
196.         {
197.             *dato_r = SpiaRegs.SPIRXBUF;
198.             *dato_recibido = TRUE;
199.             spia_ocupado = FALSE;
200.             Seleccionar_Instrumento(7); //Deshabilita todos los instrumentos
201.         } else
202.         { //Si no hay nada que recibir, se verifica el Time-Out
203.             if (hal_timer_diff(hora, hal_timer_get_time()) > SPI_LECT_TIME_OUT)
204.             {
205.                 //MANEJO DEL ERROR
206.                 *dato_r = 0xFFFF;           //Marcador del error
207.                 spia_ocupado = FALSE;
208.                 Seleccionar_Instrumento(7); //Deshabilita todos los instrumentos
209.             }
210.         }
211.     }
212. }
213.
214. boolean_t transmitir_dato(Uint16 dato, char instrumento, Uint16 *lugar_recepcion, boolean_t *flag_listo)
215. {
216.     if (spia_ocupado == TRUE)
217.         return FALSE;
218.
219.     //Asignacion de variables
220.     instrumento_activo = instrumento;
221.     dato_e = dato;
222.     dato_r = lugar_recepcion;
223.     dato_recibido = flag_listo;
224.
225.     enviar_dato = TRUE; //Da la orden de enviar
226.     return TRUE;       //Operacion exitosa
227. }
228.
229. #pragma CODE_SECTION(hal_instrumentos_ejecutar, "ramfuncs");
230. void hal_instrumentos_ejecutar(void)
231. {
232.     manejar_spi_a();
233.     lectura_instrumento();
234.
235.     //Si me toca leer algo
236.     if (hal_timer_trigger(MOD_HAL_INST) == TRUE)
237.     {
238.         ejecutar_maquina_de_estados();
239.         hal_timer_mod_triggered(MOD_HAL_INST);
240.     }
241. }
242.
243. void ejecutar_maquina_de_estados(void)
244. {
245.     static char estado = 0; //Estado de la rotacion de instrumentos
246.
247.     switch (estado) {
248.         case 0: if (INCLINOMETRO_A)         {leer_instrumento(0);}
249.                 break;
250.

```

```

251.
252.         case 1: if (INCLINOMETRO_B) {leer_instrumento(1);}
253.                 break;
254.
255.         case 2: if (GIROSCOPO_A) {leer_instrumento(2);}
256.                 break;
257.
258.         case 3: if (GIROSCOPO_B) {leer_instrumento(3);}
259.                 break;
260.
261.         case 4: if (GIROSCOPO_ANALOGICO) {leer_instrumento(4);}
262.                 break;
263.     }
264.
265.     estado++;
266.     if (estado > 4)
267.     {
268.         estado = 0;
269.     }
270.
271. }
272.
273. boolean_t leer_instrumento(char instrumento)
274. {
275.     if (leyendo_instrumento == TRUE)
276.         return FALSE;
277.
278.     inst_a_leer = instrumento;
279.     leyendo_instrumento = TRUE;
280.     return TRUE;
281. }
282.
283. #pragma CODE_SECTION(lectura_instrumento, "ramfuncs");
284. void lectura_instrumento(void)
285. {
286.     static boolean_t esperando = FALSE;
287.     static boolean_t dato_listo = FALSE;
288.     static Uint16 respuesta = 0;
289.
290.     Uint16 requerimiento;
291.
292.     if (leyendo_instrumento == TRUE) //Si estoy en proceso de lectura
293.     {
294.         if (esperando == FALSE) //Si debo enviar algo
295.         {
296.
297.             switch (inst_a_leer) {
298.                 case 0: requerimiento = INC_INCL_180_OUT; //INCLINOMETRO_A
299.                         break;
300.
301.                 case 1: requerimiento = INC_INCL_180_OUT; //INCLINOMETRO_B
302.                         break;
303.
304.                 case 2: requerimiento = GY_GYRO_OUT; //GIROSCOPO_A
305.                         break;
306.
307.                 case 3: requerimiento = GY_GYRO_OUT; //GIROSCOPO_B
308.                         break;
309.
310.                 case 4: requerimiento = INC_AUX_ADC; //GIROSCOPO_ANALOGICO
311.                         break;
312.             }
313.
314.             if (inst_a_leer == 4) //Giroscopo analógico se mide desde Inclinometro 1
315.                 transmitir_dato(requerimiento, 1, &respuesta, &dato_listo);
316.             else transmitir_dato(requerimiento, inst_a_leer, &respuesta, &dato_listo);
317.
318.             esperando = TRUE;
319.         } else //Si estoy esperando a que llegue algo
320.         {
321.             if (dato_listo) //Si llegó el dato desde SPI
322.             {
323.                 if (respuesta != 0xFFFF) //Si el dato es válido
324.                 {
325.                     switch (inst_a_leer) {
326.                         case 0: sumar_inclinacion(respuesta); //INCLINOMETRO_A
327.                                 break;
328.
329.                         case 1: sumar_inclinacion(respuesta); //INCLINOMETRO_B
330.                                 break;
331.
332.                         case 2: sumar_vel_giro(respuesta); //GIROSCOPO_A
333.                                 break;
334.

```

```

335.         case 3: sumar_vel_giro(respuesta); //GIROSCOPO_B
336.                 break;
337.
338.         case 4: //GIROSCOPO_ANALOGICO
339.                 break;
340.
341.     }
342.     //Se terminó la lectura
343.     esperando = FALSE;
344.     leyendo_instrumento = FALSE;
345.     dato_listo = FALSE;
346. }
347. }
348. }
349.
350.
351. }
352.
353. int32 hal_instrumentos_get_inclinacion(void)
354. {
355.     return PASO_INC * (suma_inclinacion / MUESTRAS_INCL) + OFFSET_INC;
356. }
357.
358. int32 hal_instrumentos_get_vel_giro(void)
359. {
360.     return PASO_VEL_GIRO * (suma_vel_giro / MUESTRAS_GYRO);
361. }
362.
363. void Seleccionar_Instrumento(char inst)
364. {
365.     switch (inst)
366.     {
367.         case 0: GpioDataRegs.GPACLEAR.bit.GPIO8 = 1;
368.                 GpioDataRegs.GPACLEAR.bit.GPIO10 = 1;
369.                 GpioDataRegs.GPACLEAR.bit.GPIO22 = 1;
370.                 break;
371.
372.         case 1: GpioDataRegs.GPASET.bit.GPIO8 = 1;
373.                 GpioDataRegs.GPACLEAR.bit.GPIO10 = 1;
374.                 GpioDataRegs.GPACLEAR.bit.GPIO22 = 1;
375.                 break;
376.
377.         case 2: GpioDataRegs.GPACLEAR.bit.GPIO8 = 1;
378.                 GpioDataRegs.GPASET.bit.GPIO10 = 1;
379.                 GpioDataRegs.GPACLEAR.bit.GPIO22 = 1;
380.                 break;
381.
382.         case 3: GpioDataRegs.GPASET.bit.GPIO8 = 1;
383.                 GpioDataRegs.GPASET.bit.GPIO10 = 1;
384.                 GpioDataRegs.GPACLEAR.bit.GPIO22 = 1;
385.                 break;
386.
387.         case 4: GpioDataRegs.GPACLEAR.bit.GPIO8 = 1;
388.                 GpioDataRegs.GPACLEAR.bit.GPIO10 = 1;
389.                 GpioDataRegs.GPASET.bit.GPIO22 = 1;
390.                 break;
391.
392.         case 5: GpioDataRegs.GPASET.bit.GPIO8 = 1;
393.                 GpioDataRegs.GPACLEAR.bit.GPIO10 = 1;
394.                 GpioDataRegs.GPASET.bit.GPIO22 = 1;
395.                 break;
396.
397.         case 6: GpioDataRegs.GPACLEAR.bit.GPIO8 = 1;
398.                 GpioDataRegs.GPASET.bit.GPIO10 = 1;
399.                 GpioDataRegs.GPASET.bit.GPIO22 = 1;
400.                 break;
401.
402.         default:GpioDataRegs.GPASET.bit.GPIO8 = 1;
403.                 GpioDataRegs.GPASET.bit.GPIO10 = 1;
404.                 GpioDataRegs.GPASET.bit.GPIO22 = 1;
405.                 break;
406.     }
407. }
408.
409.
410. void hal_instrumentos_set_horizontal(char instr)
411. {
412.     enviar_orden_a_instrumento(instr, INC_NULL);
413. }
414.
415. void enviar_orden_a_instrumento(char id, Uint16 orden)
416. {
417.     static boolean_t dato_listo = FALSE;
418.     static Uint16 respuesta = 0;

```

```

419.     transmitir_dato(orden, id, &respuesta, &dato_listo);
420. }
421. }
422.
423. //Estas funciones agregan un elemento al buffer circular
424. #pragma CODE_SECTION(sumar_inclinacion, "ramfuncs");
425. void sumar_inclinacion(Uint16 incl)
426. {
427.     int aux;
428.     int aux2;
429.     aux = MASK14 & incl;
430.
431.     //Ajuste para complemento 2 de 14 bits
432.     if (aux > 8191)
433.     {
434.         aux = (16384 - aux)*-1;
435.     }
436.
437.     aux2 = suma_inclinacion / MUESTRAS_INCL;
438.
439.     d_inc = aux2 - aux;
440.     if (d_inc > LIMITE_AC_INC)
441.     {
442.         aux = aux2 - LIMITE_AC_INC;
443.     } else if (d_inc < (-1*LIMITE_AC_INC))
444.     {
445.         aux = aux2 + LIMITE_AC_INC;
446.     }
447.
448.
449.     suma_inclinacion += aux;
450.     suma_inclinacion -= vect_inclinacion[indice_inclinacion];
451.     vect_inclinacion[indice_inclinacion] = aux;
452.     indice_inclinacion++;
453.
454.
455.     if (indice_inclinacion >= MUESTRAS_INCL)
456.         indice_inclinacion = 0;
457. }
458.
459. #pragma CODE_SECTION(sumar_vel_giro, "ramfuncs");
460. void sumar_vel_giro(Uint16 gyro)
461. {
462.     int aux;
463.     int vel_anterior;
464.
465.     aux = MASK14 & gyro;
466.
467.     //Ajuste para complemento 2 de 14 bits
468.     if (aux > 8191)
469.     {
470.         aux = (16384 - aux)*-1;
471.     }
472.
473.     if (absoluto(aux) > UMBRAL_TRIP)
474.     {
475.         hal_pwm_trip();
476.         hal_gpio_emitir_tono(D_TONO_ERROR, FALSE);
477.         set_orden_inicio(FALSE);
478.     }
479.
480.     vel_anterior = suma_vel_giro / MUESTRAS_GYRO;
481.
482.     if (vel_anterior - aux > LIMITE_ACEL) //Limitamos la aceleración
483.     {
484.         aux = vel_anterior - LIMITE_ACEL;
485.     } else if (aux - vel_anterior > LIMITE_ACEL)
486.     {
487.         aux = vel_anterior + LIMITE_ACEL;
488.     }
489.
490.     suma_vel_giro += aux;
491.     suma_vel_giro -= vect_vel_giro[indice_vel_giro];
492.     vect_vel_giro[indice_vel_giro] = aux;
493.     indice_vel_giro++;
494.
495.     if (indice_vel_giro >= MUESTRAS_GYRO)
496.         indice_vel_giro = 0;
497. }

```

```

1. //#####
2. //Archivo: hal_timer.c
3. //Titulo: Módulo HAL Timer
4. //Descripcion: Módulo administrador del temporizador del sistema.
5. //
6. //
7. //Version:
8. //2010-07-26: v1.0 - Funcionalidad basica para administracion temporal.
9. //#####
10.
11. //Librerias del sistema
12. #include "..\..\Include\header\DSP280x_Device.h" // Headers para dispositivo
13. #include "..\..\Include\header\DSP280x_DefDisp.h" // Definiciones de Dispositivo
14.
15. #include "hal_timer.h"
16.
17. #define TIMER_OVERFLOW 4294967290
18.
19. Uint32 ultima_ejecucion[NUMERO_MODULOS];
20. boolean_t debe_ejecutarse[NUMERO_MODULOS];
21.
22. //prototipos de funciones de este archivo
23. interrupt void hal_timer_isr(void);
24.
25. void Init_hal_timer(void){
26.     int i;
27.
28.     //Inicializacion de interrupciones
29.     DINT; //Deshabilita interrupciones
30.     InitPieCtrl(); //Registros de control PIE en valor predeterminado
31.
32.     //Deshabilitar interrupciones de CPU y limpia todos sus flags
33.     IER = 0x0000;
34.     IFR = 0x0000;
35.
36.     InitPieVectTable(); //Inicializa PIE con los punteros a las ISR
37.
38.     //Asigna la ISR para el timer 0
39.     EALLOW;
40.     PieVectTable.TINT0 = &hal_timer_isr;
41.     EDIS;
42.
43.     InitCpuTimers();
44.
45.     //Configura el Timer, para frec de 100MHz un periodo de 100uS (en uS)
46.     ConfigCpuTimer(&CpuTimer0, 100, 100);
47.
48.     //Inicializa el vector de tiempos
49.     for (i=0; i < NUMERO_MODULOS; i++)
50.     {
51.         ultima_ejecucion[i] = 0;
52.         debe_ejecutarse[i] = FALSE;
53.     }
54.
55.
56.     StartCpuTimer0();
57.     IER |= M_INT1;
58.     PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
59.
60.     EINT; // Enable Global interrupt INTM
61.     ERTM; // Enable Global realtime interrupt DBGM
62.
63. }
64.
65. #pragma CODE_SECTION(hal_timer_ejecutar, "ramfuncs");
66. void hal_timer_ejecutar(void)
67. {
68.     Uint32 lapso;
69.
70.     lapso = hal_timer_diff(ultima_ejecucion[MOD_CONTROL], CpuTimer0.InterruptCount);
71.     if (lapso >= PERIODO_CONTROL) debe_ejecutarse[MOD_CONTROL] = TRUE;
72.
73.     lapso = hal_timer_diff(ultima_ejecucion[MOD_COMANDO], CpuTimer0.InterruptCount);
74.     if (lapso >= PERIODO_COMANDO) debe_ejecutarse[MOD_COMANDO] = TRUE;
75.
76.     lapso = hal_timer_diff(ultima_ejecucion[MOD_HAL_INST], CpuTimer0.InterruptCount);
77.     if (lapso >= PERIODO_HAL_INST) debe_ejecutarse[MOD_HAL_INST] = TRUE;
78.
79.     lapso = hal_timer_diff(ultima_ejecucion[MOD_HAL_ADC], CpuTimer0.InterruptCount);
80.     if (lapso >= PERIODO_HAL_ADC) debe_ejecutarse[MOD_HAL_ADC] = TRUE;
81.
82.     lapso = hal_timer_diff(ultima_ejecucion[MOD_HAL_ENCODER], CpuTimer0.InterruptCount);
83.     if (lapso >= PERIODO_HAL_ENCODER) debe_ejecutarse[MOD_HAL_ENCODER] = TRUE;
84.

```

```

85.     lapso = hal_timer_diff(ultima_ejecucion[MOD_HAL_CORRIENTE], CpuTimer0.InterruptCount);
86.     if (lapso >= PERIODO_HAL_CORRIENTE) debe_ejecutarse[MOD_HAL_CORRIENTE] = TRUE;
87.
88.     lapso = hal_timer_diff(ultima_ejecucion[MOD_HAL_GPIO], CpuTimer0.InterruptCount);
89.     if (lapso >= PERIODO_HAL_GPIO) debe_ejecutarse[MOD_HAL_GPIO] = TRUE;
90.
91. }
92.
93. //Devuelve TRUE si ya es tiempo de ejecutar el modulo
94. #pragma CODE_SECTION(hal_timer_mod_triggered, "ramfuncs");
95. boolean_t hal_timer_trigger(char modulo)
96. {
97.     return debe_ejecutarse[modulo];
98. }
99.
100. //Notifica que un modulo ha sido ejecutado
101. #pragma CODE_SECTION(hal_timer_mod_triggered, "ramfuncs");
102. void hal_timer_mod_triggered(char modulo)
103. {
104.     debe_ejecutarse[modulo] = FALSE;
105.     ultima_ejecucion[modulo] = CpuTimer0.InterruptCount;
106. }
107.
108. #pragma CODE_SECTION(hal_timer_diff, "ramfuncs");
109. Uint32 hal_timer_diff( Uint32 tstart, Uint32 tend)
110. {
111.     Uint32 tmp;
112.     if ( tstart <= tend )
113.     {
114.         tmp = tend - tstart;
115.     }
116.     else
117.     {
118.         tmp = TIMER_OVERFLOW - tstart;
119.         tmp += tend;
120.     }
121.     return tmp;
122. }
123.
124. Uint32 hal_timer_get_time(void)
125. {
126.     return CpuTimer0.InterruptCount;
127. }
128.
129. interrupt void hal_timer_isr(void)
130. {
131.     CpuTimer0.InterruptCount++;
132.     if (CpuTimer0.InterruptCount >= TIMER_OVERFLOW)
133.         CpuTimer0.InterruptCount = 0;
134.     // Interrupcion atendida, se pueden recibir nuevas interrupciones del grupo 1
135.     PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
136. }

```

```

1. //#####
2. //Archivo: hal_serial.c
3. //Titulo: Módulo HAL Serial
4. //Descripcion: Módulo administrador del puerto serial para comunicacion por
5. //      comandos.
6. //
7. //Version:
8. //2010-07-19: v1.0 - Envio y recepcion de comandos flotantes y enteros
9. //#####
10.
11. //Librerias del sistema
12. #include "..\..\Include\header\DSP280x_Device.h" // Headers para dispositivo
13. #include "..\..\Include\header\DSP280x_DefDisp.h" // Definiciones de Dispositivo
14.
15. #include "hal_serial.h"
16.
17. //Prototipos de funciones de este archivo
18. void enviar_comando(COMANDO c); //envia un comando por serial sin detener el programa
19. void scia_xmit(int a); //Transmite un Byte por SCI
20. void send_msg(char *msg); //Transmite un mensaje por SCI
21.
22. COMANDO comandos[4]; //Cola de comandos
23. int indice_comando = -1; //Indice de la cola
24. Uint16 comando_saliente[7]; //Comando en espera de ser enviado
25. Uint16 comando_entrante[6]; //Comando en recepcion
26. int indice_comando_saliente = -1; //Indica si hay comando enviandose y la posicion del array enviada
27. int indice_comando_entrante = -1; //Indica si se esta recibiendo un comando y en qué indice va
28.
29. void Init_hal_serial(void)
30. {
31.
32. // === Inicializa los GPIO === //
33.     EALLOW;
34.     /* Enable internal pull-up for the selected pins */
35.     // Pull-ups can be enabled or disabled disabled by the user.
36.     // This will enable the pullups for the specified pins.
37.
38.     GpioCtrlRegs.GPAPUD.bit.GPIO28 = 0; // Enable pull-up for GPIO28 (SCIRXDA)
39.     GpioCtrlRegs.GPAPUD.bit.GPIO29 = 0; // Enable pull-up for GPIO29 (SCITXDA)
40.
41.     /* Set qualification for selected pins to asynch only */
42.     // Inputs are synchronized to SYSCLKOUT by default.
43.     // This will select asynch (no qualification) for the selected pins.
44.
45.     GpioCtrlRegs.GPAQSEL2.bit.GPIO28 = 3; // Asynch input GPIO28 (SCIRXDA)
46.
47.     /* Configure SCI-A pins using GPIO regs*/
48.     // This specifies which of the possible GPIO pins will be SCI functional pins.
49.
50.     GpioCtrlRegs.GPAMUX2.bit.GPIO28 = 1; // Configure GPIO28 for SCIRXDA operation
51.     GpioCtrlRegs.GPAMUX2.bit.GPIO29 = 1; // Configure GPIO29 for SCITXDA operation
52.
53.     EDIS;
54.     // === Fin Inicializacion los GPIO === //
55.
56.     // === Inicializa FIFO de SCI === //
57.     SciaRegs.SCIFFTX.all=0xE040;
58.     SciaRegs.SCIFFRX.all=0x204f;
59.     SciaRegs.SCIFFCT.all=0x0;
60.     // === Fin Inicializacion FIFO de SCI === //
61.
62.     // === Configura el SCI === //
63.     SciaRegs.SCICCR.all =0x0007; // 1 stop bit, No loopback
64.                                     // No parity,8 char bits,
65.                                     // async mode, idle-line protocol
66.     SciaRegs.SCICTL1.all =0x0003; // enable TX, RX, internal SCICLK,
67.                                     // Disable RX ERR, SLEEP, TXWAKE
68.     SciaRegs.SCICTL2.all =0x0003;
69.     SciaRegs.SCICTL2.bit.TXINTENA =1;
70.     SciaRegs.SCICTL2.bit.RXBKINTENA =1;
71.
72.     /* SciaRegs.SCIHBAUD =0x0001; // 9600 baud @LSPCLK = 20MHz.
73.     SciaRegs.SCILBAUD =0x0044;*/
74.
75.     SciaRegs.SCIHBAUD =0x0000; // 19200 baud @LSPCLK = 20MHz.
76.     SciaRegs.SCILBAUD =0x00A1;
77.
78.     SciaRegs.SCICTL1.all =0x0023; // Relinquish SCI from Reset
79.
80.     // === Fin Configuracion SCI === //
81.
82.
83. }
84.

```

```

85. #pragma CODE_SECTION(hal_serial_ejecutar, "ramfuncs");
86. void hal_serial_ejecutar(void)
87. {
88.     //Si hay comandos pendientes por enviar, se procede a ello
89.     if (indice_comando_saliente != -1) //Pero si no hay nada que enviar, se continua la ejecucion
90.     {
91.         if (SciaRegs.SCIFFTX.bit.TXFFST == 0) //si se puede enviar algo
92.         {
93.             SciaRegs.SCITXBUF = comando_saliente[indice_comando_saliente];
94.             if (indice_comando_saliente > 5) //Si ya termino de enviar el comando
95.             {
96.                 indice_comando_saliente = -1;
97.             } else indice_comando_saliente ++;
98.         }
99.     }
100.
101.     //Si hay algo que recibir, se procede a ello y si no, se deja ejecutar lo demás
102.     if (SciaRegs.SCIFFRX.bit.RXFFST ==1)
103.     {
104.         indice_comando_entrante++;
105.         comando_entrante[indice_comando_entrante] = SciaRegs.SCIRXBUF.all;
106.         if (indice_comando_entrante > 4) //Si termine de recibir un comando, lo ingreso al sistema
107.         {
108.             if (indice_comando < 3) //Aún hay espacio en la cola (si no, se ignora)
109.             {
110.                 indice_comando ++;
111.                 comandos[indice_comando].Codigo = comando_entrante[0];
112.                 comandos[indice_comando].ID = comando_entrante[1];
113.                 comandos[indice_comando].n_elemento = indice_comando;
114.                 comandos[indice_comando].valor_ent = comando_entrante[2];
115.                 comandos[indice_comando].valor_ent = (comandos[indice_comando].valor_ent << 8)
+ comando_entrante[3];
116.                 comandos[indice_comando].valor_ent = (comandos[indice_comando].valor_ent << 8)
+ comando_entrante[4];
117.                 comandos[indice_comando].valor_ent = (comandos[indice_comando].valor_ent << 8)
+ comando_entrante[5];
118.             }
119.             indice_comando_entrante = -1;
120.         }
121.     }
122. }
123.
124. boolean_t hal_serial_puede_enviar(void)
125. {
126.     if (indice_comando_saliente == -1)
127.     {
128.         return TRUE;
129.     } else
130.     {
131.         return FALSE;
132.     }
133. }
134.
135. boolean_t hal_serial_hay_comando(void)
136. {
137.     if (indice_comando < 0)
138.     {
139.         return FALSE;
140.     } else
141.     {
142.         return TRUE;
143.     }
144. }
145.
146. COMANDO hal_serial_get_comando(void)
147. {
148.     return comandos[0]; //Obtiene el comando mas antiguo
149. }
150.
151. void hal_serial_enviar_comando(COMANDO c)
152. {
153.     Uint16 indice;
154.
155.     enviar_comando(c); //Envía el comando C
156.
157.     if (c.n_elemento == 255) //Si pertenece a un flujo no se despeja cola y terminamos
158.         return;
159.
160.     //Y se despeja la cola
161.     indice = c.n_elemento;
162.
163.     while (indice < 3)
164.     {
165.         comandos[indice].Codigo = comandos[indice+1].Codigo;

```



```

166.         comandos[indice].ID = comandos[indice+1].ID;
167.         comandos[indice].n_elemento = comandos[indice+1].n_elemento;
168.         comandos[indice].valor_ent = comandos[indice+1].valor_ent;
169.         indice++;
170.     }
171.
172.     indice_comando--;
173. }
174.
175.
176. void enviar_comando(COMANDO c){
177.     Uint32 mascara = 0x000000FF;
178.
179.     if(indice_comando_saliente == -1) //No hay comando en proceso, se puede enviar uno nuevo
180.     {
181.
182.         comando_saliente[0] = c.Codigo;
183.         comando_saliente[1] = c.ID;
184.
185.         //Envio numero entero
186.         comando_saliente[2] = mascara & (c.valor_ent >> 24);
187.         comando_saliente[3] = mascara & (c.valor_ent >> 16);
188.         comando_saliente[4] = mascara & (c.valor_ent >> 8);
189.         comando_saliente[5] = mascara & c.valor_ent;
190.         comando_saliente[6] = 255;
191.
192.         //Una vez lleno el comando saliente, se indica que hay algo para enviar
193.         indice_comando_saliente = 0;
194.     }
195. }

```

```

1. //#####
2. //Archivo: hal_gpio.c
3. //Titulo: Módulo HAL GPIO
4. //Descripcion: Módulo administrador de entradas y salidas generales más el
5. //          altavoz del sistema.
6. //
7. //Version:
8. //2010-07-27: v1.0 - Creación del Archivo
9. //#####
10.
11. //Librerias del sistema
12. #include "..\..\Include\header\DSP280x_Device.h" // Headers para dispositivo
13. #include "..\..\Include\header\DSP280x_DefDisp.h" // Definiciones de Dispositivo
14.
15. #include "hal_gpio.h"
16.
17. Uint32 contador_tono;
18. Uint32 periodo_tono;
19. boolean_t usar_loop;
20. boolean_t activo;
21.
22. void Init_hal_gpio(void)
23. {
24.     //Inicializacion GPIO
25.     EALLOW;
26.     GpioCtrlRegs.GPAPUD.bit.GPIO31 = 0; // Activar Pull up en GPIO31 (buzzer)
27.     GpioDataRegs.GPASET.bit.GPIO31 = 0; // Poner un cero en ese pin
28.     GpioCtrlRegs.GPAMUX2.bit.GPIO31 = 0; // GPIO31 = GPIO31
29.     GpioCtrlRegs.GPADIR.bit.GPIO31 = 1; // GPIO31 = output
30.
31.     GpioCtrlRegs.GPBPUD.bit.GPIO32 = 0; // Activar Pull up en GPIO32 (JP2)
32.     GpioDataRegs.GPBSET.bit.GPIO32 = 0; // Poner un cero en ese pin
33.     GpioCtrlRegs.GPBMUX1.bit.GPIO32 = 0; // GPIO32 = GPIO32
34.     GpioCtrlRegs.GPBDIR.bit.GPIO32 = 1; // GPIO32 = output
35.
36.     GpioCtrlRegs.GPBPUD.bit.GPIO33 = 0; // Activar Pull up en GPIO33 (JP1)
37.     GpioDataRegs.GPBSET.bit.GPIO33 = 0; // Poner un cero en ese pin
38.     GpioCtrlRegs.GPBMUX1.bit.GPIO33 = 0; // GPIO33 = GPIO33
39.     GpioCtrlRegs.GPBDIR.bit.GPIO33 = 1; // GPIO33 = output
40.     EDIS;
41.
42.     contador_tono = 0;
43.     periodo_tono = 0;
44.     usar_loop = FALSE;
45.     activo = FALSE;
46. }
47.
48. #pragma CODE_SECTION(hal_gpio_ejecutar, "ramfuncs");
49. void hal_gpio_ejecutar(void)
50. {
51.     //Operacion de altavoz
52.     if (activo == FALSE)
53.     {
54.         //Si no esta activo se silencia todo
55.         GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;
56.     } else
57.     {
58.         contador_tono++;
59.         if (contador_tono < periodo_tono)
60.         {
61.             //dentro del periodo en que suena debe alternar el encendido del pin
62.             GpioDataRegs.GPATOGGLE.bit.GPIO31 = 1;
63.         } else
64.         {
65.             //en el periodo de silencio mantiene el pin en 0
66.             GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;
67.         }
68.
69.         if (contador_tono > 2*periodo_tono)
70.         {
71.             if (usar_loop == TRUE)
72.             {
73.                 contador_tono = 0;
74.             } else
75.             {
76.                 contador_tono = 0;
77.                 activo = FALSE;
78.             }
79.         }
80.
81.     }
82. }
83. }
84.

```

```
85. void hal_gpio_emitir_tono(Uint32 periodo, boolean_t loop)
86. {
87.     periodo_tono = periodo;
88.     usar_loop = loop;
89.     activo = TRUE;
90. }
91.
92. void hal_gpio_silenciar_tono(void)
93. {
94.     activo = FALSE;
95. }
```

```

1. //#####
2. //Archivo: hal_adc.c
3. //Titulo: Módulo HAL ADC
4. //Descripcion: Módulo administrador de entradas analógicas.
5. //
6. //Version:
7. //2010-09-24: v1.0 - Creación del Archivo
8. //#####
9.
10. //Librerias del sistema
11. #include "..\..\Include\header\DSP280x_Device.h" // Headers para dispositivo
12. #include "..\..\Include\header\DSP280x_DefDisp.h" // Definiciones de Dispositivo
13.
14. #include "hal_adc.h"
15.
16. #define ADC_MODCLK 0x4 // HSPCLK = SYSCLKOUT/2*ADC_MODCLK2 = 100/(2*4) = 12.5MHz
17. #define ADC_SHCLK 0xf // S/H width in ADC module periods = 16 ADC clocks
18. #define ADC_CKPS 0x1 // ADC module clock = HSPCLK/2*ADC_CKPS = 12.5MHz/(1*2) = 6.25MHz
19. // for 60 MHz devices: ADC module clk = 7.5MHz/(1*2) = 3.75MHz
20.
21. Uint16 mando_direccion[BUFF_ADC];
22. Uint16 bateria[BUFF_ADC];
23. Uint16 presion[BUFF_ADC];
24. Uint32 suma_direccion;
25. Uint32 suma_bateria;
26. Uint32 suma_presion;
27. Uint16 offset_direccion;
28. char indice_direccion;
29. char indice_bateria;
30. char indice_presion;
31.
32. boolean_t adc_iniciado;
33.
34. void Init_hal_adc(void)
35. {
36.     Uint16 i,j;
37.
38.     suma_direccion = 0;
39.     indice_direccion = 0;
40.     suma_bateria = 0;
41.     indice_bateria = 0;
42.     suma_presion = 0;
43.     indice_presion = 0;
44.
45.     offset_direccion = MITAD_DIRECCION;
46.
47.     for (i=0; i<BUFF_ADC; i++)
48.     {
49.         mando_direccion[i] = 0;
50.         bateria[i] = 0;
51.         presion[i] = 0;
52.     }
53.
54.     EALLOW;
55.     SysCtrlRegs.HISPCP.all = ADC_MODCLK; // HSPCLK = SYSCLKOUT/ADC_MODCLK
56.     EDIS;
57.
58.     AdcRegs.ADCTRL3.all = 0x00E0; // Power up bandgap/reference/ADC circuits
59.     /* Delay, mínimo 5 ms */
60.     for(i = 0; i < 65000; i++)
61.         for(j = 0; j < 50; j++);
62.
63.     // Inicializando ADC
64.     AdcRegs.ADCTRL1.bit.ACQ_PS = ADC_SHCLK; // Ancho de ventana: 16 ADC clocks
65.     AdcRegs.ADCTRL3.bit.ADCCLKPS = ADC_CKPS; // 6.25 Mhz
66.     AdcRegs.ADCTRL1.bit.SEQ_CASC = 0; // Cascaded mode off
67.     AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x7; //ADC Canal 1
68.     AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 0x6; //ADC Canal 2
69.     AdcRegs.ADCCHSELSEQ1.bit.CONV02 = 0x5; //ADC Canal 3
70.     AdcRegs.ADCCHSELSEQ1.bit.CONV03 = 0x4; //ADC Canal 4
71.
72.     AdcRegs.ADCMAXCONV.bit.MAX_CONV1 = 0x3; // convert and store in 4 results registers
73.     AdcRegs.ADCTRL1.bit.CONT_RUN = 1; // Setup continuous run
74.
75.     adc_iniciado = FALSE;
76. }
77.
78. #pragma CODE_SECTION(hal_adc_ejecutar, "ramfuncs");
79. void hal_adc_ejecutar(void)
80. {
81.     int16 valor_temp;
82.     int16 valor_ant;
83.     Uint16 i;
84.

```

```

85.  if (adc_iniciado == FALSE)
86.  {
87.      /* Se inicia la conversión del ADC */
88.      AdcRegs.ADCCTRL2.all = 0x2000;
89.
90.      adc_iniciado = TRUE;
91.  }
92.
93.  if (AdcRegs.ADCST.bit.INT_SEQ1 == 1) //Si ha terminado una secuencia de conversión
94.  {
95.      if (suma_direccion == 0) // Marca de que no se han inicializado las variables
96.      {
97.          for (i=0; i<BUFF_ADC; i++)
98.          {
99.              suma_direccion += AdcRegs.ADCRESULT0>>4;
100.             mando_direccion[i] = AdcRegs.ADCRESULT0>>4;
101.         }
102.     }
103.
104.     if (suma_bateria == 0) // Marca de que no se han inicializado las variables
105.     {
106.         for (i=0; i<BUFF_ADC; i++)
107.         {
108.             suma_bateria += AdcRegs.ADCRESULT1>>4;
109.             bateria[i] = AdcRegs.ADCRESULT1>>4;
110.         }
111.     }
112.
113.     if (suma_presion == 0) // Marca de que no se han inicializado las variables
114.     {
115.         for (i=0; i<BUFF_ADC; i++)
116.         {
117.             suma_presion += AdcRegs.ADCRESULT2>>4;
118.             presion[i] = AdcRegs.ADCRESULT2>>4;
119.         }
120.     }
121.
122.     /* MANDO DE DIRECCION */
123.     valor_temp = AdcRegs.ADCRESULT0>>4; //valor del mando de dirección
124.
125.     valor_ant = suma_direccion/BUFF_ADC;
126.
127.     if (valor_temp - valor_ant > LIMITE_AC_DIR)
128.     {
129.         valor_temp = valor_ant + LIMITE_AC_DIR;
130.     } else {
131.         if (valor_ant - valor_temp > LIMITE_AC_DIR)
132.         {
133.             valor_temp = valor_ant - LIMITE_AC_DIR;
134.         }
135.     }
136.
137.
138.     suma_direccion -= mando_direccion[indice_direccion];
139.     suma_direccion += valor_temp;
140.     mando_direccion[indice_direccion] = valor_temp;
141.
142.     indice_direccion++;
143.     if (indice_direccion >= BUFF_ADC) indice_direccion = 0;
144.
145.     /* ESTADO BATERÍAS */
146.     valor_temp = AdcRegs.ADCRESULT1>>4; //valor del estado de baterías
147.     valor_ant = suma_bateria/BUFF_ADC;
148.
149.     if ((valor_temp < 4050) && (valor_ant < 4050)) //4095 es el valor para la alimentación
desconectada, ese no se filtra
150.     {
151.         if (valor_temp > valor_ant)
152.         {
153.             valor_temp = valor_ant + 1;
154.         } else {
155.             valor_temp = valor_ant - 1;
156.         }
157.     }
158.
159.     suma_bateria -= bateria[indice_bateria];
160.     suma_bateria += valor_temp;
161.     bateria[indice_bateria] = valor_temp;
162.
163.     indice_bateria++;
164.     if (indice_bateria >= BUFF_ADC) indice_bateria = 0;
165.
166.     /* SENSOR PRESION */
167.     valor_temp = AdcRegs.ADCRESULT2>>4; //valor del sensor de presion

```

```

168.         valor_ant = suma_presion/BUFF_ADC;
169.
170.         if (valor_temp - valor_ant > LIMITE_AC_PRES)
171.         {
172.             valor_temp = valor_ant + LIMITE_AC_PRES;
173.         } else {
174.             if (valor_ant - valor_temp > LIMITE_AC_PRES)
175.             {
176.                 valor_temp = valor_ant - LIMITE_AC_PRES;
177.             }
178.         }
179.
180.         suma_presion -= presion[indice_presion];
181.         suma_presion += valor_temp;
182.         presion[indice_presion] = valor_temp;
183.
184.         indice_presion++;
185.         if (indice_presion >= BUFF_ADC) indice_presion = 0;
186.
187.         /* Avisamos que se recibió la interrupcion */
188.         AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;
189.     }
190. }
191.
192.
193. int16 hal_adc_get_direccion(void)
194. {
195.     int16 valor = (suma_direccion/BUFF_ADC) - offset_direccion;
196.     if (valor > 300) return 300;
197.     if (valor < -300) return -300;
198.     return valor;
199. }
200.
201. int16 hal_adc_get_bateria(void)
202. {
203.     return ((suma_bateria/BUFF_ADC)+402)/7; //los números son la linealización de la lectura calibrada
204. }
205.
206. int16 hal_adc_get_presion(void)
207. {
208.     return suma_presion/BUFF_ADC;
209. }

```

```

1. //#####
2. //Archivo: hal_pwm.c
3. //Titulo: Módulo HAL PWM
4. //Descripcion: Maneja los PWM que accionan los motores.
5. //
6. //
7. //Version:
8. //2010-08-12: v0.1 - Creacion de Archivo. Rodrigo Maureira.
9. //#####
10.
11. //Librerias del sistema
12. #include "..\..\Include\header\DSP280x_Device.h" // Headers para dispositivo
13. #include "..\..\Include\header\DSP280x_DefDisp.h" // Definiciones de Dispositivo
14.
15. #include "hal_pwm.h"
16. #include "hal_main.h"
17.
18. Uint16 periodo_contador;
19. Uint16 duty_maximo;
20. Uint16 duty_abajo;
21. boolean_t pwm_funcionando;
22.
23. void _init_pwm(volatile struct EPWM_REGS *EPwmxRegs){
24.     /* Time Base */
25.     EPwmxRegs->TBPRD = periodo_contador; //Define el periodo del contador
26.     EPwmxRegs->TBPHS.all = 0; //Fase cero
27.     EPwmxRegs->TBCTR = 0; //Reinicio contador
28.     EPwmxRegs->TBCTL.bit.CTRMODE = TB_COUNT_UP; //UP-COUNT MODE
29.     EPwmxRegs->TBCTL.bit.PHSEN = TB_ENABLE; // SINCRONIZACION POR FASE
30.     EPwmxRegs->TBCTL.bit.PRDL = TB_SHADOW; //Se activa el Shadow Register del counter
31.     EPwmxRegs->TBCTL.bit.SYNCOSEL = TB_SYNC_IN; // SyncOut = SyncIn
32.     EPwmxRegs->TBCTL.bit.HSPCLKDIV = TB_DIV1; //Resolución del PWM según SYSCLK
33.     EPwmxRegs->TBCTL.bit.CLKDIV = TB_DIV1; //Resolución del PWM según SYSCLK
34.
35.     /*Counter Compare*/
36.     EPwmxRegs->CMPA.half.CMPA = 0; //Compare A se activa al inicio (por defecto)
37.     EPwmxRegs->CMPB = 0; //Compare B se activa al inicio (por
38.     defecto)
39.     EPwmxRegs->CMPCTL.bit.SHDWAMODE = CC_SHADOW; //se activa el shadow register del Compare A
40.     EPwmxRegs->CMPCTL.bit.SHDWBMODE = CC_SHADOW; //se activa el shadow register del Compare B
41.     EPwmxRegs->CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; //se carga el valor en el registro en el siguiente
42.     zero
43.     EPwmxRegs->CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; //se carga el valor en el registro en el siguiente
44.     zero
45.
46.     /*Generacion de pulsos*/
47.     EPwmxRegs->AQCTLA.bit.ZRO = AQ_SET; //Pin de PWM A en 1 cuando la cuenta llega a cero
48.     EPwmxRegs->AQCTLA.bit.CAU = AQ_CLEAR; //Pin de PWM A en 0 cuando la cuenta supera el contador
49.     EPwmxRegs->AQCTLB.bit.ZRO = AQ_SET; //Pin de PWM B en 1 cuando la cuenta llega a cero
50.     EPwmxRegs->AQCTLB.bit.CBU = AQ_CLEAR; //Pin de PWM B en 0 cuando la cuenta supera el contador
51.
52.     /*Se fija en 0 el ciclo de trabajo */
53.     EPwmxRegs->CMPA.half.CMPA = 0;
54.     EPwmxRegs->CMPB = 0;
55. }
56.
57. void Init_hal_pwm(void)
58. {
59.     periodo_contador = 100000/FREQ; //Periodo del contador del pwm
60.     duty_abajo = (periodo_contador/1000)*900; //Ciclo de trabajo de los mosfet de abajo
61.     duty_maximo = (periodo_contador/1000)*DUTY_MAX; //Ciclo de trabajo maximo
62.     pwm_funcionando = FALSE; //Inicia desactivado
63.
64.     //Configuración de los GPIO para PWM
65.     EALLOW;
66.     GpioCtrlRegs.GPAPUD.bit.GPIO0 = 0; // Enable pull-up on GPIO0 (EPWM1A)
67.     GpioCtrlRegs.GPAPUD.bit.GPIO1 = 0; // Enable pull-up on GPIO1 (EPWM1B)
68.     GpioCtrlRegs.GPAPUD.bit.GPIO2 = 0; // Enable pull-up on GPIO2 (EPWM2A)
69.     GpioCtrlRegs.GPAPUD.bit.GPIO3 = 0; // Enable pull-up on GPIO3 (EPWM2B)
70.     GpioCtrlRegs.GPAPUD.bit.GPIO4 = 0; // Enable pull-up on GPIO4 (EPWM3A)
71.     GpioCtrlRegs.GPAPUD.bit.GPIO5 = 0; // Enable pull-up on GPIO5 (EPWM3B)
72.     GpioCtrlRegs.GPAPUD.bit.GPIO6 = 0; // Enable pull-up on GPIO6 (EPWM4A)
73.     GpioCtrlRegs.GPAPUD.bit.GPIO7 = 0; // Enable pull-up on GPIO7 (EPWM4B)
74.
75.     GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 1; // Configure GPIO0 as EPWM1A
76.     GpioCtrlRegs.GPAMUX1.bit.GPIO1 = 1; // Configure GPIO1 as EPWM1B
77.     GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 1; // Configure GPIO2 as EPWM2A
78.     GpioCtrlRegs.GPAMUX1.bit.GPIO3 = 1; // Configure GPIO3 as EPWM2B
79.     GpioCtrlRegs.GPAMUX1.bit.GPIO4 = 1; // Configure GPIO4 as EPWM3A
80.     GpioCtrlRegs.GPAMUX1.bit.GPIO5 = 1; // Configure GPIO5 as EPWM3B
81.     GpioCtrlRegs.GPAMUX1.bit.GPIO6 = 1; // Configure GPIO6 as EPWM4A
82.     GpioCtrlRegs.GPAMUX1.bit.GPIO7 = 1; // Configure GPIO7 as EPWM4B

```

```

82.     EDIS;
83.
84.     //Inicializamos el periférico
85.     EALLOW;
86.     SysCtrlRegs.PCLKCR0.bit.TBCLKSYSNCR = 0;    // Se deshabilita timer mientras se configura PWM
87.     EDIS;
88.
89.     //Se inicializan los PWM
90.     _init_pwm(&EPwm1Regs);
91.     _init_pwm(&EPwm2Regs);
92.     _init_pwm(&EPwm3Regs);
93.     _init_pwm(&EPwm4Regs);
94.
95.     EALLOW;
96.     SysCtrlRegs.PCLKCR0.bit.TBCLKSYSNCR = 1; // Se vuelve a habilitar el timer
97.     EDIS;
98.
99. }
100.
101. void hal_pwm_ejecutar(void)
102. {
103. }
104.
105.
106. #pragma CODE_SECTION(hal_pwm_accionar, "ramfuncs");
107. void hal_pwm_accionar(MOTOR m, DIRECCION_MOTOR d, Uint16 ciclo_trabajo)
108. {
109.     /*
110.     PWM1A -> M1, ADELANTE, H |           PWM2A -> M2, ATRAS, H
111.     PWM3A -> M1, ADELANTE, L |           PWM4A -> M2, ATRAS, L
112.     PWM1B -> M1, ATRAS, H |             PWM2B -> M2, ADELANTE, H
113.     PWM3B -> M1, ATRAS, L |             PWM4B -> M2, ADELANTE, L
114.     */
115.     Uint32 duty;
116.
117.     if (ciclo_trabajo > DUTY_MAX) //Se limita el ciclo de trabajo al resto del software
118.         ciclo_trabajo = DUTY_MAX;
119.
120.     if (pwm_funcionando == FALSE)
121.     {
122.         EPwm1Regs.CMPA.half.CMPA = 0;
123.         EPwm1Regs.CMPB = 0;
124.         EPwm2Regs.CMPA.half.CMPA = 0;
125.         EPwm2Regs.CMPB = 0;
126.         EPwm3Regs.CMPA.half.CMPA = 0;
127.         EPwm3Regs.CMPB = 0;
128.         EPwm4Regs.CMPA.half.CMPA = 0;
129.         EPwm4Regs.CMPB = 0;
130.         return;
131.     }
132.
133.     //      duty = (periodo_contador/1000) * ciclo_trabajo;
134.
135.     duty = (periodo_contador/(1000- UMBRAL_DUTY)) * ciclo_trabajo + UMBRAL_DUTY * (periodo_contador/1000);
136.
137.     if (duty > duty_maximo) { duty = duty_maximo; } //Por esas cosas de la vida que no sea mayor al 30%
138.
139.     if (m == MOTOR_DERECHO)
140.     {
141.         if (d == ATRAS)
142.         {
143.             EPwm1Regs.CMPA.half.CMPA = duty;
144.             EPwm1Regs.CMPB = 0;
145.             EPwm3Regs.CMPA.half.CMPA = duty_abajo;
146.             EPwm3Regs.CMPB = 0;
147.         } else
148.         {
149.             EPwm1Regs.CMPA.half.CMPA = 0;
150.             EPwm1Regs.CMPB = duty;
151.             EPwm3Regs.CMPA.half.CMPA = 0;
152.             EPwm3Regs.CMPB = duty_abajo;
153.         }
154.     } else
155.     {
156.         if (d == ADELANTE)
157.         {
158.             EPwm2Regs.CMPA.half.CMPA = 0;
159.             EPwm2Regs.CMPB = duty;
160.             EPwm4Regs.CMPA.half.CMPA = 0;
161.             EPwm4Regs.CMPB = duty_abajo;
162.         } else
163.         {
164.             EPwm2Regs.CMPA.half.CMPA = duty;
165.             EPwm2Regs.CMPB = 0;

```



```

166.             EPwm4Regs.CMPA.half.CMPA = duty_abajo;
167.             EPwm4Regs.CMPB = 0;
168.         }
169.     }
170.
171. }
172.
173. void hal_pwm_trip(void)
174. {
175.     pwm_funcionando = FALSE;
176.     EPwm1Regs.CMPA.half.CMPA = 0;
177.     EPwm1Regs.CMPB = 0;
178.     EPwm2Regs.CMPA.half.CMPA = 0;
179.     EPwm2Regs.CMPB = 0;
180.     EPwm3Regs.CMPA.half.CMPA = 0;
181.     EPwm3Regs.CMPB = 0;
182.     EPwm4Regs.CMPA.half.CMPA = 0;
183.     EPwm4Regs.CMPB = 0;
184. }
185.
186. boolean_t hal_pwm_esta_funcionando(void)
187. {
188.     return pwm_funcionando;
189. }
190.
191. void hal_pwm_start(void)
192. {
193.     pwm_funcionando = TRUE;
194. }

```

```

1. //#####
2. //Archivo: hal_corriente.c
3. //Titulo: Módulo HAL Corriente
4. //Descripcion: Obtiene las lecturas de corriente desde los puentes H
5. //
6. //
7. //Version:
8. //2010-08-13: v1.0 - Creacion de Archivo. Rodrigo Maureira.
9. //#####
10.
11. //Librerias del sistema
12. #include "..\..\Include\header\DSP280x_Device.h" // Headers para dispositivo
13. #include "..\..\Include\header\DSP280x_DefDisp.h" // Definiciones de Dispositivo
14.
15. #include "hal_corriente.h"
16. #include "hal_timer.h"
17. #include "hal_pwm.h"
18.
19. Uint32 bufferMD[LARGO_BUFFER];
20. Uint32 bufferMI[LARGO_BUFFER];
21. char indice_buffer;
22. Uint32 sumaMD;
23. Uint32 sumaMI;
24.
25. int32 offset_MI;
26. int32 offset_MD;
27.
28. //Set points de corriente para ambos motores
29. int32 sp_corriente_MD;
30. DIRECCION_MOTOR sp_corriente_MD_dir;
31.
32. int32 sp_corriente_MI;
33. DIRECCION_MOTOR sp_corriente_MI_dir;
34.
35. int32 duty_MD; //ciclo de trabajo para motor derecho
36. int32 duty_MI;
37.
38. Uint32 MarcaTiempo;
39.
40. void _init_ecap(volatile struct ECAP_REGS *ECapxRegs)
41. {
42. // Configuración del capture
43. ECapxRegs->ECEINT.all = 0x0000; // Deshabilita todas las interrupciones
44. ECapxRegs->ECCLR.all = 0xFFFF; // Limpia todos los flags de interrupcion.
45. ECapxRegs->ECCTL1.bit.CAPLDEN = 0; // Deshabilita la carga de los registros.
46. ECapxRegs->ECCTL2.bit.TSCTRSTOP = 0; // Se asegura que el contador esté detenido.
47.
48. // Configura periférico.
49. ECapxRegs->ECCTL2.bit.CONT_ONESHOT = 0; // Captura continua.
50. ECapxRegs->ECCTL2.bit.STOP_WRAP = 3; // Reinicia contador de cap1..cap4 en cap4.
51. ECapxRegs->ECCTL1.bit.CAP1POL = 0; // Flanco descendente para cap1.
52. ECapxRegs->ECCTL1.bit.CAP2POL = 0; // Flanco descendente para cap2.
53. ECapxRegs->ECCTL1.bit.CAP3POL = 0; // Flanco descendente para cap3.
54. ECapxRegs->ECCTL1.bit.CAP4POL = 0; // Flanco descendente para cap4.
55. ECapxRegs->ECCTL1.bit.CTRRST1 = 1; // Resetea contador al ocurrir un evento en cap1.
56. ECapxRegs->ECCTL1.bit.CTRRST2 = 1; // Resetea contador al ocurrir un evento en cap2.
57. ECapxRegs->ECCTL1.bit.CTRRST3 = 1; // Resetea contador al ocurrir un evento en cap3.
58. ECapxRegs->ECCTL1.bit.CTRRST4 = 1; // Resetea contador al ocurrir un evento en cap4.
59. ECapxRegs->ECCTL2.bit.SYNCO_SEL = 0; // Sin sincronización.
60. ECapxRegs->ECCTL2.bit.SYNCO_SEL = 0; // Pass through
61. ECapxRegs->ECCTL1.bit.CAPLDEN = 0; // Deshabilita temporalmente módulos de captura.
62. ECapxRegs->ECEINT.bit.CEVT1 = 0; // Deshabilita interrupción al primer evento.
63. ECapxRegs->ECEINT.bit.CEVT2 = 0; // Deshabilita interrupción al segundo evento.
64. ECapxRegs->ECEINT.bit.CEVT3 = 0; // Deshabilita interrupción al tercer evento.
65. ECapxRegs->ECEINT.bit.CEVT4 = 0; // Deshabilita interrupción al cuarto evento.
66.
67. //Pone el módulo en funcionamiento
68. ECapxRegs->ECCTL2.bit.TSCTRSTOP = 1; // Inicia contador
69. ECapxRegs->ECCTL2.bit.REARM = 1; // Arma el módulo capture.
70. ECapxRegs->ECCTL1.bit.CAPLDEN = 1; // habilita captura.
71. }
72.
73. void Init_hal_corriente(void)
74. {
75. int i;
76.
77. //Se inicializan los buffer
78. for (i=0; i<LARGO_BUFFER; i++)
79. {
80. bufferMD[i] = 0;
81. bufferMI[i] = 0;
82. }
83.
84. indice_buffer = 0;

```

```

85.     sumaMD = 0;
86.     sumaMI = 0;
87.
88.     offset_MI = 0;
89.     offset_MD = 0;
90.
91.     sp_corriente_MD = 0;
92.     sp_corriente_MI = 0;
93.
94.     sp_corriente_MD_dir = ADELANTE;
95.     sp_corriente_MI_dir = ADELANTE;
96.
97.     duty_MD = 0;
98.     duty_MI = 0;
99.
100.    MarcaTiempo = hal_timer_get_time();
101.
102.    EALLOW;
103.
104.    //Configuración GPIO
105.    GpioCtrlRegs.GPAPUD.bit.GPIO9 = 1; // Deshabilitar Pull up en GPIO9
106.    GpioCtrlRegs.GPAQSEL1.bit.GPIO9 = 0; // Sincronizar a SYSCLKOUT
107.    GpioCtrlRegs.GPAMUX1.bit.GPIO9 = 3; // GPIO9 = ECAP3
108.
109.    GpioCtrlRegs.GPAPUD.bit.GPIO11 = 1; // Deshabilitar Pull up en GPIO11
110.    GpioCtrlRegs.GPAQSEL1.bit.GPIO11 = 0; // Sincronizar a SYSCLKOUT
111.    GpioCtrlRegs.GPAMUX1.bit.GPIO11 = 3; // GPIO11 = ECAP4
112.
113.    EDIS;
114.
115.    //Inicializa los puertos de captura de las dos mediciones
116.    _init_ecap(&ECap3Regs);
117.    _init_ecap(&ECap4Regs);
118. }
119.
120. #pragma CODE_SECTION(hal_corriente_ejecutar, "ramfuncs");
121. void hal_corriente_ejecutar(void)
122. {
123.     //ECap3Regs -> Motor 1
124.     //ECap4Regs -> Motor 2
125.
126.     static boolean_t operando = FALSE;
127.
128.     int32 error_MD = 0;
129.     static int32 i_error_MD = 0;
130.     int32 duty_final_MD = 0;
131.     DIRECCION_MOTOR direccion_final_MD = ADELANTE;
132.
133.     int32 error_MI = 0;
134.     static int32 i_error_MI = 0;
135.     int32 duty_final_MI = 0;
136.     DIRECCION_MOTOR direccion_final_MI = ADELANTE;
137.
138.     Uint32 tiempoMD[4];
139.     Uint32 tiempoMI[4];
140.     Uint32 suma_parc_MD = 0;
141.     Uint32 suma_parc_MI = 0;
142.
143.     Uint32 c_anterior_MD = 0;
144.     Uint32 c_anterior_MI = 0;
145.     int i;
146.
147.
148.     //Congela los contadores mientras se obtiene la muestra
149.     ECap3Regs.ECCTL2.bit.TSCTRSTOP = 0;
150.     ECap4Regs.ECCTL2.bit.TSCTRSTOP = 0;
151.
152.     //Tiempos Motor Derecho
153.     tiempoMD[3] = ECap3Regs.CAP4;
154.     tiempoMD[2] = ECap3Regs.CAP3;
155.     tiempoMD[1] = ECap3Regs.CAP2;
156.     tiempoMD[0] = ECap3Regs.CAP1;
157.
158.     //Tiempos Motor Izquierdo
159.     tiempoMI[3] = ECap4Regs.CAP4;
160.     tiempoMI[2] = ECap4Regs.CAP3;
161.     tiempoMI[1] = ECap4Regs.CAP2;
162.     tiempoMI[0] = ECap4Regs.CAP1;
163.
164.     //Arranca los contadores luego de tomada la muestra
165.     ECap3Regs.ECCTL2.bit.TSCTRSTOP = 1;
166.     ECap4Regs.ECCTL2.bit.TSCTRSTOP = 1;
167.
168.

```

```

169. //Se inicia el cálculo de la corriente definitiva
170. for (i=0; i<4; i++)
171. {
172.     suma_parc_MD += tiempoMD[i];
173.     suma_parc_MI += tiempoMI[i];
174. }
175.
176. //corriente las últimas 4 muestras
177. suma_parc_MD = suma_parc_MD / 4;
178. suma_parc_MI = suma_parc_MI / 4;
179.
180.
181. c_anterior_MI = sumaMI / LARGO_BUFFER;
182. c_anterior_MD = sumaMD / LARGO_BUFFER;
183.
184. if (c_anterior_MI > suma_parc_MI)
185. {
186.     if (c_anterior_MI - suma_parc_MI > LIMITE_DERIVADA) suma_parc_MI = c_anterior_MI -
LIMITE_DERIVADA;
187. } else
188. {
189.     if (suma_parc_MI > c_anterior_MI > LIMITE_DERIVADA) suma_parc_MI = c_anterior_MI +
LIMITE_DERIVADA;
190. }
191.
192. if (c_anterior_MD > suma_parc_MD)
193. {
194.     if (c_anterior_MD - suma_parc_MD > LIMITE_DERIVADA) suma_parc_MD = c_anterior_MD -
LIMITE_DERIVADA;
195. } else
196. {
197.     if (suma_parc_MD > c_anterior_MD > LIMITE_DERIVADA) suma_parc_MD = c_anterior_MD +
LIMITE_DERIVADA;
198. }
199.
200. //se agrega la muestra obtenida al buffer
201. sumaMD -= bufferMD[indice_buffer];
202. sumaMD += suma_parc_MD;
203. bufferMD[indice_buffer] = suma_parc_MD;
204.
205. sumaMI -= bufferMI[indice_buffer];
206. sumaMI += suma_parc_MI;
207. bufferMI[indice_buffer] = suma_parc_MI;
208.
209. indice_buffer++; //incrementamos el índice
210. if (indice_buffer >= LARGO_BUFFER) // y rotamos si hay que hacerlo
211. {
212.     indice_buffer = 0;
213. }
214.
215. if (operando == FALSE)
216. {
217.     if (hal_timer_diff(MarcaTiempo, hal_timer_get_time()) > TIEMPO_INICIO)
218.     {
219.         //Si ya ha estado midiendo durante un rato, se toma esa lectura como cero
220.         offset_MD = CTE_CAL_MD/(sumaMD/LARGO_BUFFER);
221.         offset_MI = CTE_CAL_MI/(sumaMI/LARGO_BUFFER);
222.         operando = TRUE;
223.     }
224. } else
225. {
226.     //Lazo de control de corriente
227.     if (hal_pwm_esta_funcionando())
228.     {
229.         if (absoluto(sp_corriente_MI) > 100)//200 //Sólo acciona si el set point es mayor a
3A
230.         {
231.             error_MI = sp_corriente_MI - hal_corriente_get(MOTOR_IZQUIERDO);
232.
233.             i_error_MI += error_MI;
234.
235.             if (error_MI > 250) error_MI = 250;
236.             else if (error_MI < -250) error_MI = -250;
237.
238.
239.             duty_MI += error_MI/1;// + i_error_MI/20000; // dt = 0,0005s = 1/2000 s
240.
241.             if (duty_MI < -40000) duty_MI = -40000; // DUTY_MAX*100;
242.             if (duty_MI > 40000) duty_MI = 40000;
243.
244.             if (duty_MI < 0)
245.             {
246.                 direccion_final_MI = ATRAS;
247.                 duty_final_MI = -1*duty_MI;

```

```

248.         } else {
249.             direccion_final_MI = ADELANTE;
250.             duty_final_MI = duty_MI;
251.         }
252.
253.         hal_pwm_accionar(MOTOR_IZQUIERDO, direccion_final_MI, duty_final_MI/100);
254.     } else hal_pwm_accionar(MOTOR_IZQUIERDO, direccion_final_MI, 0);
255.
256.     if (absoluto(sp_corriente_MD) > 100)
257.     {
258.         error_MD = sp_corriente_MD - hal_corriente_get(MOTOR_DERECHO);
259.
260.         i_error_MD += error_MD;
261.
262.
263.         if (error_MD > 250) error_MD = 250;
264.         else if (error_MD < -250) error_MD = -250;
265.
266.         duty_MD += error_MD/1;// + i_error_MD/20000; // dt = 0,0005s = 1/2000 s
267.
268.         if (duty_MD < -40000) duty_MD = -40000; // DUTY_MAX*100;
269.         if (duty_MD > 40000) duty_MD = 40000;
270.
271.         if (duty_MD < 0)
272.         {
273.             direccion_final_MD = ADELANTE;
274.             duty_final_MD = -1*duty_MD;
275.         } else {
276.             direccion_final_MD = ATRAS;
277.             duty_final_MD = duty_MD;
278.         }
279.
280.         hal_pwm_accionar(MOTOR_DERECHO, direccion_final_MD, duty_final_MD/100);
281.
282.     } else hal_pwm_accionar(MOTOR_DERECHO, direccion_final_MD, 0);
283.     } else
284.     {
285.         duty_MD = 0;
286.         duty_MI = 0;
287.     }
288.
289.     }
290.
291. }
292.
293.
294. #pragma CODE_SECTION(hal_corriente_set_ref, "ramfuncs");
295. void hal_corriente_set_ref(MOTOR m, DIRECCION_MOTOR direccion, int32 ref)
296. {
297.     Uint32 ref_final = ref;
298.
299.     if (ref > 2000) //Casi al tope de lo que los motores aguantan 2250
300.     {
301.         ref_final = 2*ref - 2000; //Aumenta la respuesta del controlador
302.     }
303.
304.     if (ref_final > 2700) //Limita la referencia a lo que el controlador puede medir 2700
305.     {
306.         ref_final = 2700;
307.     }
308.
309.     if (m==MOTOR_DERECHO)
310.     {
311.         sp_corriente_MD = ref_final;
312.         if (direccion == ATRAS) sp_corriente_MD *= -1;
313.
314.         sp_corriente_MD_dir = direccion;
315.     } else
316.     {
317.         sp_corriente_MI = ref_final;
318.         if (direccion == ATRAS) sp_corriente_MI *= -1;
319.
320.         sp_corriente_MI_dir = direccion;
321.     }
322. }
323.
324. #pragma CODE_SECTION(hal_corriente_get_duty, "ramfuncs");
325. int32 hal_corriente_get_duty(MOTOR m)
326. {
327.     if (m==MOTOR_DERECHO)
328.     {
329.         return duty_MD;
330.     } else
331.     {

```

```
332.         return sp_corriente_MD;// duty_MI;
333.     }
334. }
335.
336. #pragma CODE_SECTION(hal_corriente_get, "ramfuncs");
337. int32 hal_corriente_get(MOTOR m)
338. {
339.     if (m==MOTOR_DERECHO)
340.     {
341.         return (offset_MD - (CTE_CAL_MD/(sumaMD/LARGO_BUFFER)));
342.     } else
343.     {
344.         return CTE_CAL_MI/(sumaMI/LARGO_BUFFER) - offset_MI;
345.     }
346. }
```

```

1. //#####
2. //Archivo: mod_comando.c
3. //Titulo: Módulo de comandos
4. //Descripcion: Módulo dedicado a interpretar y ejecutar los comandos enviados
5. //      por serial.
6. //
7. //Version:
8. //2010-07-29: v1.0 -
9. //#####
10.
11. //Librerias del sistema
12. #include "..\Include\header\DSP280x_Device.h" // Headers para dispositivo
13. #include "..\Include\header\DSP280x_DefDisp.h" // Definiciones de Dispositivo
14.
15. #include "hal\hal_timer.h"
16. #include "hal\hal_serial.h"
17. #include "hal\hal_instrumentos.h"
18. #include "hal\hal_pwm.h"
19. #include "hal\hal_corriente.h"
20. #include "hal\hal_adc.h"
21. #include "mod_control.h"
22.
23. //Los 2 bits más significativos del comando definen su función;
24. #define MASCARA_TIPO 0x00C0 //rescata los dos bits mas significativos
25. #define C_DESCONOCIDO 0x0000 //00 -> desconocido
26. #define C_ESCRIBIR 0x0040 //01 -> escribir
27. #define C_LEER 0x0080 //10 -> leer
28. #define C_ORDEN 0x00C0 //11 -> orden
29.
30. //Nombres de codigo de comando
31. #define C_PING_DSP 192
32. #define C_TRIP_MOTOR 193
33. #define C_UNLOCK_MOTOR 194
34. #define C_FLJAR_INCLINACION_NULA 196
35. #define C_DETENER_FLUJOS 197
36.
37. #define C_LEER_INCLINOMETROS 129
38. #define C_LEER_GIROSCOPOS 130
39. #define C_LEER_CORRIENTE_D 131
40. #define C_LEER_CORRIENTE_I 132
41. #define C_FLUJO_INCLINOMETROS 133
42. #define C_FLUJO_GIROSCOPOS 134
43. #define C_FLUJO_CORRIENTE_MD 135
44. #define C_FLUJO_CORRIENTE_MI 136
45. #define C_FLUJO_IMU 137
46. #define C_FLUJO_ADC 138 // 138 para dirección, 139 para batería, 140 para presión
47. #define C_FLUJO_CONTROL 141
48.
49. #define C_SET_MD_F_DUTY 65
50. #define C_SET_MD_B_DUTY 66
51. #define C_SET_MI_F_DUTY 67
52. #define C_SET_MI_B_DUTY 68
53. #define C_SET_CTRL_KP 69
54. #define C_SET_CTRL_KD 70
55.
56.
57. //--Prototipos de funciones internas--
58. //Funciones de control de flujo
59. void capturar_comando(void);
60. void interpretar_comando(void);
61. void devolver_comando(void);
62. void ejecutar_orden(void);
63. void enviar_respuesta(void);
64. void leer_parametro(void);
65. void escribir_parametro(void);
66.
67. //Ejecución de comandos
68. void leer_inclinometros(void);
69. void leer_giroscofos(void);
70. void leer_corriente(MOTOR m);
71. void enviar_flujo(char orden);
72.
73. COMANDO c_activo; //Comando en proceso
74. char en_ejecucion; //Define las etapas de ejecucion
75. boolean_t orden_inicio; //Indica si se ha solicitado el encendido del vehiculo
76.
77. void mod_comando_inicializar(void)
78. {
79.     c_activo.Codigo = 0;
80.     c_activo.ID = 0;
81.     c_activo.n_elemento = 0;
82.     c_activo.valor_ent = 0;
83.     en_ejecucion = 0;
84.     orden_inicio = FALSE;

```

```

85. }
86.
87.
88. void mod_comando_ejecutar(void)
89. {
90.     switch (en_ejecucion) {
91.         case 0: capturar_comando();
92.             break;
93.
94.         case 1: interpretar_comando();
95.             break;
96.
97.         case 2: devolver_comando();
98.             break;
99.
100.        case 3: ejecutar_orden();
101.            break;
102.
103.        case 4: leer_parametro();
104.            break;
105.
106.        case 5: escribir_parametro();
107.            break;
108.
109.        case 6: enviar_respuesta();
110.            break;
111.
112.        case 7: enviar_flujo(0);
113.            break;
114.
115.        default: en_ejecucion = 0;
116.            break;
117.    }
118. }
119.
120. void capturar_comando(void)
121. {
122.     if (hal_serial_hay_comando() == TRUE)
123.     {
124.         c_activo = hal_serial_get_comando();
125.         en_ejecucion = 1;
126.     }
127. }
128.
129. void interpretar_comando(void)
130. {
131.     Uint16 tipo;
132.
133.     tipo = MASCARA_TIPO & c_activo.Codigo;
134.
135.     switch (tipo){
136.         case C_ESCRIBIR    : escribir_parametro();
137.                             break;
138.
139.         case C_LEER       : leer_parametro();
140.                             break;
141.
142.         case C_ORDEN      : ejecutar_orden();
143.                             break;
144.
145.         case C_DESCONOCIDO: devolver_comando(); //Devuelve un error
146.                             break;
147.
148.         default          : devolver_comando(); //Devuelve un error
149.                             break;
150.     }
151. }
152.
153. void devolver_comando(void)
154. {
155.     en_ejecucion = 2;
156.     if (hal_serial_puede_enviar() == TRUE) //Si ya se puede, retorna el comando de error
157.     {
158.         c_activo.valor_ent = 0xAAAAAAAA; //Marca para distinguir un comando de error
159.         hal_serial_enviar_comando(c_activo);
160.
161.         en_ejecucion = 0; //se deshace de todo y queda listo para nuevos comandos
162.     }
163. }
164.
165. void enviar_respuesta(void)
166. {
167.     en_ejecucion = 6;
168.

```



```

169.   if (hal_serial_puede_enviar() == TRUE)
170.   {
171.       hal_serial_enviar_comando(c_activo);
172.       en_ejecucion = 0;
173.   }
174. }
175.
176. void leer_parametro(void)
177. {
178.     en_ejecucion = 4;
179.
180.     switch (c_activo.Codigo){
181.         case C_LEER_INCLINOMETROS : leer_inclinometros();
182.                                     break;
183.
184.         case C_LEER_GIROSCOPOS      : leer_giroscofos();
185.                                     break;
186.
187.         case C_LEER_CORRIENTE_D    : leer_corriente(MOTOR_DERECHO);
188.                                     break;
189.
190.         case C_LEER_CORRIENTE_I    : leer_corriente(MOTOR_IZQUIERDO);
191.                                     break;
192.
193.         case C_FLUJO_INCLINOMETROS: enviar_flujo(C_FLUJO_INCLINOMETROS);
194.                                     break;
195.
196.         case C_FLUJO_GIROSCOPOS    : enviar_flujo(C_FLUJO_GIROSCOPOS);
197.                                     break;
198.
199.         case C_FLUJO_CORRIENTE_MD  : enviar_flujo(C_FLUJO_CORRIENTE_MD);
200.                                     break;
201.
202.         case C_FLUJO_CORRIENTE_MI  : enviar_flujo(C_FLUJO_CORRIENTE_MI);
203.                                     break;
204.
205.         case C_FLUJO_IMU           : enviar_flujo(C_FLUJO_IMU);
206.                                     break;
207.
208.         case C_FLUJO_ADC           : enviar_flujo(C_FLUJO_ADC);
209.                                     break;
210.
211.         case C_FLUJO_CONTROL       : enviar_flujo(C_FLUJO_CONTROL);
212.                                     break;
213.
214.         default : devolver_comando();
215.                 break;
216.     }
217. }
218.
219. void enviar_flujo(char orden)
220. {
221.     //0: Inclinometro  1: Giroscopo  2:Corriente MD 3: Corriente MI  4:IMU  5:ADC
222.
223.     static char instrumento = 0;
224.     static char estado_lect = 0; //guarda el estado de lectura para flujos de más de un dato
225.     int32 lectura;
226.
227.     if (orden != 0) //Si se llama por primera vez
228.     {
229.         en_ejecucion = 7; //Hace que se quede en esta función
230.         switch (orden){
231.             case C_FLUJO_INCLINOMETROS: instrumento = 0;
232.                                     lectura =
233.
234.             hal_instrumentos_get_inclinacion();
235.                                     break;
236.
237.             case C_FLUJO_GIROSCOPOS : instrumento = 1;
238.                                     lectura =
239.
240.             hal_instrumentos_get_vel_giro();
241.                                     break;
242.
243.             case C_FLUJO_CORRIENTE_MD: instrumento = 2;
244.                                     lectura =
245.
246.             hal_corriente_get_duty(MOTOR_DERECHO);
247.                                     break;
248.
249.             case C_FLUJO_CORRIENTE_MI: instrumento = 3;
250.                                     lectura =
251.
252.             hal_corriente_get_duty(MOTOR_IZQUIERDO);
253.                                     break;
254.
255.             case C_FLUJO_IMU           : instrumento = 4;
256.                                     lectura =
257.
258.             hal_imu_get_vel_giro();
259.                                     break;
260.
261.             case C_FLUJO_ADC           : instrumento = 5;
262.                                     lectura =
263.
264.             hal_adc_get_valor();
265.                                     break;
266.
267.             default :
268.                 return;
269.         }
270.     }
271.     estado_lect = 1;
272. }

```

```

249.                                     lectura =
hal_instrumentos_get_inclinacion();
250.                                     estado_lect = 1;
251.                                     break;
252.
253.             case C_FLUJO_ADC           : instrumento = 5;
254.                                     lectura =
hal_adc_get_direccion();
255.                                     estado_lect = 1;
256.                                     break;
257.
258.             case C_FLUJO_CONTROL       : instrumento = 6;
259.                                     lectura =
mod_control_get_torque();
260.                                     estado_lect = 1;
261.                                     break;
262.
263.             default : devolver_comando();
264.                     return;
265.     }
266.
267.     c_activo.valor_ent = 2000000000 + lectura; //sumamos 2000000000 para que los enteros queden sin
signo
268.     enviar_respuesta(); //envía la respuesta según conducto regular y permite esperar el desbloqueo
269.     en_ejecucion = 7; //Insiste en que se quede en esta función
270.     return;
271. }
272.
273. //Si la orden es 0 estamos en régimen permanente
274. switch (instrumento){
275.     case 0: lectura = hal_instrumentos_get_inclinacion();
276.             break;
277.
278.     case 1: lectura = hal_instrumentos_get_vel_giro();
279.             break;
280.
281.     case 2: lectura = hal_corriente_get(MOTOR_DERECHO);
282.             break;
283.
284.     case 3: lectura = hal_corriente_get(MOTOR_IZQUIERDO);
285.             break;
286.
287.     case 4: if (estado_lect == 1)
288.             {
289.                 lectura = hal_instrumentos_get_vel_giro();
290.                 c_activo.Codigo = C_FLUJO_GIROSCOPOS;
291.                 estado_lect = 0;
292.             } else {
293.                 lectura = hal_instrumentos_get_inclinacion();
294.                 c_activo.Codigo = C_FLUJO_INCLINOMETROS;
295.                 estado_lect = 1;
296.             }
297.             break;
298.
299.     case 5: if (estado_lect == 1)
300.             {
301.                 lectura = hal_adc_get_direccion();
302.                 c_activo.Codigo = C_FLUJO_ADC;
303.                 estado_lect = 0;
304.             } else {
305.                 lectura = hal_adc_get_bateria();
306.                 c_activo.Codigo = C_FLUJO_ADC + 1;
307.                 estado_lect = 1;
308.             }
309.             break;
310.
311.     case 6: if (estado_lect == 1)
312.             {
313.                 lectura = hal_corriente_get_duty(MOTOR_IZQUIERDO);
314.                 c_activo.Codigo = C_FLUJO_CORRIENTE_MI;
315.                 estado_lect = 0;
316.             } else {
317.                 lectura = hal_corriente_get_duty(MOTOR_DERECHO);
318.                 c_activo.Codigo = C_FLUJO_CORRIENTE_MD;
319.                 estado_lect = 1;
320.             }
321.             break;
322.
323.     default: lectura = 0;
324.             en_ejecucion = 0; //error, nos salimos del flujo
325.             break;
326. }
327.

```

```

328.     c_activo.n_elemento = 255; //Marcamos el comando para que no intente vaciar la cola de recepción
        serial
329.     c_activo.valor_ent = 2000000000 + lectura; //sumamos 2000000000 para que los enteros queden sin signo
330.     c_activo.ID += 1; //Sumamos 1 al ID para que el cliente sepa si se pierden paquetes del flujo
331.
332.     if (hal_serial_puede_enviar() == TRUE) //Se envía el elemento del flujo
333.     {
334.         hal_serial_enviar_comando(c_activo);
335.     }
336.     capturar_comando(); //Y espera por comandos, cualquier cosa detiene el flujo
337. }
338.
339. void escribir_parametro(void)
340. {
341.     en_ejecucion = 5;
342.
343.     switch (c_activo.Codigo){
344.         case C_SET_MD_F_DUTY: hal_corriente_set_ref(MOTOR_DERECHO, ADELANTE, c_activo.valor_ent);
345.                                 enviar_respuesta();
346.                                 break;
347.
348.         case C_SET_MD_B_DUTY: hal_corriente_set_ref(MOTOR_DERECHO, ATRAS, c_activo.valor_ent);
349.                                 enviar_respuesta();
350.                                 break;
351.
352.         case C_SET_MI_F_DUTY: hal_corriente_set_ref(MOTOR_IZQUIERDO, ADELANTE, c_activo.valor_ent);
353.                                 enviar_respuesta();
354.                                 break;
355.
356.         case C_SET_MI_B_DUTY: hal_corriente_set_ref(MOTOR_IZQUIERDO, ATRAS, c_activo.valor_ent);
357.                                 enviar_respuesta();
358.                                 break;
359.
360.         case C_SET_CTRL_KP : hal_pwm_trip();
361.                                 mod_control_set_var(KP, c_activo.valor_ent/1.0);
362.                                 enviar_respuesta();
363.                                 break;
364.
365.         case C_SET_CTRL_KD : hal_pwm_trip();
366.                                 mod_control_set_var(KD, c_activo.valor_ent/1.0);
367.                                 enviar_respuesta();
368.                                 break;
369.
370.         default : devolver_comando();
371.                 break;
372.     }
373. }
374.
375. void ejecutar_orden(void)
376. {
377.     en_ejecucion = 3;
378.     switch (c_activo.Codigo){
379.         case C_FIJAR_INCLINACION_NULA : hal_instrumentos_set_horizontal(c_activo.valor_ent);
380.                                         c_activo.valor_ent = 1;
381.                                         enviar_respuesta();
382.                                         //Redirige al envío de la respuesta
383.                                         break;
384.
385.         case C_PING_DSP : enviar_respuesta(); //devuelve el comando enviado
386.                                 break;
387.
388.         case C_TRIP_MOTOR: hal_pwm_trip(); //Detengo inmediatamente los motores
389.                                 orden_inicio = FALSE; //Aviso que apagué al módulo
390.                                 principal
391.                                 c_activo.valor_ent = 1; //indicar que se realizo la accion
392.                                 enviar_respuesta(); //Redirige al envío de la respuesta
393.                                 break;
394.
395.         case C_UNLOCK_MOTOR: orden_inicio = TRUE; //Solicita el encendido de los motores
396.                                 accion
397.                                 c_activo.valor_ent = 1; //indicar que se realizo la
398.                                 enviar_respuesta(); //Redirige al envío de la respuesta
399.                                 break;
400.
401.         default : devolver_comando();
402.                 break;
403.     }
404. }
405. void leer_corriente(MOTOR m)
406. {

```

```

407.     Uint32 f = 0;
408.     f = hal_corriente_get(m);
409.     c_activo.valor_ent = f;
410.     enviar_respuesta();
411. }
412.
413. void leer_inclinometros(void)
414. {
415.     int32 inclinacion;
416.     inclinacion = hal_instrumentos_get_inclinacion();
417.     c_activo.valor_ent = inclinacion;
418.
419.     enviar_respuesta();
420. }
421.
422. void leer_giros copos(void)
423. {
424.     int32 velocidad_giro;
425.     velocidad_giro = hal_instrumentos_get_vel_giro();
426.     c_activo.valor_ent = velocidad_giro;
427.
428.     enviar_respuesta();
429. }
430.
431. boolean_t get_orden_inicio(void)
432. {
433.     return orden_inicio;
434. }
435.
436. void set_orden_inicio(boolean_t valor)
437. {
438.     orden_inicio = valor;
439. }

```

```

1. //#####
2. //Archivo: mod_control.c
3. //Titulo: Módulo de control
4. //Descripcion: Módulo dedicado al control del vehículo
5. //
6. //Version:
7. //2010-08-26: v1.0 -
8. //#####
9.
10. //Librerías del sistema
11. #include "..\Include\header\DSP280x_Device.h" // Headers para dispositivo
12. #include "..\Include\header\DSP280x_DefDisp.h" // Definiciones de Dispositivo
13.
14. #include "hal\hal_instrumentos.h"
15. #include "hal\hal_pwm.h"
16. #include "hal\hal_corriente.h"
17. #include "hal\hal_gpio.h"
18. #include "hal\hal_adc.h"
19. #include "mod_control.h"
20. #include "mod_comando.h"
21.
22. #define PI 3.141592
23.
24. int32 torque_final;
25. int32 dir_final;
26. float kp;
27. float kd;
28. float kp_s;
29. float kd_s;
30.
31. void accionar_motores(int32 torque, int giro);
32. float seno(float x);
33. float coseno(float x);
34.
35. void mod_control_inicializar(void)
36. {
37.     torque_final = 0;
38.     dir_final = 0;
39. }
40.
41. #pragma CODE_SECTION(seno, "ramfuncs");
42. float seno(float x)
43. {
44.     return x - x*x*x/6.0;
45. }
46.
47. #pragma CODE_SECTION(coseno, "ramfuncs");
48. float coseno(float x)
49. {
50.     return 1.0 - x*x/2.0;
51. }
52.
53. #pragma CODE_SECTION(mod_control_ejecutar, "ramfuncs");
54. void mod_control_ejecutar(void)
55. {
56.     int32 inclinacion;
57.     int32 velocidad_giro;
58.     int32 incl_proyectada;
59.     int32 dir_temp;
60.     // float incl_rad;
61.
62.     /* Obtención de datos para el control */
63.     inclinacion = hal_instrumentos_get_inclinacion();
64.     velocidad_giro = hal_instrumentos_get_vel_giro();
65.
66.     /* Obtención de lectura del mando de dirección */
67.     dir_temp = hal_adc_get_direccion();
68.
69.     if (absoluto(dir_temp) < Z_MUERTA_DIR)
70.     {
71.         dir_temp = 0;
72.     } else {
73.         if (dir_temp > 0)
74.         {
75.             dir_temp -= Z_MUERTA_DIR;
76.         } else {
77.             dir_temp += Z_MUERTA_DIR;
78.         }
79.     }
80.
81.     //Proyección de la inclinación a 220 ms para compensar retardo de inclinómetro
82.     incl_proyectada = inclinacion + velocidad_giro*(22/100); //22 proyecciones por dt = 1/100
83.
84.     // incl_rad = PI*(incl_proyectada/UNO)/180.0;

```

```

85.
86.     if (hal_adc_get_presion() > 2000) //Si hay alguien arriba, aplico unos parámetros 2100
87.     {
88.         torque_final = kp*(incl_proyectada/UNO) + kd*(velocidad_giro/UNO);
89.         //      duty_final = kp*seno(incl_rad) + kd*(velocidad_giro/UNO)*coseno(incl_rad);
90.
91.         //Compensación velocidad: más velocidad, menos fuerte el giro
92.         dir_final = dir_temp * (3.2 - absoluto(torque_final)/1500.0);
93.
94.         accionar_motores(torque_final, dir_final);
95.
96.     } else {
97.         //Si no hay nadie, ocupo estos otros parámetros
98.         torque_final = kp_s*(incl_proyectada/UNO) + kd_s*(velocidad_giro/UNO);
99.
100.        //Compensación velocidad: más velocidad, menos fuerte el giro
101.        dir_final = dir_temp * (1.5 - absoluto(torque_final)/6000.0);
102.
103.        accionar_motores(torque_final, dir_final);
104.    }
105.
106.    if (hal_pwm_esta_funcionando())
107.    {
108.        if ((inclinacion > 2000000) || (inclinacion < -2000000))
109.        {
110.            hal_gpio_emitir_tono(D_TONO_ERROR, FALSE);
111.            hal_pwm_trip();
112.            set_orden_inicio(FALSE);
113.        }
114.    }
115. }
116.
117. int32 mod_control_get_torque()
118. {
119.     return torque_final;
120. }
121.
122. void mod_control_set_var(VARIABLE_CONTROL v, float valor)
123. {
124.     switch(v){
125.         case KP: kp = valor;
126.             break;
127.
128.         case KD: kd = valor;
129.             break;
130.
131.         case KP_S: kp_s = valor;
132.             break;
133.
134.         case KD_S: kd_s = valor;
135.             break;
136.     }
137. }
138.
139. void accionar_motores(int32 torque, int giro)
140. {
141.
142.     int ref_izquierdo = 0;
143.     int ref_derecho = 0;
144.
145.     if (torque < 0)
146.     {
147.         torque = torque * -1;
148.         ref_izquierdo = torque - giro;
149.         ref_derecho = torque + giro;
150.
151.         if (ref_izquierdo < 0) hal_corriente_set_ref(MOTOR_IZQUIERDO, ATRAS, -1.1 * ref_izquierdo);
152.         else hal_corriente_set_ref(MOTOR_IZQUIERDO, ADELANTE, ref_izquierdo);
153.
154.         if (ref_derecho < 0) hal_corriente_set_ref(MOTOR_DERECHO, ATRAS, -1 * ref_derecho);
155.         else hal_corriente_set_ref(MOTOR_DERECHO, ADELANTE, ref_derecho * 1.1);
156.
157.     } else
158.     {
159.         ref_izquierdo = torque + giro;
160.         ref_derecho = torque - giro;
161.
162.         if (ref_izquierdo < 0) hal_corriente_set_ref(MOTOR_IZQUIERDO, ADELANTE, -1 * ref_izquierdo);
163.         else hal_corriente_set_ref(MOTOR_IZQUIERDO, ATRAS, 1.1 * ref_izquierdo);
164.
165.         if (ref_derecho < 0) hal_corriente_set_ref(MOTOR_DERECHO, ADELANTE, -1.1 * ref_derecho);
166.         else hal_corriente_set_ref(MOTOR_DERECHO, ATRAS, ref_derecho * 1);
167.     }
168. }

```

SOFTWARE DEL CONTROLADOR DE DISPLAY LCD

```
1. #include <l8F252.h>
2. #device ADC=10
3.
4. #fuses HS,NOWDT,PUT,NOLVP
5. #use delay(clock=16000000)
6. #include "flex_lcd.c"
7. #include <stdlib.h>
8.
9. #use rs232(baud=19200, xmit=PIN_C6, rcv=PIN_C7, stream = DSP)
10.
11. /* Pines para Leds */
12. #define LED_1 PIN_A0
13. #define LED_2 PIN_A2
14.
15. /* Valor ASCII para el signo de grado y de porcentaje*/
16. #define ASCII_GRADO 223
17. #define ASCII_PORC 37
18.
19. typedef struct Comando {
20.     int8 Codigo;
21.     int8 ID;
22.     int32 valor_ent;
23. }COMANDO;
24.
25.
26. // VARIABLES EN RAM ////////////////////////////////////////
27. int8 ibuff = 0; // índice: siguiente char en cbuffer
28. char cbuffer[7]; // Buffer
29. char rcvchar = 0x00; // último carácter recibido
30. int1 flag_c_recibido = 0; // Flag para indicar comando disponible
31. int1 flag_aux = 0; //Flag relativo al rs232
32. int1 flag_timer = 0; //Flag para timer
33. int1 flag_boton_1 = 0; //Flag que indica pulsacion de boton 1
34. int1 flag_boton_2 = 0; //Flag que indica pulsacion de boton 2
35. char c_pendiente[6]; //Comando pendiente para ser enviado
36. int1 flag_c_pendiente = 0; //Indica comando pendiente de enviar
37. int8 ID_comando = 1; //ID de comando a enviar
38. COMANDO c_recibido; //Comando recibido
39.
40. // Declaración de Funciones ////////////////////////////////////////
41.
42. void manejar_serial(void); //Manejo puerto serial
43. void enviar_comando(COMANDO c);
44.
45. // INTERRUPTACIONES ////////////////////////////////////////
46. #int_rda
47. void serial_isr() { // Interrupción recepción serie USART
48.
49.     rcvchar=0x00; // Inicializo carácter recibido
50.     if(kbhit()){ // Si hay algo pendiente de recibir ...
51.         rcvchar=getc(); // lo descargo y ...
52.         if (ibuff < 7){
53.             cbuffer[ibuff] = rcvchar;
54.             ibuff++;
55.         }
56.     }
57. }
58.
59. /* Interrupción del Timer0, cada 262,144 ms */
60. /* Actualiza la información del LCD y envía */
61. /* datos del encoder al DSP. */
62. #int_RTCC
63. void RTCC_isr(){
64.     flag_timer = 1;
65. }
66.
67. /* Interrupción botón 1 (izquierda)*/
68. #int_ext1
69. void ext1_isr(){
70.     flag_boton_1 = 1;
71. }
72.
73. /* Interrupción botón 2 (derecha)*/
74. #int_ext
75. void ext_isr(){
76.     flag_boton_2 = 1;
77. }
78.
79. void manejar_serial(void){
80.     int32 mascara = 0x000000FF;
81.     int8 i;
82.
```

```

83.     if(ibuff > 5){ // Si ha llegado algo desde el DSP
84.         c_recibido.Codigo = cbuff[0];
85.         c_recibido.ID = cbuff[1];
86.
87.         c_recibido.valor_ent = cbuff[2];
88.         c_recibido.valor_ent = (c_recibido.valor_ent << 8) + cbuff[3];
89.         c_recibido.valor_ent = (c_recibido.valor_ent << 8) + cbuff[4];
90.         c_recibido.valor_ent = (c_recibido.valor_ent << 8) + cbuff[5];
91.
92.         ibuff = 0;
93.         flag_c_recibido = 1;
94.     }
95. }
96.
97. void enviar_comando(COMANDO c){
98.     int8 i;
99.     int32 mascara = 0x000000FF;
100.
101.     ID_comando++;
102.
103.     c_pendiente[0] = c.Codigo;
104.     c_pendiente[1] = ID_comando;
105.
106.     c_pendiente[2] = mascara & (c.valor_ent >> 24);
107.     c_pendiente[3] = mascara & (c.valor_ent >> 16);
108.     c_pendiente[4] = mascara & (c.valor_ent >> 8);
109.     c_pendiente[5] = mascara & c.valor_ent;
110.
111.     for (i=0; i<7; i++)
112.     {
113.         putchar(c_pendiente[i]);
114.         delay_ms(5);
115.     }
116.
117. }
118.
119. //=====
120. void main(){
121.
122.     int1 iniciado = 0;
123.     int16 time_out = 0;
124.     COMANDO c;
125.
126.     delay_ms(500);
127.
128.     setup_counters(RTCC_INTERNAL, RTCC_DIV_8); // TIMER0: Clock Interno 16 bits, Prescaler 16 //8 //64
129.                                             // para una RTCC cada 65,536 milisegundos
130.
131.     //setup_timer_0(RTCC_INTERNAL | RTCC_DIV_256 | RTCC_8_BIT);
132.     setup_timer_1(T1_DISABLED);
133.     setup_timer_2(T2_DISABLED,0,1);
134.
135.     enable_interrupts(INT_RTCC); // Habilito Interrupción RTCC
136.     enable_interrupts(INT_EXT1); // Interrupción para Botón 1
137.     enable_interrupts(INT_EXT); // Interrupción para Botón 2
138.     enable_interrupts(int_rda); // Interrupción serial
139.     enable_interrupts(global); // Habilito Interrupciones
140.
141.     ext_int_edge(1, L_TO_H); /* Botones se activan al soltar */
142.     ext_int_edge(0, L_TO_H);
143.
144.     port_b_pullups(TRUE);
145.
146.     /* Enciende LED's para debugging */
147.     output_high(LED_1);
148.     delay_ms(100);
149.     output_high(LED_2);
150.     delay_ms(100);
151.     output_low(LED_1);
152.     delay_ms(100);
153.     output_low(LED_2);
154.
155.     lcd_init(); // Inicializa LCD
156.     lcd_gotoxy(1,1); lcd_putc(" Vehiculo ");
157.     lcd_gotoxy(1,2); lcd_putc(" Autobalanceado ");
158.     lcd_gotoxy(20,1);
159.     delay_ms(2000);
160.     lcd_gotoxy(1,1); lcd_putc("Iniciando ");
161.     lcd_gotoxy(1,2); lcd_putc("Sistemas... ");
162.     lcd_gotoxy(20,1);
163.     delay_ms(1000);
164.     flag_boton_1 = 0;
165.     flag_boton_2 = 0;
166.

```



```

167. while(1){
168.     manejar_serial();
169.
170.     if (iniciado == 0){
171.         if (time_out == 0)
172.             {
173.                 lcd_gotoxy(1,1); lcd_putc("Probando      ");
174.                 lcd_gotoxy(1,2); lcd_putc("Comunicaciones..");
175.                 lcd_gotoxy(20,1);
176.                 delay_ms(1000);
177.                 c.Codigo = 192;
178.                 c.ID = 1;
179.                 c.valor_ent = 1;
180.                 enviar_comando(c); //Envía un PING al DSP
181.             }
182.             time_out++;
183.             delay_ms(50);
184.             if (flag_c_recibido) {
185.                 if (c_recibido.Codigo == 192){
186.                     lcd_gotoxy(1,1); lcd_putc("Conexion      ");
187.                     lcd_gotoxy(1,2); lcd_putc("Establecida  ");
188.                     lcd_gotoxy(20,1);
189.                     delay_ms(1000);
190.                     iniciado = 1;
191.                 }
192.             } else {
193.                 if (time_out > 100) {
194.                     lcd_gotoxy(1,1); lcd_putc("ERROR:      ");
195.                     lcd_gotoxy(1,2); lcd_putc("DSP NO RESPONDE ");
196.                     lcd_gotoxy(20,1);
197.                     time_out = 101;
198.                     delay_ms(1000);
199.                 }
200.             }
201.         } else {
202.             //Acá ya está establecida la comunicación
203.             /* Timer cada 65,536 ms */
204.             if(flag_timer){
205.                 flag_timer = 0;
206.
207.                 if(flag_boton_1){
208.                     lcd_gotoxy(1,1); lcd_putc("FUNCIONANDO  ");
209.                     lcd_gotoxy(1,2); lcd_putc("              ");
210.                     lcd_gotoxy(20,1);
211.
212.                     c.Codigo = 194;
213.                     c.ID = 1;
214.                     c.valor_ent = 1;
215.                     enviar_comando(c); //Envía la orden al DSP
216.
217.                     delay_ms(500);
218.                     flag_boton_1 = 0;
219.                 }
220.                 if(flag_boton_2){
221.                     lcd_gotoxy(1,1); lcd_putc("DETENIDO     ");
222.                     lcd_gotoxy(1,2); lcd_putc("             ");
223.                     lcd_gotoxy(20,1);
224.
225.                     c.Codigo = 193;
226.                     c.ID = 1;
227.                     c.valor_ent = 1;
228.                     enviar_comando(c); //Envía la orden al DSP
229.
230.                     delay_ms(500);
231.                     flag_boton_2 = 0;
232.                 }
233.             }
234.         }
235.     }
236. }

```

SOFTWARE DE MONITOREO

```
1.  unit Unidad1;
2.
3.  interface
4.
5.  uses
6.    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
7.    Dialogs, CPDrv, StdCtrls, MPlayer, ExtCtrls, ComCtrls, A3nalogGauge, LCDLine;
8.
9.  type
10.   TComando = Record
11.     Codigo: Byte;
12.     ID: Byte;
13.     Valor: Single;
14.     Valor_int: Integer;
15.   end;
16.
17.   TBuffComandos = class(TObject)
18.   private
19.     FCount: Integer;
20.     FBuffer: array of TComando;
21.   protected
22.     procedure setElemento(Index: Integer; const Value: TComando);
23.     function getElemento(Index: Integer): TComando;
24.   public
25.     constructor Create;
26.     destructor Destroy; override;
27.     procedure Add(iElemento: TComando);
28.     procedure Delete(Index: Integer);
29.     procedure Insertar(Index: Integer; iElemento: TComando);
30.     procedure Clear;
31.     procedure Asignar(Elementos: TBuffComandos);
32.     function getCopia: TBuffComandos;
33.     property Elemento[I: Integer]:TComando read getElemento write setElemento;
34.     property Count: Integer read FCount;
35.   end;
36.
37.   TBuffDatos = class(TObject)
38.   private
39.     FCount: Integer;
40.     FBuffer: array of Byte;
41.   protected
42.     procedure setElemento(Index: Integer; const Value: Byte);
43.     function getElemento(Index: Integer): Byte;
44.   public
45.     constructor Create;
46.     destructor Destroy; override;
47.     procedure Add(iElemento: Byte);
48.     procedure Delete(Index: Integer);
49.     procedure Insertar(Index: Integer; iElemento: Byte);
50.     procedure Clear;
51.     procedure Asignar(Elementos: TBuffDatos);
52.     function getCopia: TBuffDatos;
53.     property Elemento[I: Integer]:Byte read getElemento write setElemento;
54.     property Count: Integer read FCount;
55.   end;
56.
57.
58.   TThreadLectura = class(TThread)
59.   private
60.     Conectado: Boolean;
61.     Conectar: Boolean;
62.     BufferDatos: TBuffDatos;
63.     Vivo: Boolean;
64.     Bloquear: Boolean;
65.     UsandoBuffer: Boolean;
66.     BaudRate: DWORD;
67.     Puerto: String;
68.     ComandoEnEspera: TComando;
69.     ComandoFlotante: Boolean;
70.     EnviandoComando: Boolean;
71.     ValorSaliente: array[0..4] of Byte;
72.   protected
73.     procedure Execute; override;
74.     procedure CargarEntero(Valor: LongWord);
75.     procedure CargarFlotante(Valor: Single);
76.   public
77.     Conexion: TCommPortDriver;
78.     constructor Create;
79.     destructor Destroy; override;
80.     procedure Terminar;
81.     procedure Conectarse;
82.     procedure Desconectar;
```

```

83.     function getBufferDatos: TBuffDatos;
84.     procedure SetPuerto(Valor: string);
85.     procedure SetBaudRate(Valor: DWORD);
86.     procedure Configurar(AOwner: TComponent);
87.     function EnviarComando(Comando: TComando; Flotante: Boolean): Boolean;
88. end;
89.
90. TForm1 = class(TForm)
91.     GroupBox1: TGroupBox;
92.     B_conectar: TButton;
93.     B_desconectar: TButton;
94.     Button2: TButton;
95.     PlayPIP: TMediaPlayer;
96.     SaveDialog: TFileSaveDialog;
97.     Label1: TLabel;
98.     PortComboBox: TComboBox;
99.     BaudRateComboBox: TComboBox;
100.    Label2: TLabel;
101.    Actualizador: TTimer;
102.    GroupBox2: TGroupBox;
103.    EditCodigo: TLabeledEdit;
104.    EditValor: TLabeledEdit;
105.    BotonEnviar: TButton;
106.    Button3: TButton;
107.    ComboComando: TComboBox;
108.    GroupBox3: TGroupBox;
109.    HistorialAcciones: TMemo;
110.    Interfaz: TGroupBox;
111.    LCD_L1: TLCDLine;
112.    LCD_L2: TLCDLine;
113.    Button1: TButton;
114.    Button4: TButton;
115.    GroupBox4: TGroupBox;
116.    Label_gyro: TLabel;
117.    Label_inc: TLabel;
118.    VisorGyro: TA3nalogGauge;
119.    VisorInc: TA3nalogGauge;
120.    Label3: TLabel;
121.    Label4: TLabel;
122.    B_Capturar: TButton;
123.    Boton_guardar: TButton;
124.    Combo_Flujo: TComboBox;
125.    TrCorrienteMI: TTrackBar;
126.    L_corriente_MI: TLabel;
127.    TrCorrienteMD: TTrackBar;
128.    L_corriente_MD: TLabel;
129.    TrDireccion: TTrackBar;
130.    LDireccion: TLabel;
131.    TrBateria: TTrackBar;
132.    LBateria: TLabel;
133.    GroupBox5: TGroupBox;
134.    TrControl: TTrackBar;
135.    Label5: TLabel;
136.    procedure Button2Click(Sender: TObject);
137.    procedure FormCreate(Sender: TObject);
138.    procedure FormShow(Sender: TObject);
139.    procedure B_conectarClick(Sender: TObject);
140.    procedure B_desconectarClick(Sender: TObject);
141.    procedure PortComboBoxChange(Sender: TObject);
142.    procedure BaudRateComboBoxChange(Sender: TObject);
143.    procedure FormDestroy(Sender: TObject);
144.    procedure ActualizadorTimer(Sender: TObject);
145.    procedure BotonEnviarClick(Sender: TObject);
146.    procedure Button3Click(Sender: TObject);
147.    procedure FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
148.    procedure EditValorKeyPress(Sender: TObject; var Key: Char);
149.    procedure EditCodigoKeyPress(Sender: TObject; var Key: Char);
150.    procedure B_CapturarClick(Sender: TObject);
151.    procedure Boton_guardarClick(Sender: TObject);
152. private
153.     { Private declarations }
154. public
155.     { Public declarations }
156. end;
157.
158. var
159.     Form1: TForm1;
160.     Con_Serial: TThreadLectura;
161.     Datos_Recibidos: TBuffDatos;
162.     Comandos_Recibidos: TBuffComandos;
163.     ID_Enviado: Integer;
164.     Datos_flujo: TBuffComandos;
165.     Capturando: Boolean;
166.

```

```

167. implementation
168.
169. {$R *.dfm}
170.
171. uses
172.     SettingsDlg;
173.
174. //Métodos de la clase TBuffComandos
175. constructor TBuffComandos.Create;
176. begin
177.     inherited Create;
178.     FCount := 0;
179.     SetLength(FBuffer, FCount);
180. end;
181.
182. destructor TBuffComandos.Destroy;
183. begin
184.     FCount := 0;
185.     SetLength(FBuffer, 0);
186.     inherited Destroy;
187. end;
188.
189. function TBuffComandos.getElemento(Index: Integer): TComando;
190. begin
191.     Result := FBuffer[Index];
192. end;
193.
194. procedure TBuffComandos.setElemento(Index: Integer; const Value: TComando);
195. begin
196.     FBuffer[Index] := Value;
197. end;
198.
199. procedure TBuffComandos.Add(iElemento: TComando);
200. begin
201.     FCount := FCount + 1;
202.     SetLength(FBuffer, FCount);
203.     FBuffer[FCount - 1] := iElemento;
204. end;
205.
206. procedure TBuffComandos.Delete(Index: Integer);
207. var
208.     i: Integer;
209. begin
210.     for i:= Index + 1 to FCount - 1 do
211.         FBuffer[i-1] := FBuffer[i];
212.     FCount := FCount - 1;
213.     SetLength(FBuffer, FCount);
214. end;
215.
216. procedure TBuffComandos.Insertar(Index: Integer; iElemento: TComando);
217. var
218.     i: Integer;
219. begin
220.     FCount := FCount + 1;
221.     SetLength(FBuffer, FCount);
222.     for i:= FCount - 1 downto Index + 1 do
223.         FBuffer[i] := FBuffer[i-1];
224.     FBuffer[Index] := iElemento;
225. end;
226.
227. procedure TBuffComandos.Clear;
228. begin
229.     FCount := 0;
230.     SetLength(FBuffer, 0);
231. end;
232.
233. procedure TBuffComandos.Asignar(Elementos: TBuffComandos);
234. var
235.     I: Integer;
236. begin
237.     if not Assigned(Elementos) then
238.         Abort;
239.     FCount := Elementos.Count;
240.     SetLength(FBuffer, FCount);
241.     for I:= 0 to Elementos.Count - 1 do
242.         FBuffer[I] := Elementos.Elemento[I];
243. end;
244.
245. function TBuffComandos.getCopia: TBuffComandos;
246. var
247.     I: Integer;
248.     R: TBuffComandos;
249. begin
250.     R := TBuffComandos.Create;

```

```

251.     for I:= 0 to FCount - 1 do
252.         R.Add(FBuffer[I]);
253.     Result := R;
254. end;
255.
256. //Métodos de la clase TBuffDatos
257. constructor TBuffDatos.Create;
258. begin
259.     inherited Create;
260.     FCount := 0;
261.     SetLength(FBuffer, FCount);
262. end;
263.
264. destructor TBuffDatos.Destroy;
265. begin
266.     FCount := 0;
267.     SetLength(FBuffer, 0);
268.     inherited Destroy;
269. end;
270.
271. function TBuffDatos.getElemento(Index: Integer): Byte;
272. begin
273.     Result := FBuffer[Index];
274. end;
275.
276. procedure TBuffDatos.setElemento(Index: Integer; const Value: Byte);
277. begin
278.     FBuffer[Index] := Value;
279. end;
280.
281. procedure TBuffDatos.Add(iElemento: Byte);
282. begin
283.     FCount := FCount + 1;
284.     SetLength(FBuffer, FCount);
285.     FBuffer[FCount - 1] := iElemento;
286. end;
287.
288. procedure TBuffDatos.Delete(Index: Integer);
289. var
290.     i: Integer;
291. begin
292.     for i:= Index + 1 to FCount - 1 do
293.         FBuffer[i-1] := FBuffer[i];
294.     FCount := FCount - 1;
295.     SetLength(FBuffer, FCount);
296. end;
297.
298. procedure TBuffDatos.Insertar(Index: Integer; iElemento: Byte);
299. var
300.     i: Integer;
301. begin
302.     FCount := FCount + 1;
303.     SetLength(FBuffer, FCount);
304.     for i:= FCount - 1 downto Index + 1 do
305.         FBuffer[i] := FBuffer[i-1];
306.     FBuffer[Index] := iElemento;
307. end;
308.
309. procedure TBuffDatos.Clear;
310. begin
311.     FCount := 0;
312.     SetLength(FBuffer, 0);
313. end;
314.
315. procedure TBuffDatos.Asignar(Elementos: TBuffDatos);
316. var
317.     I: Integer;
318. begin
319.     if not Assigned(Elementos) then
320.         Abort;
321.     FCount := Elementos.Count;
322.     SetLength(FBuffer, FCount);
323.     for I:= 0 to Elementos.Count - 1 do
324.         FBuffer[I] := Elementos.Elemento[I];
325. end;
326.
327. function TBuffDatos.getCopia: TBuffDatos;
328. var
329.     I: Integer;
330.     R: TBuffDatos;
331. begin
332.     R := TBuffDatos.Create;
333.     for I:= 0 to FCount - 1 do
334.         R.Add(FBuffer[I]);

```

```

335.     Result := R;
336. end;
337.
338. //Métodos de la clase TThreadLectura
339. constructor TThreadLectura.Create;
340. begin
341.     inherited Create(True);
342.     Conexion := TCommPortDriver.Create(nil);
343.     Conexion.PortName := '\\.\COM4';
344.     Conexion.BaudRateValue := 19200;
345.     BaudRate := Conexion.BaudRateValue;
346.     Puerto := Conexion.PortName;
347.     Conexion.PollingDelay := 5;
348.     Conexion.InputTimeout := 500;
349.     Conectado := False;
350.     Conectar := False;
351.     BufferDatos := TBuffDatos.Create;
352.     Vivo := True;
353.     EnviandoComando := False;
354.     Bloquear := False;
355.     UsandoBuffer := False;
356.     Priority := tpIdle;
357.     FreeOnTerminate := True;
358.     Suspended := False;
359. end;
360.
361. destructor TThreadLectura.Destroy;
362. begin
363.     Conexion.Disconnect;
364.     Conexion.Free;
365.     BufferDatos.Free;
366. end;
367.
368. procedure TThreadLectura.Execute;
369. var
370.     dato: Byte;
371. begin
372.     while Vivo do
373.     begin
374.         Sleep(1);
375.         if Conectado then
376.         begin
377.             //CODIGO SI ESTOY CONECTADO
378.
379.             if EnviandoComando then
380.             begin
381.                 if ComandoFlotante then
382.                     CargarFlotante(ComandoEnEspera.Valor)
383.                 else
384.                     CargarEntero(ComandoEnEspera.Valor_int);
385.
386.                 Conexion.SendByte(ComandoEnEspera.Codigo);
387.                 Conexion.SendByte(ComandoEnEspera.ID);
388.                 Conexion.SendByte(ValorSaliente[0]);
389.                 Conexion.SendByte(ValorSaliente[1]);
390.                 Conexion.SendByte(ValorSaliente[2]);
391.                 Conexion.SendByte(ValorSaliente[3]);
392.
393.                 EnviandoComando := False;
394.             end;
395.
396.             while Bloquear do
397.             begin
398.                 //no hacer nada
399.             end;
400.
401.             UsandoBuffer := True;           // Bloquea el Buffer
402.             while (Conexion.CountRX > 0) and (Conexion.CountRX < 65535) do
403.             begin
404.                 if Conexion.ReadByte(dato) then //Si se leyó un byte se agrega
405.                     BufferDatos.Add(dato);
406.                 end;
407.                 UsandoBuffer := False;     // Desbloquea el Buffer
408.
409.                 if Conectado = False then
410.                     Conexion.Disconnect;
411.                 end else
412.                 begin
413.                     //CODIGO SI NO ESTOY CONECTADO
414.                     Conexion.BaudRateValue := BaudRate;
415.                     Conexion.PortName := Puerto;
416.                     if Conectar then
417.                     begin
418.                         Conectado := Conexion.Connect;
419.                         Conectar := False;

```

```

419.         end;
420.     end;
421. end;
422. end;
423.
424. procedure TThreadLectura.CargarEntero(Valor: LongWord);
425. var
426.     mascara: LongWord;
427. begin
428.     mascara := 255;
429.
430.     ValorSaliente[3] := Valor and mascara;
431.     ValorSaliente[2] := (Valor shr 8) and mascara;
432.     ValorSaliente[1] := (Valor shr 16) and mascara;
433.     ValorSaliente[0] := (Valor shr 24) and mascara;
434. end;
435.
436. procedure TThreadLectura.CargarFlotante(Valor: Single);
437. var
438.     Puntero: Pointer;
439.     Numero: ^Cardinal;
440.     Mascara: Cardinal;
441. begin
442.     Mascara := 255; //0x000000FF
443.     Puntero := @Valor;
444.     Numero := Puntero;
445.
446.     ValorSaliente[3] := Numero^ and Mascara;
447.     ValorSaliente[2] := (Numero^ shr 8) and Mascara;
448.     ValorSaliente[1] := (Numero^ shr 16) and Mascara;
449.     ValorSaliente[0] := (Numero^ shr 24) and Mascara;
450. end;
451.
452. function TThreadLectura.EnviaComando(Comando: TComando; Flotante: Boolean): Boolean;
453. begin
454.     if not EnviandoComando then
455.     begin
456.         ComandoEnEspera := Comando;
457.         ComandoFlotante := Flotante;
458.         EnviandoComando := True;
459.         Result := True;
460.     end else Result := False;
461. end;
462.
463. procedure TThreadLectura.Terminar;
464. begin
465.     Vivo := False;
466. end;
467.
468. procedure TThreadLectura.Conectarse;
469. begin
470.     BufferDatos.Clear;
471.     Conectar := True;
472. end;
473.
474. procedure TThreadLectura.Desconectar;
475. begin
476.     Conectado := False;
477. end;
478.
479. function TThreadLectura.getBufferDatos: TBuffDatos;
480. var
481.     Resultado: TBuffDatos;
482. begin
483.     while UsandoBuffer do
484.     begin
485.         // Espera a que el Buffer se desocupe
486.     end;
487.     Bloquear := True;
488.     Resultado := BufferDatos.getCopia;
489.     BufferDatos.Clear;
490.     Bloquear := False;
491.     Result := Resultado;
492. end;
493.
494. procedure TThreadLectura.SetPuerto(Valor: String);
495. begin
496.     Puerto := Valor;
497. end;
498.
499. procedure TThreadLectura.SetBaudRate(Valor: DWORD);
500. begin
501.     BaudRate := Valor;
502. end;

```

```

503.
504. procedure TThreadLectura.Configurar(AOwner: TComponent);
505. var
506.   dlg: TSettingsForm;
507. begin
508.   // Tell the user we cannot change settings while a connection is active
509.   if Conexion.Connected then
510.     begin
511.       if Application.MessageBox( 'No se pueden cambiar los parámetros si la conexión está activa.'#13#10+
512.         '¿Cerrar la conexión y continuar?',
513.           'Confirmación',
514.             MB_OKCANCEL or MB_ICONQUESTION ) <> ID_OK then
515.         exit;
516.     end;
517.   // Let the user to customize settings
518.   dlg := nil;
519.   try
520.     dlg := TSettingsForm.Create( AOwner, Conexion );
521.     dlg.ShowModal;
522.   finally
523.     dlg.Free;
524.   end;
525. end;
526.
527.
528. function ByteToFloat(B1, B2, B3, B4: Cardinal): Single;
529. var
530.   Num: Cardinal; //UInt32
531.   Puntero: Pointer;
532.   Resultado: ^Single;
533. begin
534.   Num := B1 Shl 24;
535.   Num := Num + (B2 Shl 16);
536.   Num := Num + (B3 Shl 8);
537.   Num := Num + B4;
538.
539.   Puntero := @Num;
540.   Resultado := Puntero;
541.   Result := Resultado^;
542. end;
543.
544. function ByteToGyro(B1, B2, B3, B4: Cardinal): Integer;
545. var
546.   Num: Cardinal; //UInt32
547.   Puntero: Pointer;
548.   Resultado: ^Integer;
549.   Res: Integer;
550. begin
551.   Num := B1 Shl 24;
552.   Num := Num + (B2 Shl 16);
553.   Num := Num + (B3 Shl 8);
554.   Num := Num + B4;
555.
556.   Puntero := @Num;
557.   Resultado := Puntero;
558.   Res := Resultado^;
559.
560.   //Conversión si es que es negativo
561.   if Res > 8191 then
562.     Res := (16384 - Res)*-1;
563.   Result := Res;
564. end;
565.
566.
567. procedure TForm1.ActualizadorTimer(Sender: TObject);
568. var
569.   Comando: TComando;
570.   Informacion: TBuffDatos;
571.   J: Integer;
572.   Flotante: Double;
573.   S: String;
574. begin
575.   { Datos_Recibidos: TBuffDatos;
576.     Comandos_Recibidos: TBuffComandos;}
577.
578.   Informacion := Con_Serial.getBufferDatos;
579.
580.   for J := 0 to Informacion.Count - 1 do
581.     begin
582.       Datos_Recibidos.Add(Informacion.Elemento[J]);
583.     end;
584.
585.   while Datos_Recibidos.Count > 6 do
586.     begin

```



```

587.   if Datos_Recibidos.Elemento[6] = 255 then
588.   begin
589.       //Encontré un comando, lo agrego
590.       Comando.Codigo := Datos_Recibidos.Elemento[0];
591.       Comando.ID := Datos_Recibidos.Elemento[1];
592.
593.       {Comando.Valor := ByteToFloat(Datos_Recibidos.Elemento[2],
594.                                     Datos_Recibidos.Elemento[3],
595.                                     Datos_Recibidos.Elemento[4],
596.                                     Datos_Recibidos.Elemento[5]);}
597.       Comando.Valor_int := ByteToGyro(Datos_Recibidos.Elemento[2],
598.                                       Datos_Recibidos.Elemento[3],
599.                                       Datos_Recibidos.Elemento[4],
600.                                       Datos_Recibidos.Elemento[5]);
601.       Comandos_Recibidos.Add(Comando);
602.       S := '-> COD='+IntToStr(Comando.Codigo)+' ID='+IntToStr(Comando.ID);
603.       S := S + ' VAL='+ IntToStr(Comando.Valor_int);
604.       for J := 1 to 7 do
605.           Datos_Recibidos.Delete(0);
606.       end else
607.           Datos_Recibidos.Delete(0);
608.   end;
609.
610.   //Ahora decido qué hacer con los comandos
611.   for J := 0 to Comandos_Recibidos.Count - 1 do
612.   begin
613.       //Código de captura para cuando está activa
614.       if Capturando then
615.       begin
616.           Datos_flujo.Add(Comandos_Recibidos.Elemento[J]);
617.       end;
618.
619.       //192 -> Recibe Ping
620.       if Comandos_Recibidos.Elemento[J].Codigo = 192 then
621.       begin
622.           HistorialAcciones.Lines.Add(S);
623.       end;
624.       //134 -> Recibe Dato de Giróscopo (flujo)
625.       if Comandos_Recibidos.Elemento[J].Codigo = 134 then
626.       begin
627.           Flotante := Comandos_Recibidos.Elemento[J].Valor_int - 2000000000.0;
628.           Flotante := Flotante / 100000.0;
629.           Label_gyro.Caption := FloatToStr(Flotante) + '°/s';
630.           VisorGyro.Position := Round((Flotante + 90)*10);
631.           //Registro.Add('GY->'+IntToStr(Comandos_Recibidos.Elemento[J].Valor_int));
632.       end;
633.       //133 -> Recibe Dato de Inclinometro (flujo)
634.       if Comandos_Recibidos.Elemento[J].Codigo = 133 then
635.       begin
636.           Flotante := Comandos_Recibidos.Elemento[J].Valor_int - 2000000000.0;
637.           Flotante := Flotante / 100000.0;
638.           Label_inc.Caption := FloatToStr(Flotante) + '°';
639.           VisorInc.Position := Round((Flotante + 45)*10);
640.           // Registro.Add('IN->'+IntToStr(Comandos_Recibidos.Elemento[J].Valor_int));
641.       end;
642.       //135 -> Recibe Dato de Corriente Motor Derecho
643.       if Comandos_Recibidos.Elemento[J].Codigo = 135 then
644.       begin
645.           L_corriente_MD.Caption := 'Motor Derecho: ' + FloatToStr((Comandos_Recibidos.Elemento[J].Valor_int
- 1999983616)/100.0) + ' A';
646.           TrCorrienteMD.Position := 50000 + (Comandos_Recibidos.Elemento[J].Valor_int - 1999983616);
647.       end;
648.       //136 -> Recibe Dato de Corriente Motor Izquierdo
649.       if Comandos_Recibidos.Elemento[J].Codigo = 136 then
650.       begin
651.           L_corriente_MI.Caption := 'Motor Izquierdo: ' +
FloatToStr((Comandos_Recibidos.Elemento[J].Valor_int - 1999983616)/100.0) + ' A';
652.           TrCorrienteMI.Position := 5000 + (Comandos_Recibidos.Elemento[J].Valor_int - 1999983616);
653.       end;
654.       //138 -> Recibe Dato Dirección
655.       if Comandos_Recibidos.Elemento[J].Codigo = 138 then
656.       begin
657.           LDireccion.Caption := 'Dirección: ' + IntToStr(Comandos_Recibidos.Elemento[J].Valor_int -
1999983616);
658.           TrDireccion.Position := 250 - (Comandos_Recibidos.Elemento[J].Valor_int - 1999983616);
659.       end;
660.       //139 -> Recibe Dato Batería
661.       if Comandos_Recibidos.Elemento[J].Codigo = 139 then
662.       begin
663.           LBateria.Caption := 'Batería: ' + FloatToStr((Comandos_Recibidos.Elemento[J].Valor_int -
1999983616)/10.0) + ' V';
664.           TrBateria.Position := Comandos_Recibidos.Elemento[J].Valor_int - 1999983616;
665.       end;
666.       //141 -> Recibe Dato Control

```

```

667.     if Comandos_Recibidos.Elemento[J].Codigo = 141 then
668.     begin
669.         TrControl.Position := 6000 + Comandos_Recibidos.Elemento[J].Valor_int - 1999983616;
670.         Label5.Caption := IntToStr(Comandos_Recibidos.Elemento[J].Valor_int - 1999983616);
671.     end;
672.
673.     end;
674.     Comandos_Recibidos.Clear;
675.
676. end;
677.
678. procedure TForm1.BaudRateComboBoxChange(Sender: TObject);
679. begin
680.     Con_Serial.BaudRate := StrToInt(BaudRateComboBox.Text);
681. end;
682.
683. procedure TForm1.BotonEnviarClick(Sender: TObject);
684. var
685.     Comando: TComando;
686.     ComandoFlotante: Boolean;
687.     S: String;
688. begin
689.     if Capturando then
690.     begin
691.         Capturando := False;
692.         HistorialAcciones.Lines.Add('-- Fin captura');
693.         B_Capturar.Caption := 'Capturar';
694.     end;
695.
696.     ID_Enviado := ID_Enviado + 1;
697.     if ID_Enviado > 255 then
698.         ID_Enviado := 1;
699.
700.     Comando.Codigo := StrToInt(EditCodigo.Text);
701.     Comando.ID := ID_Enviado;
702.     Comando.Valor_int := StrToInt(EditValor.Text);
703.     ComandoFlotante := False;
704.
705.     S := '<- COD='+IntToStr(Comando.Codigo)+' ID='+IntToStr(Comando.ID)+' VAL='+ EditValor.Text;
706.     S := S + ' STATUS=';
707.
708.     if Con_Serial.EnviaComando(Comando, ComandoFlotante) then
709.         S := S + 'OK'
710.     else S := S + 'ERROR';
711.     HistorialAcciones.Lines.Add(S);
712. end;
713.
714. procedure TForm1.Boton_guardarClick(Sender: TObject);
715. var
716.     Salida, sTiempo, sInc, sGiro: TStrings;
717.     I, I_esperado: Integer;
718.     Tiempo, dt, Valor: Double;
719.     S: String;
720.
721. procedure interpolar(var lista: TStrings);
722. var
723.     Val, vAnt, vSig: Double;
724.     I, J, Pasos: Integer;
725. begin
726.     Pasos := 0;
727.     vAnt := 1000000000000;
728.     vSig := 1000000000000;
729.     for I := 0 to lista.Count - 1 do
730.     begin
731.         if lista.Strings[I] = '--' then
732.         begin
733.             if vAnt = 1000000000000 then
734.                 lista.Strings[I] := '0'
735.             else Pasos := Pasos + 1;
736.         end else
737.         begin
738.             if vAnt = 1000000000000 then
739.                 vAnt := StrToFloat(lista.Strings[I]);
740.
741.             if Pasos > 0 then
742.             begin
743.                 vSig := StrToFloat(lista.Strings[I]);
744.                 Val := (vSig - vAnt)/(Pasos + 1);
745.
746.                 for J := I - Pasos to I - 1 do
747.                 begin
748.                     lista.Strings[J] := FloatToStr(StrToFloat(lista.Strings[J-1]) + Val);
749.                 end;
750.                 Pasos := 0;

```

```

751.         vAnt := StrToFloat(lista.Strings[I]);
752.     end else vAnt := StrToFloat(lista.Strings[I]);
753.
754.     end;
755. end;
756. end;
757.
758. begin
759.     Salida := TStringList.Create;
760.     sTiempo := TStringList.Create;
761.     sInc := TStringList.Create;
762.     sGiro := TStringList.Create;
763.
764.     Tiempo := 0;
765.     dt := 0.01;
766.
767.     I_esperado := 1;
768.     for I := 0 to Datos_flujo.Count - 1 do
769.     begin
770.         //chr(9)
771.         Tiempo := Tiempo + dt;
772.
773.         while Datos_flujo.Elemento[I].ID <> I_esperado do
774.         begin
775.             I_esperado := I_esperado + 1;
776.             if I_esperado > 255 then
777.                 I_esperado := 0;
778.             Tiempo := Tiempo + dt;
779.         end;
780.
781.         if Datos_flujo.Elemento[I].Codigo = 133 then //Inclinometro
782.         begin
783.             Valor := Datos_flujo.Elemento[I].Valor_int - 2000000000.0;
784.             Valor := Valor / 100000.0;
785.             sInc.Add(FloatToStr(Valor));
786.         end else sInc.Add('--');
787.
788.         if Datos_flujo.Elemento[I].Codigo = 134 then //Gir6scopo
789.         begin
790.             Valor := Datos_flujo.Elemento[I].Valor_int - 2000000000.0;
791.             Valor := Valor / 100000.0;
792.             sGiro.Add(FloatToStr(Valor));
793.         end else sGiro.Add('--');
794.
795.         sTiempo.Add(FloatToStr(Tiempo));
796.
797.         I_esperado := I_esperado + 1;
798.         if I_esperado > 255 then
799.             I_esperado := 0;
800.     end;
801.
802.     interpolar(sInc);
803.     interpolar(sGiro);
804.     for I := 0 to sTiempo.Count - 1 do
805.     begin
806.         Salida.Add(sTiempo.Strings[I] + chr(9) + sInc.Strings[I] + chr(9) + sGiro.Strings[I]);
807.     end;
808.
809.     if SaveDialog.Execute then
810.         Salida.SaveToFile(SaveDialog.FileName);
811.
812.
813.     Salida.Free;
814.     sTiempo.Free;
815.     sInc.Free;
816.     sGiro.Free;
817. end;
818.
819. procedure TForm1.Button2Click(Sender: TObject);
820. begin
821.     Application.Terminate;
822. end;
823.
824. procedure TForm1.Button3Click(Sender: TObject);
825. begin
826.     HistorialAcciones.Lines.Clear;
827. end;
828.
829. procedure TForm1.B_CapturarClick(Sender: TObject);
830. var
831.     Comando: TComando;
832.     S: String;
833. begin
834.     {Datos IMU

```

```

835. Inclinación
836. Giróscopos
837. Corriente Motor Derecho
838. Corriente Motor Izquierdo)
839.
840.   if TButton(Sender).Caption = 'Capturar' then
841.   begin
842.     if not Datos_flujo.Count > 0 then
843.     begin
844.       if Application.MessageBox( 'La última captura de datos no ha sido guardada.'#13#10+
845.         '¿Desea continuar sin guardar?',
846.         'Pregunta',
847.         MB_YESNO or MB_ICONQUESTION ) <> ID_YES then
848.         exit;
849.     end;
850.
851.     Case Combo_Flujo.ItemIndex of
852.       0 : Comando.Codigo := 137; //C_FLUJO_IMU
853.       1 : Comando.Codigo := 133; //C_FLUJO_INCLINOMETROS
854.       2 : Comando.Codigo := 134; //C_FLUJO_GIROSCOPOS
855.       3 : Comando.Codigo := 135; //C_FLUJO_CORRIENTE_MD
856.       4 : Comando.Codigo := 136; //C_FLUJO_CORRIENTE_MI
857.     else Abort;
858.     end;
859.
860.     Comando.ID := 1;
861.     Comando.Valor_int := 0;
862.     S := '<- Capturar ' + Combo_Flujo.Text + ' ST=';
863.     if Con_Serial.EnviaComando(Comando, False) then
864.     begin
865.       S := S + 'OK';
866.       Datos_flujo.Clear;
867.       Capturando := True;
868.       TButton(Sender).Caption := 'Detener';
869.       PlayPIP.Play;
870.     end else S := S + 'ERROR';
871.     HistorialAcciones.Lines.Add(S);
872.   end else
873.   begin
874.     Comando.Codigo := 192; //Ping DSP
875.     Con_Serial.EnviaComando(Comando, False);
876.     Capturando := False;
877.     HistorialAcciones.Lines.Add('-- Fin captura');
878.     TButton(Sender).Caption := 'Capturar';
879.   end;
880. end;
881.
882. procedure TForm1.B_conectarClick(Sender: TObject);
883. var
884.   Espera: Integer;
885. begin
886.   Espera := 0;
887.   Con_Serial.Conectarse;
888.   while Con_Serial.Conectado = False do
889.   begin
890.     Sleep(1);
891.     Espera := Espera + 1;
892.     if Espera > 500 then
893.     begin
894.       Application.MessageBox( 'No se ha conseguido conectar',
895.         'Error',
896.         MB_OK or MB_ICONERROR );
897.       Exit;
898.     end;
899.   end;
900.   B_Desconectar.Enabled := True;
901.   B_Conectar.Enabled := False;
902. end;
903.
904. procedure TForm1.B_desconectarClick(Sender: TObject);
905. begin
906.   Con_Serial.Desconectar;
907.   B_Desconectar.Enabled := False;
908.   B_Conectar.Enabled := True;
909. end;
910.
911. procedure TForm1.EditCodigoKeyPress(Sender: TObject; var Key: Char);
912. begin
913.   if (StrScan('0123456789',Key) <> nil) or ( Key = Char(VK_BACK) ) then
914.     else Key := #0;
915. end;
916.
917. procedure TForm1.EditValorKeyPress(Sender: TObject; var Key: Char);
918. begin

```

```

919.     if (StrScan('0123456789',Key) <> nil) or ( Key = Char(VK_BACK) ) then
920.         else Key := #0;
921.     end;
922.
923. procedure TForm1.FormCreate(Sender: TObject);
924. begin
925.     Con_Serial := TThreadLectura.Create;
926.     Datos_Recibidos := TBufDatos.Create;
927.     Comandos_Recibidos := TBufComandos.Create;
928.     Datos_flujo := TBufComandos.Create;
929.     ID_Enviado := 0;
930.     Capturando := False;
931. end;
932.
933. procedure TForm1.FormDestroy(Sender: TObject);
934. begin
935.     Datos_Recibidos.Free;
936.     Comandos_Recibidos.Free;
937.     Datos_flujo.Free;
938. end;
939.
940. procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
941.     Shift: TShiftState);
942. var
943.     Comando: TComando;
944. begin
945.     if Key = 27 then //ESCAPE
946.     begin
947.         if Capturando then
948.         begin
949.             Capturando := False;
950.             HistorialAcciones.Lines.Add('-- Fin captura');
951.             B_Capturar.Caption := 'Capturar';
952.         end;
953.
954.         Comando.Codigo := 193; //TRIP_MOTOR
955.         ID_Enviado := ID_Enviado + 1;
956.         if ID_Enviado > 255 then
957.             ID_Enviado := 1;
958.
959.         Comando.Valor := 0;
960.         Comando.ID := ID_Enviado;
961.         Con_Serial.EnviaComando(Comando, False);
962.     end;
963.
964.     if Key = 85 then //U
965.     begin
966.         if Capturando then
967.         begin
968.             Capturando := False;
969.             HistorialAcciones.Lines.Add('-- Fin captura');
970.             B_Capturar.Caption := 'Capturar';
971.         end;
972.
973.         Comando.Codigo := 194; //C_UNLOCK_MOTOR
974.         ID_Enviado := ID_Enviado + 1;
975.         if ID_Enviado > 255 then
976.             ID_Enviado := 1;
977.
978.         Comando.Valor := 0;
979.         Comando.ID := ID_Enviado;
980.         Con_Serial.EnviaComando(Comando, False);
981.     end;
982.
983.
984. end;
985.
986. procedure TForm1.FormShow(Sender: TObject);
987. begin
988.     PortComboBox.Text := Con_Serial.Conexion.PortName;
989.     BaudRateComboBox.Text := IntToStr(Con_Serial.Conexion.BaudRateValue);
990. end;
991.
992. procedure TForm1.PortComboBoxChange(Sender: TObject);
993. begin
994.     Con_Serial.Puerto := PortComboBox.Text;
995. end;
996.
997. end.

```

Anexo D: Material en formato digital

Se adjunta un DVD, el cual contiene el siguiente material:

- Copia digital de este documento.
- Presentación del Examen de Grado.
- Circuitos esquemáticos y diseño en CAD de las tarjetas electrónicas diseñadas.
- Códigos fuente de las diferentes plataformas en formato de proyecto compilable.
- Hojas de datos de los componentes utilizados.
- Fotografías y videos del vehículo en diversos estados de avance.